Introduction

The challenges during this hackathon assumes no previous experience of Flogo and Spotfire. All you need is your laptop and some basic computer skills.

To begin with, we will demonstrate 2 things to you

1) A working Flogo application based on the story you will read below in the document
2) A Spotfire demonstration and it's ease of use.

After both these demonstrations, we will use this document as a guide to provide step by step instructions to understanding the scenario and executing the challenges.

By the time we finish the workshop, you will have a good understanding of what Flogo and Spotfire is all about. You will have this handbook that you can use as a guide to follow. Should be pretty simple. We will be around to help you as well.

At some point, we will leave you with just this guide to follow along and a data set for Spotfire that you can use for building your own visualizations.You will work as a team and use any of your skills in combination with what you have learnt during the workshop. You can utilize anything that you have learnt today, whether it's using a Arduino kit, Estimote beacons, touch board kits in combination with Flogo and Spotfire.

As the hackathon is a team event, you will be working with a team and scoring is determined by ease of consumption and explanations and **creative out of the box thinking**. More details are on http://teenhacksli.com and will also be explained by the organizers.

You will have the opportunity to present it to the judges and walk away with prizes.

The Challenge

What does Geoff's Logistics want?

Geoff's International and Regional Logistics (**GIRL**) wants to **protect its working truck drivers** and also ensure that they are **not a danger to themselves or other road users.**

To achieve this, they have installed a Motion Understanding Mechanism (**MUM**), an **On-Board IoT device to report statistics** about the location of the truck and it's journey.

One feature of the device is Driver Attention Detection (**DAD**). DAD uses a Machine Learning (ML) model to locally process the image of the driver to ensure that he/she is not falling asleep and paying attention to the road and not distracted by anything else, for example no mobile phone use.

The On-Board-Unit (OBU) in the truck **sends data every second on two channels**. The first includes **location, speed and fuel levels**, the second comes from the **DAD** and includes the **attention data**.

The **driver attention data needs to be further evaluated on a serverless platform** alongside the additional data from the OBU using ML inference logic. An **AWS Lambda function** provided to you will evaluate if the driver is reaching their attention limit.

The **app should then suggest to stop at the closest rest point**, based on the current position (lat, long) from a third party geolocation system.

Work with us as we guide you step by step to build an app for **GIRL** that will achieve what they are looking for.

Follow along as we take you through a ride while helping you understand serverless architecture, AWS Lambda, MQTT , API's, Flogo, TIBCO Cloud and many other things.

At some point we will leave you just with this guide for you to follow on your own. You will work with your team to enhance what you have already accomplished. If you have the hands-on skills to build something, it's great - otherwise we are happy to hear your ideas too as a team. The

judges would love to hear your ideas and see what you have. This document has several suggestions with examples for you to try.

This is your chance to enhance the story further and show us what you can do with the data. We have several tools for you to use brainstorm with your team. Geoff's International welcomes innovations of all kinds.

Be creative! Think data, think Spotfire, think mobile apps , think notifications , think Amazon Aurora, think serverless, think Flogo!!... There are endless options !!

At the end of the hackathon, judging will begin. Present the ideas and solution to the judges, be prepared to amaze them and win a prize!

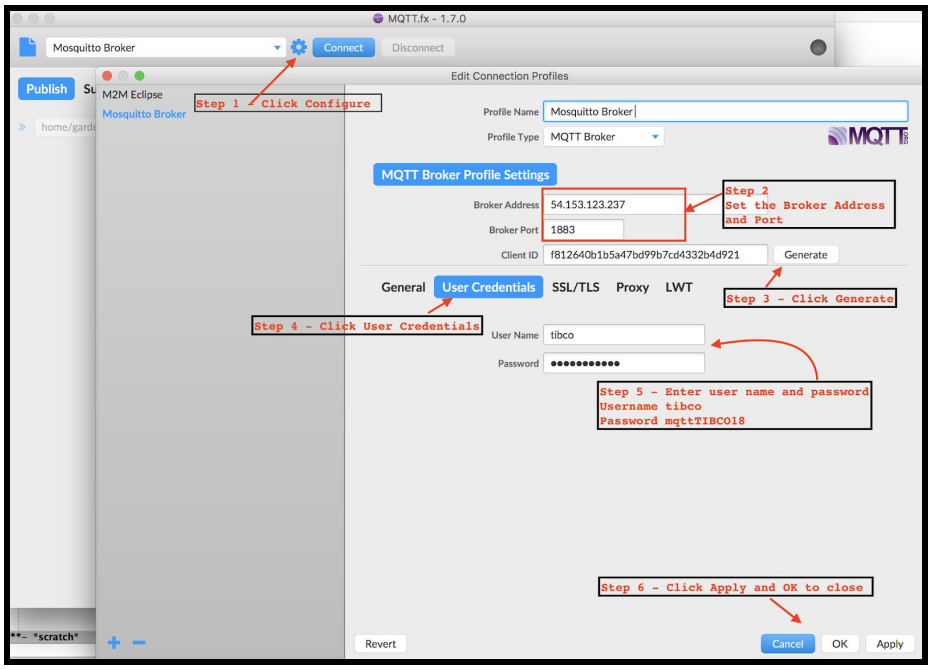
Installing Docker

Your first step is to make sure you install Docker on your machine. Start by going to https://www.docker.com/get-started and downloading the relevant binary for your OS. If you already have docker installed you can skip this step.

Installing MQTT.fx

Download MQTT.fx from https://mqttfx.jensd.de/index.php
On MacOS launch the application MQTT.fx from your applications. On windows find the program and launch it. On linux, the main folder is found in /opt/MQTTfx. You do not need to be in "sudo su" to run the jar file. Just double-click on this to open the tool.

Follow the instructions to configure the tool.



Click on the configure button to launch the config screen.
Set the profile name to anything you like and profile type to MQTT Broker.
Set the Broker Address to "**54.153.123.237**" and Broker port to "**1883**" without the quotes.
Click on Generate to generate a new client id. This client id will be used to identify and and manage your connection in the MQTT broker. The MQTT broker has needs to be provided with credentials to connect. Click on the "User Credentials" and type in the user name and password. The user name for the MQTT broker is "tibco" and password is "mqttTIBCO18" (all without the quotes)

To test if your MQTT client is working fine, test by following the instructions provided in Appendix B

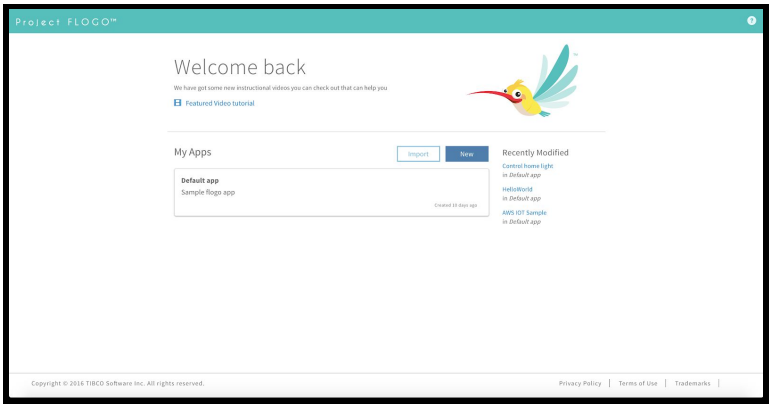
Installing Flogo

For the purposes of this hackathon, this guide will be sufficient for you. In the future, for learning, understanding and eventually contributing to Flogo you should start here.

Installing Flogo is very simple. Once you have docker installed on your machine, all you need to do is run the following command.

```
docker run -it -p 3303:3303 flogo/flogo-docker:v0.5.5-hf1
eula-accept
```

As you see you are not installing the latest but a tagged version “v0.5.5-hf1”. In the future when you try on your own, we recommend you try the latest. Just replace 0.5.5-hf1 with “latest” without the quotes.



Details about what you just executed is explained here in the Flogo quickstart

Launching the Web UI

To launch Flogo WebUI simply open your favorite web browser, and navigate to http://localhost:3303. You’ll see our mascot Flynn there to greet you!

Custom activities in Flogo

To create the microservices required for Geoff's Logistics, a few custom activities will have to be imported into your Flogo installation. Custom activities add reusable functionality to Flogo. Developers contribute a lot of custom activities to Flogo and some of them are part of the showcase .

There are three activities that need to be installed to complete the challenges.

1) Convert a JSON string to a JSON object - github.com/jvanderl/flogo-components/activity/jsontodata

2) Aggregate on a data item - github.com/ayh20/flogo-components/activity/aggregate

3) MQTT Publisher (This version supports the ANY data type, so will publish a string or a JSON object) - github.com/jvanderl/flogo-components/incubator/activity/mqtt

Installing the custom activities

The three activity URL's to install are

1) github.com/jvanderl/flogo-components/activity/jsontodata
2) github.com/ayh20/flogo-components/activity/aggregate
3) github.com/jvanderl/flogo-components/incubator/activity/mqtt

This video will walk you through the process of installing the 3 activities. The rest of the guide provides you instructions with screenshots and is more guided. Installing the custom activities is a one time task after your Flogo installation.

First Challenge: Flogo Microservice for Aggregation

The first challenge will involve building a microservice in Flogo that will aggregate the average speed of a truck over a 5 second interval. As each message is received from the Onboard unit of the truck, every five seconds the flogo microservice will compute the average speed.

There is a lot you will be learning as you build the first challenge considering we will be introducing you to many aspects of Flogo. The subsequent challenges should feel very simple once you complete the first challenge.

As you execute this challenge you will do the following:

1) Receive msg on MQTT at 1 event each second produce from the Simulator
2) Compute the avg speed
3) Send to MQTT at 1 event each 5 second to a topic

Simulation data format in json published on a common MQTT topic (mum/data):

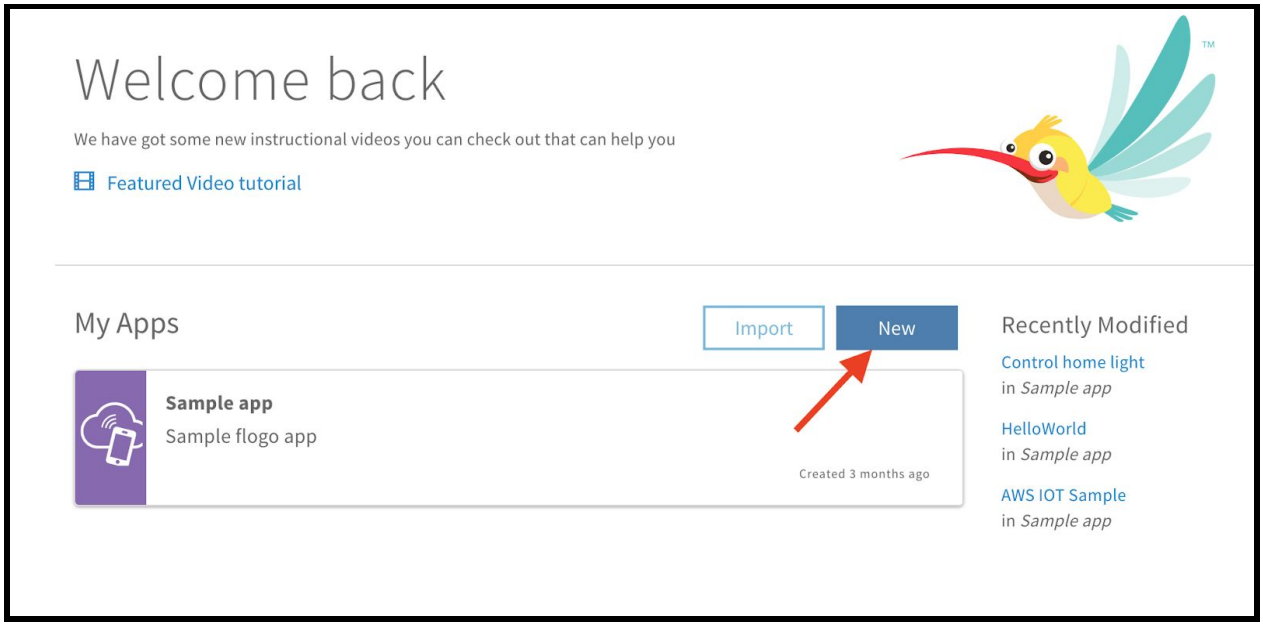
```
{
  "mumreport": {
    "truckId": "1",
    "dateTime": "2018-07-1221:39:49.356+0200",
    "fuelLevel": "50.5",
    "position":
{
        "lat": "47.34",
        "long": "23.34",
        "speed": "34.0"
      }
  }
}
```

Note: fuel is 0 - 100% (in percentage)

Flogo uses a concept of an App. An **App contains one or more flows**. When you build and deploy Flogo the **deployment unit is the Application**.

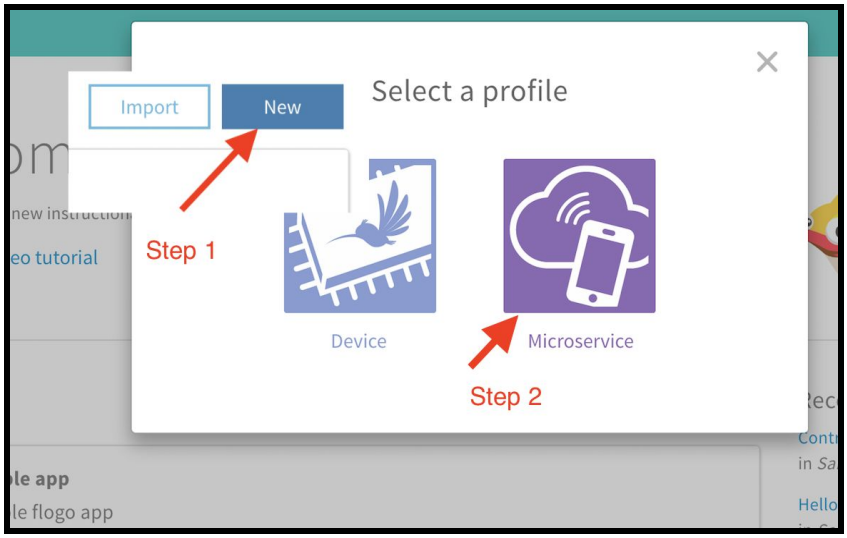
Therefore, to complete the challenge we need to start by **creating a new App and our first flow**:

1) Open the browser and navigate to http://localhost:3303 from your local browser. On the

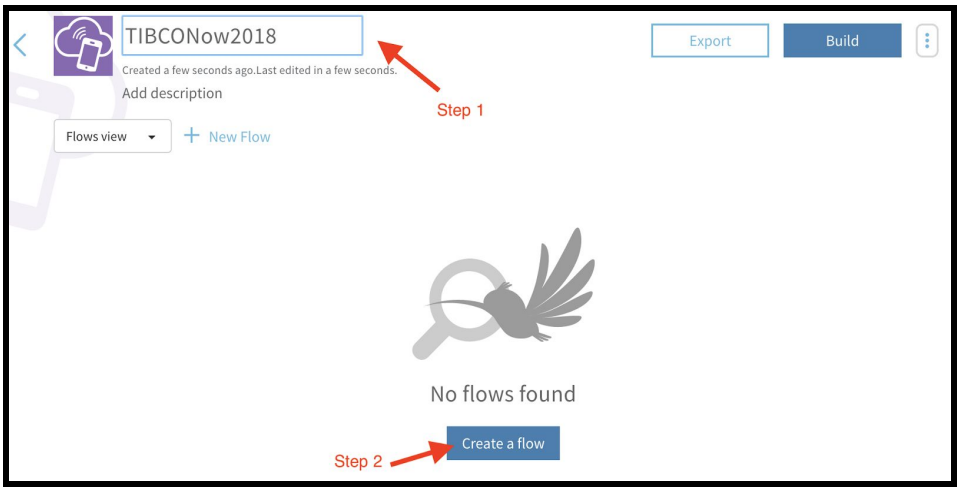


Flogo apps page, click new. The Flogo apps page is also the home page.

2) **Select the microservices profile.** This will allow us to create a Flogo executable that can be deployed to Operating systems like Linux, Windows, MacOS and more.



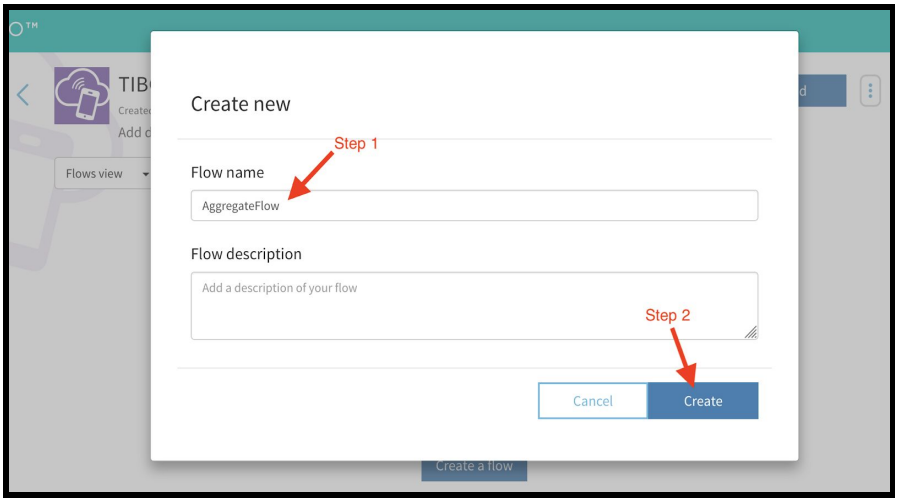
3) Click on Untitled App to **set the name for this Application**. For e.g. type name **TIBCONow2018**



This name is also used as a name of the executable that will be built. After setting the app name, **click Create a Flow**

4) Name your first flow “AggregateFlow” (without the quotes) and click “Create”

If you are wondering why “AggregateFlow” then wonder no longer! The purpose of this use case is simple enough.



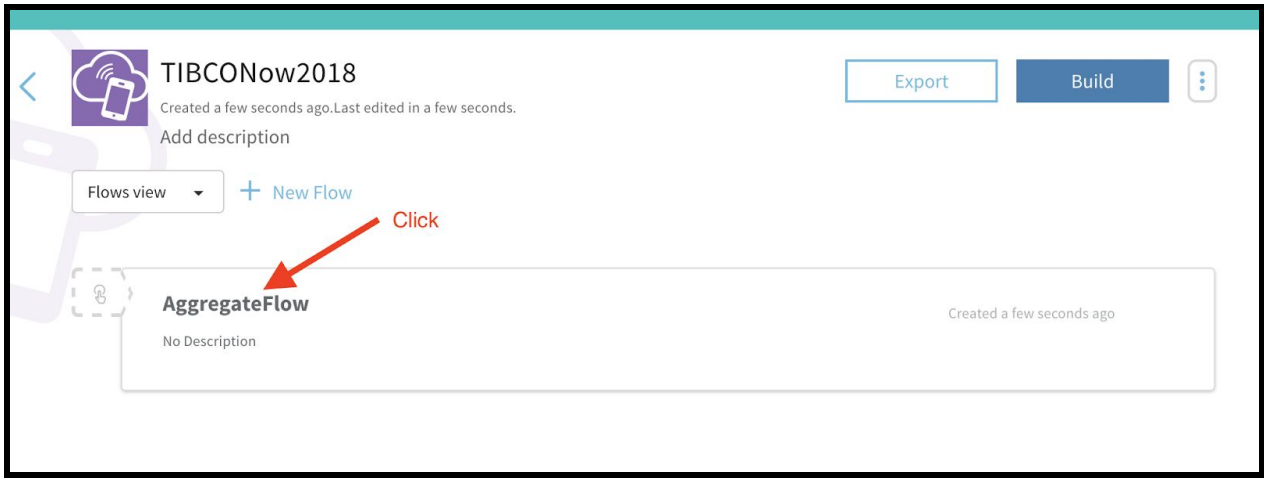
You are getting a stream of interleaved events from the Trucks MUM unit as 1 message per Truck per Second.

What you want to do is reduce some of the noise and reduce the message rate to one message every three seconds … **Let's face it .. this is status information from a Truck, it really doesn’t travel very far in 3 seconds !**

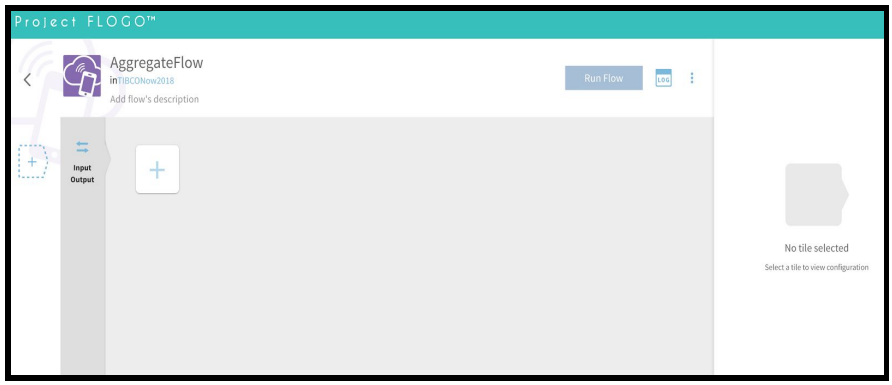
We are going to use a Flogo Activity that will aggregate data based on an input value, **in this case speed**. This way we will get an aggregate speed over five messages from each of the trucks.

We have a flow!. Now we will step through the process of configuring the flow and introduce you to the Flogo UI and adding activities to flows.

5) Edit the flow by clicking on the name “AggregateFlow”

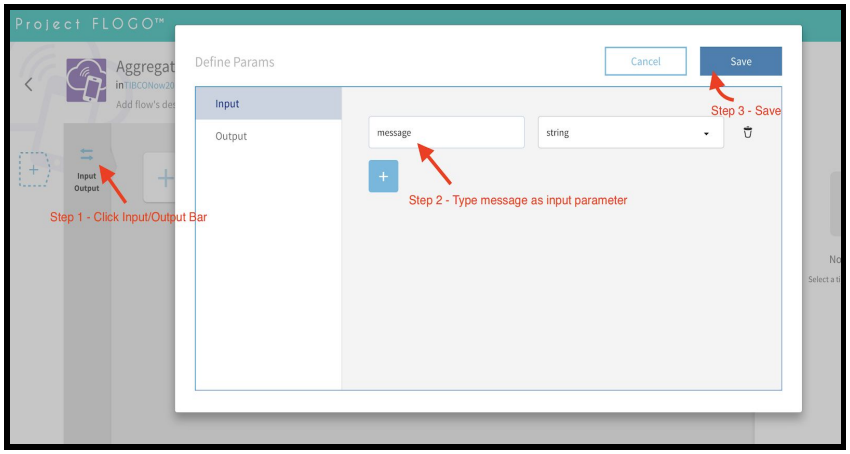


Most Flows will have data that is passed into or returned by the flow. For example having a HTTP trigger that calls the flow means that you will probably want to pass in the HTTP payload.
To do this the flow needs to have it’s inputs and output defined.



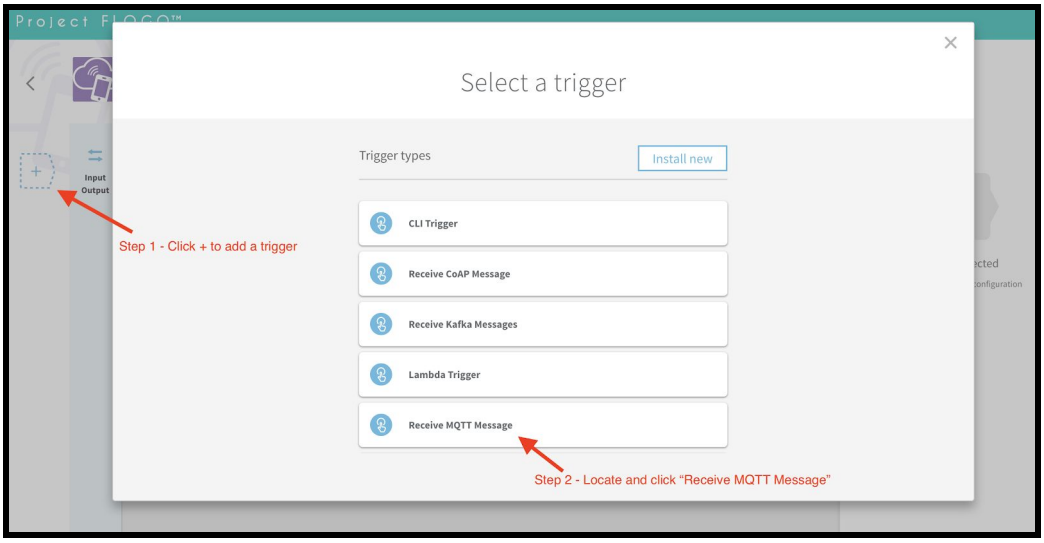
That’s exactly what we intend to do next !

6) Start by creating the flow input/output settings.
Click on the input output bar enter the parameter name of “message” (only entering the input param is needed). This will be used to pass data from the trigger to the flow.. **Click Save**

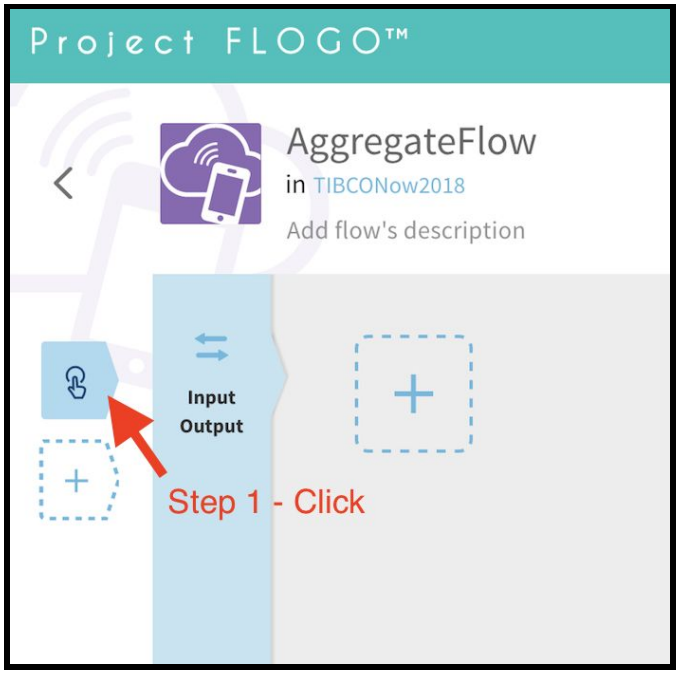


Next, we will add the trigger event that will start the Flow. In our case the data is coming from an MQTT broker in the cloud.

7) Click on the "**Add Trigger**" icon (dashed box to the left of the input/output bar with a Plus in the middle) Locate and click on "**Receive MQTT Message**"

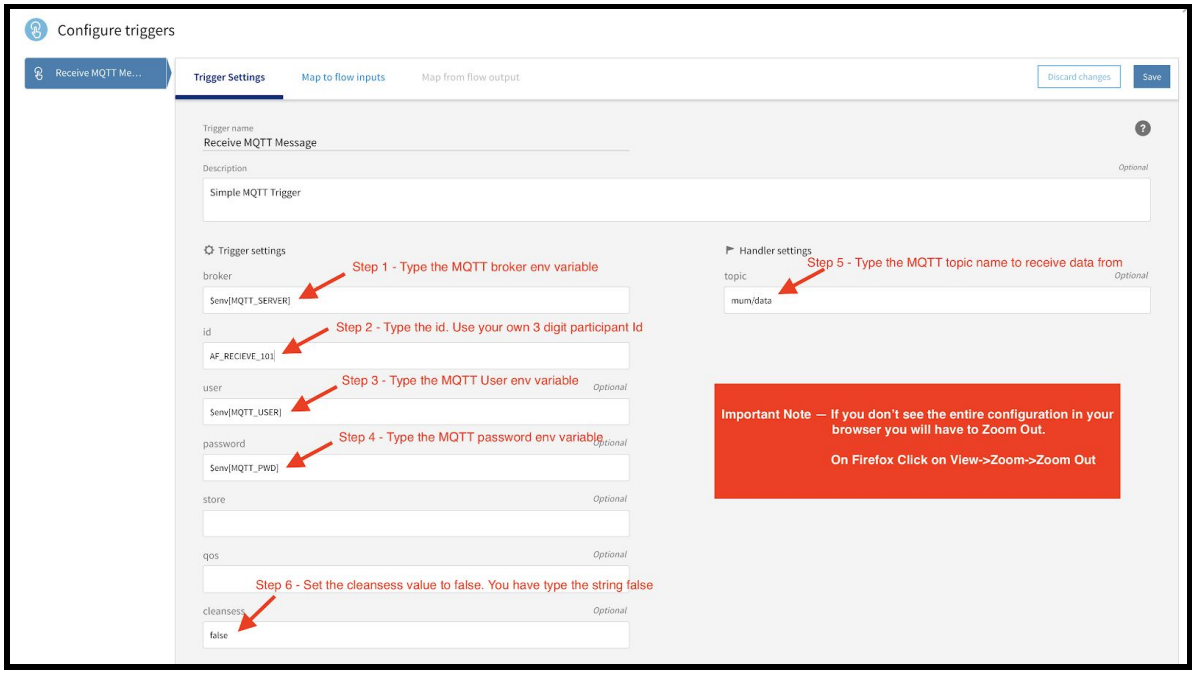


8) Click on the trigger just added … As soon you click it you will see a settings panel on the right. This is the configuration panel for the trigger. In this specific case, this is the configuration panel for the Receive MQTT trigger.

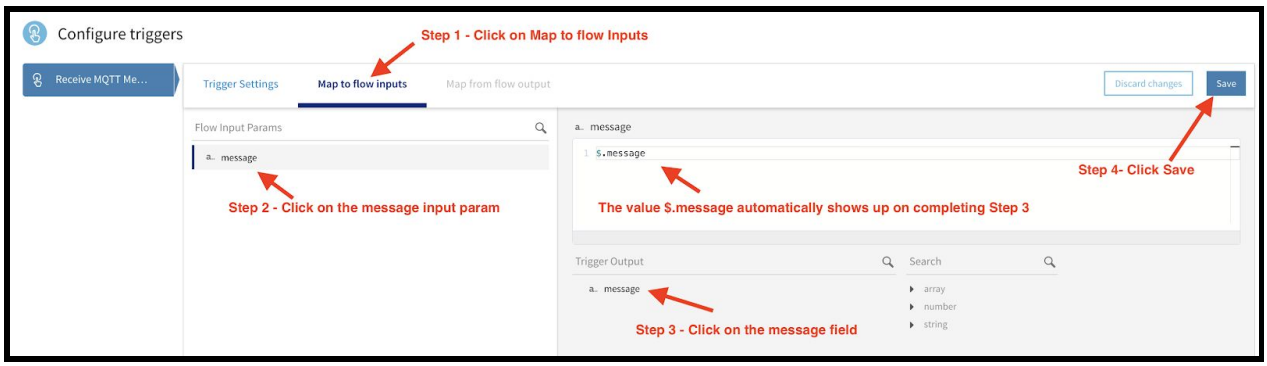


Note the values to be mapped. It's pretty straightforward. See next screenshot.

broker = $env[MQTT_SERVER]
id = AF_RECEIVE_*nnn*
nnn is your participant ID that uniquely identifies you as a MQTT client. Come up with a random 3 digit number and use it. For e.g 101. If you are not sure ask.
The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. AF_RECEIVE_*nnn* represents the MQTT message receive on Aggregate Flow , hence AF in the beginning.
user = $env[MQTT_USER]
password = $env[MQTT_PWD]
topic = mum/data
cleansess = false
NOTE: The participantId you decided to use is for the entire guide. Use it everywhere**.**



9) Now we need to configure the data mapping. Next to the "**Trigger Settings**", you should see a tab for "**Map to flow inputs**". Click on it.



Clicking on **Map to flow Inputs** will open the Mapper. Values for activities can be specified here or on the tiles settings panel (which is not introduced to you yet) - so don't worry about it at this time. If they are specified in both places then the values specified here in the Mapper takes precedence.

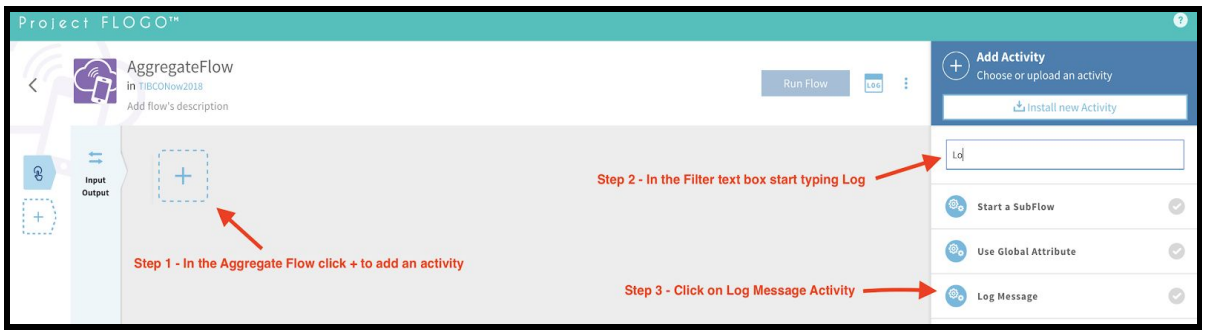
Select the message field in the Flow input Params column (Step 2 from screenshot). Then select the message field under Trigger output (Step 3 in the screenshot). The result should be mapping of input field message is $.message

Click on Save and then the X button.

At this point we have a empty flow that will be kicked off when a MQTT message arrives. **Now we need to do something with the message !**

Let's do something very simple. Let's just map the inbound message to the log activity so we can see what we are receiving from MQTT. Should be simple

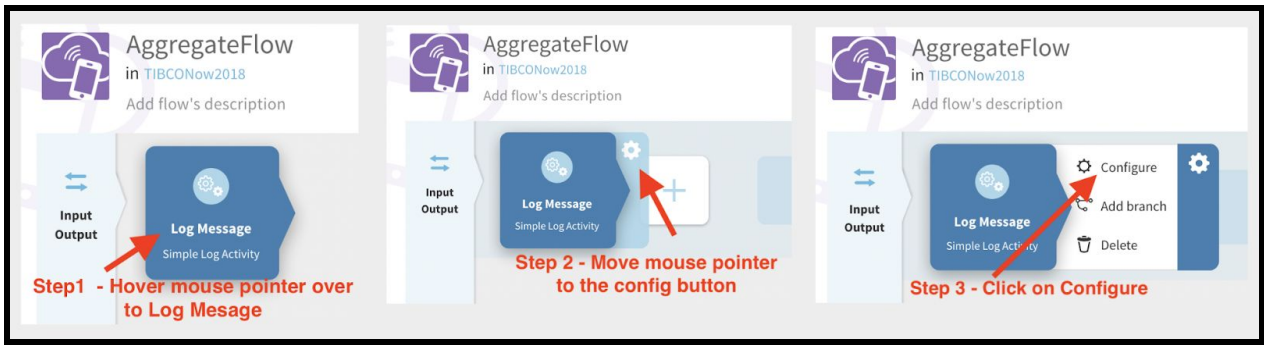
10) Click into the flow on the **Plus icon**. This will display a list of activities on the right hand side. Your version of Flogo may show a pop up.



On the right hand side you can scroll to find the **Log Message Activity** OR start typing in the filter to limit the list. Click on Log Message and it appears on the flow canvas.

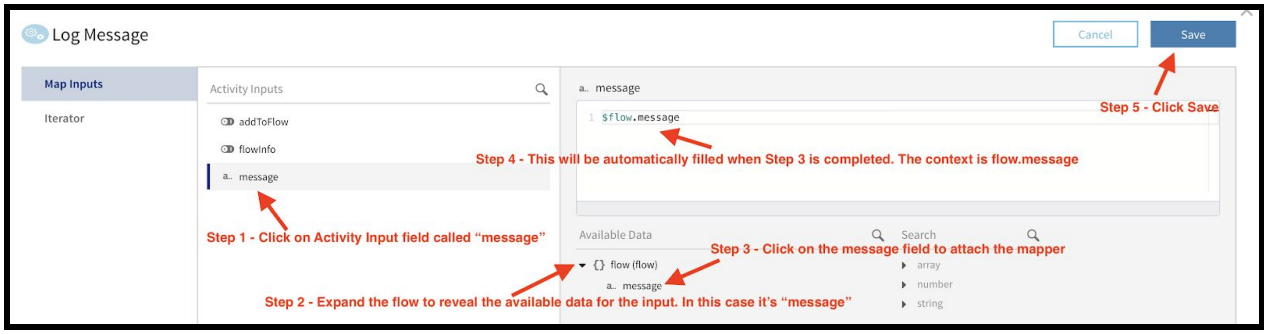
At this point you will see tile properties on the right hand side. At the top (see screenshot) you will see the name of the tile which you can change.
A description of the activity and the ID of the activity. **This ID is important**, as you will use this value when mapping to or from activities. From the screenshot you will notice that the id for this Log Activity is **log_2.** Different activities will have different id's and you will notice the importance it when you do some mapping.

11) Hover the mouse pointer over the activity tile and a config menu will appear.



Move the mouse over to the settings menu and it will expand ..Click on **Configure** to open the Mapper.

12) In the activity Inputs, select the message field (Step 1 in the screenshot)



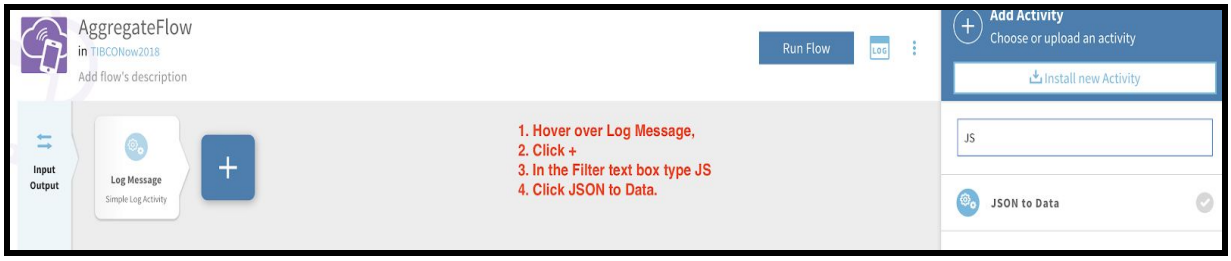
Under Available data, expand the flow (step 2 in the screenshot), to reveal message. Click on **message** (step 3 in the screenshot) and it will fill in the upper box ….. $flow.message (step 4)

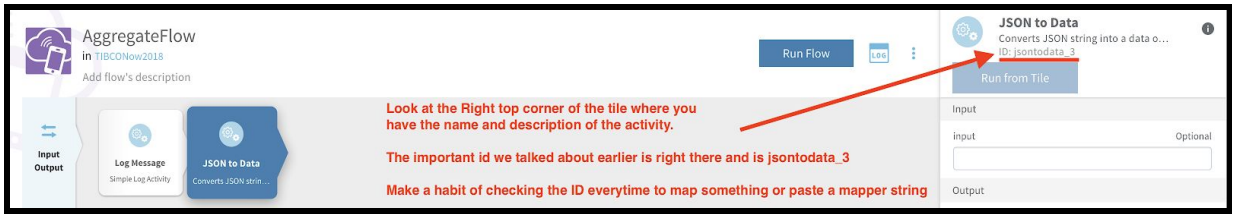
NOTE - When using the fields under available data **be careful to only click when required**. A click will insert the current object name into the mapping window.

Click **Save** (Step 5 in the above screenshot)

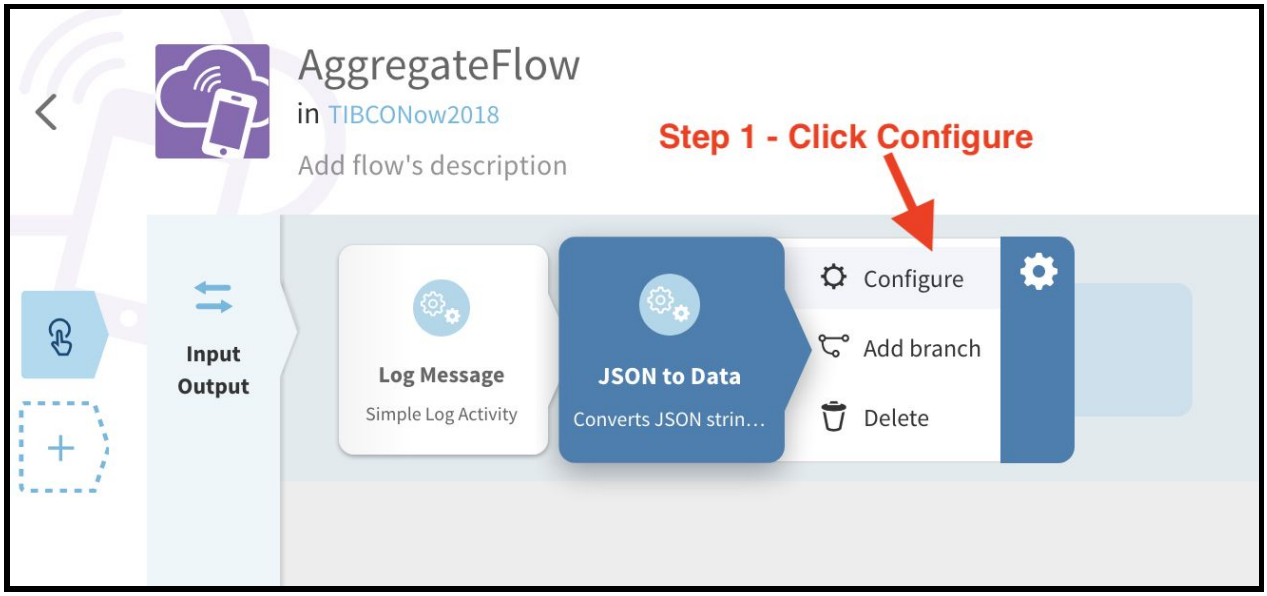
Congratulations… At this stage you should have a working flow. Let's continue to add more activities to this flow to make it functional.

13) The inbound message is coming in as a JSON string. Let's convert the JSON string to a JSON object with the JSON to Data activity. Hover over the Log Message and you will see a + button to add another activity.

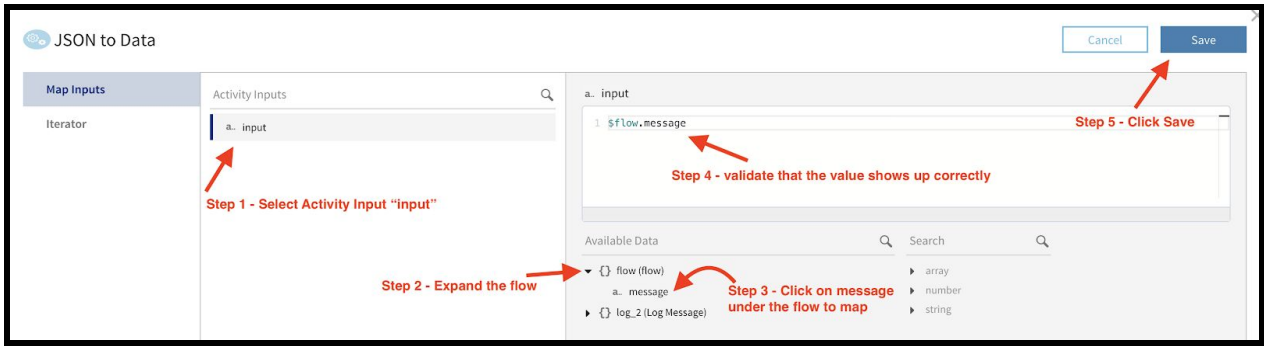




14) Same deal as before to configure. Hover over JSON to Data activity , move the mouse to settings and click on configure



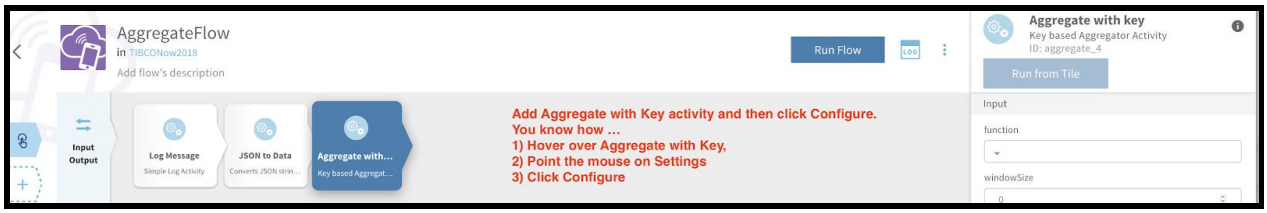
15) Map it's input to $flow.message . This is to make sure that the inbound message which is a JSON string is mapped to the input of JSON to Data activity.



By now, you should have gotten a hang of configuring an activity, mapping inputs, etc. Let's continue to build on that skills that you have just learnt.

We now need to add the magic that will aggregate the data for each truck. If you remember from the story we will aggregate data every 3 seconds.

16) Add a new activity (you should know the steps to add a new activity by now). The activity is called “Aggregate with key”. You can type Agg in the filter if you don’t find it right away.
NOTE - check the Id of the activity. It should show aggregate_4 or something else if you have deleted or added additional activities.



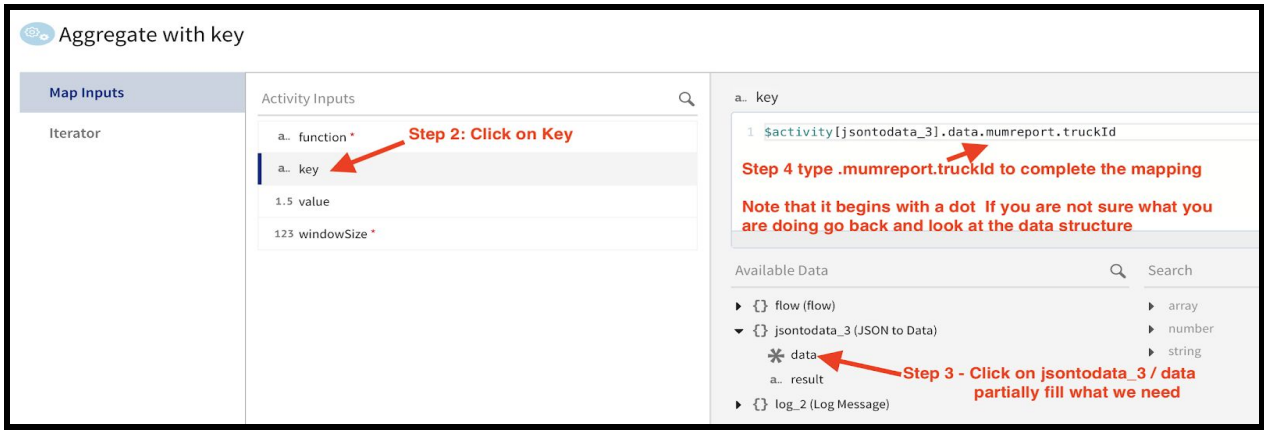
17) It’s time to configure “Aggregate With Key” Activity.
Aggregate with key activity requires 4 parameters to be input so it can return an aggregated value.

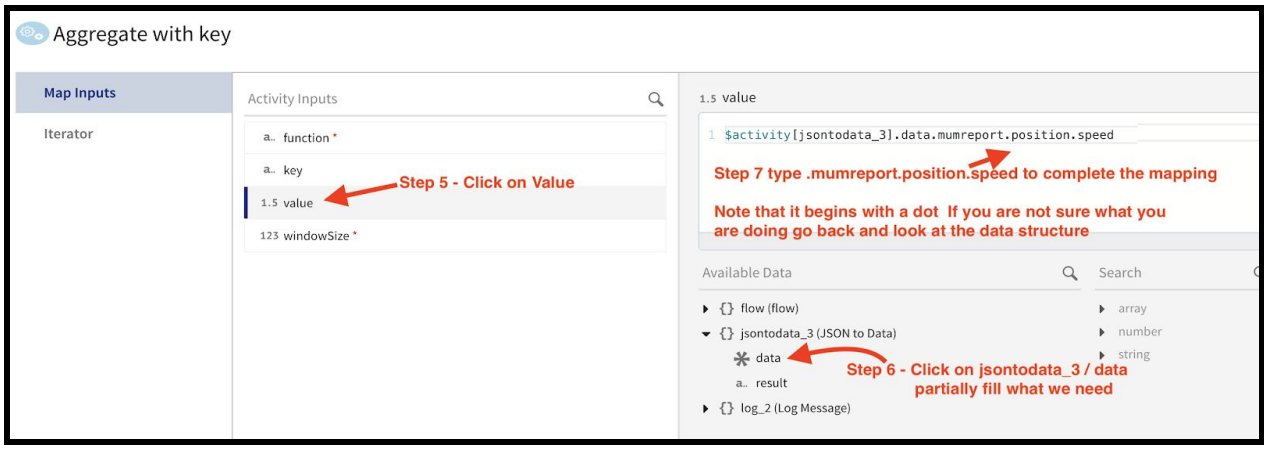
- Function - There are several aggregation methods supported but we will use block average method.
- Key - The key that we will use for aggregation will be the truck Id
- Value - We will be aggregating the speed of the truck so the value in this case will be speed.
- windowSize - To test quickly we will use a windowSize of 3 seconds.

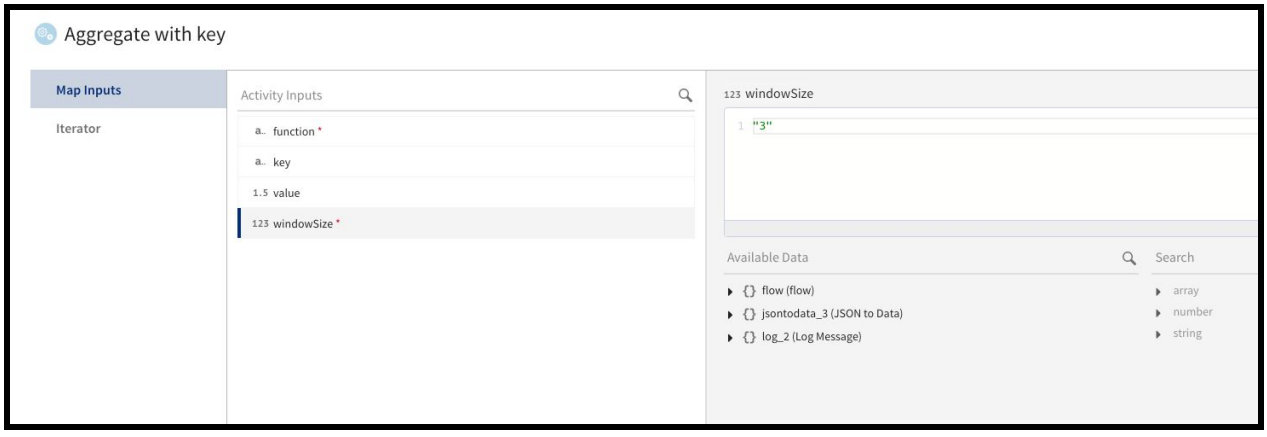
Let’s do it one step at the time if you have the config screen open. Take a look at the screenshots as well. Basically what you see below if what we will be mapping**.**

Function = `"block_avg"`
Key = `$activity[jsontodata_3].data.mumreport.truckId`
Value = `$activity[jsontodata_3].data.mumreport.position.speed`
windowSize = `"3"`

Note that the activity Id might not be the same as the example so **don’t just copy paste!** This document is using jsontodata_3 but it can be different in your case. Just look at the **Available Data items for the correct naming** (you can add the data element this way and then add the suffixes manually). You can follow the screenshots as well.





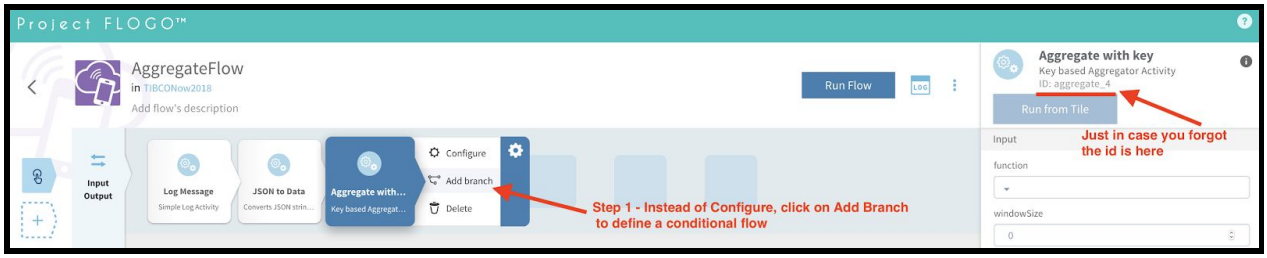


18) **Click Save. Close by clicking the X button if you are not returned to the activity view.** If you saved and closed before making all the mappings open the configuration panel and complete all the mappings.

We now know how to identify the id of an activity. We need to know the id of the Aggregate With Key activity. If you are following this document step by step then the activity id should be **aggregate_4.** You should confirm that by clicking on the aggregate with key activity and looking at the top right corner.

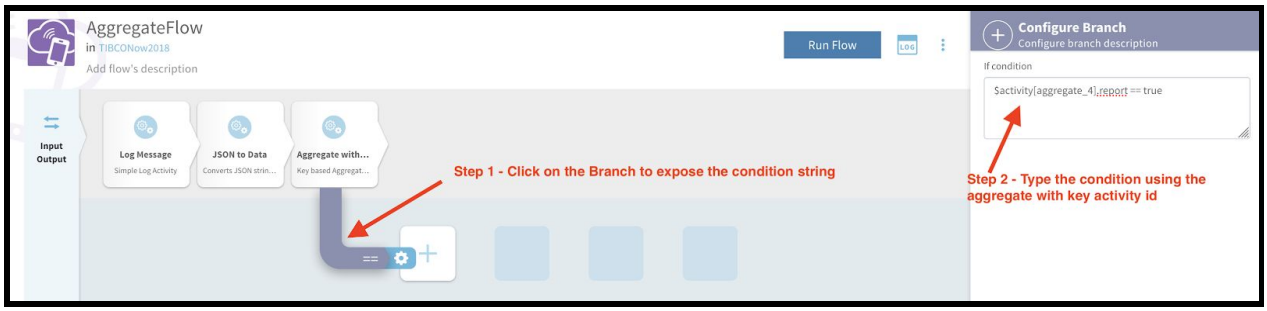
In the next step, we will introduce you to the notion of a branch. The data that was sent to the aggregator may not always return data. That is because it expects a certain number of messages to come from a truck before it can perform a 3 second aggregation. When the aggregator has an aggregated value it sets the report value to true in addition to sending the aggregated data. If the report value is not set to true then the data is captured and we can ignore this message from the truck.

19) Instead of clicking on the Configure button to enter the configuration page, this time click on “Add branch” to define a conditional flow.

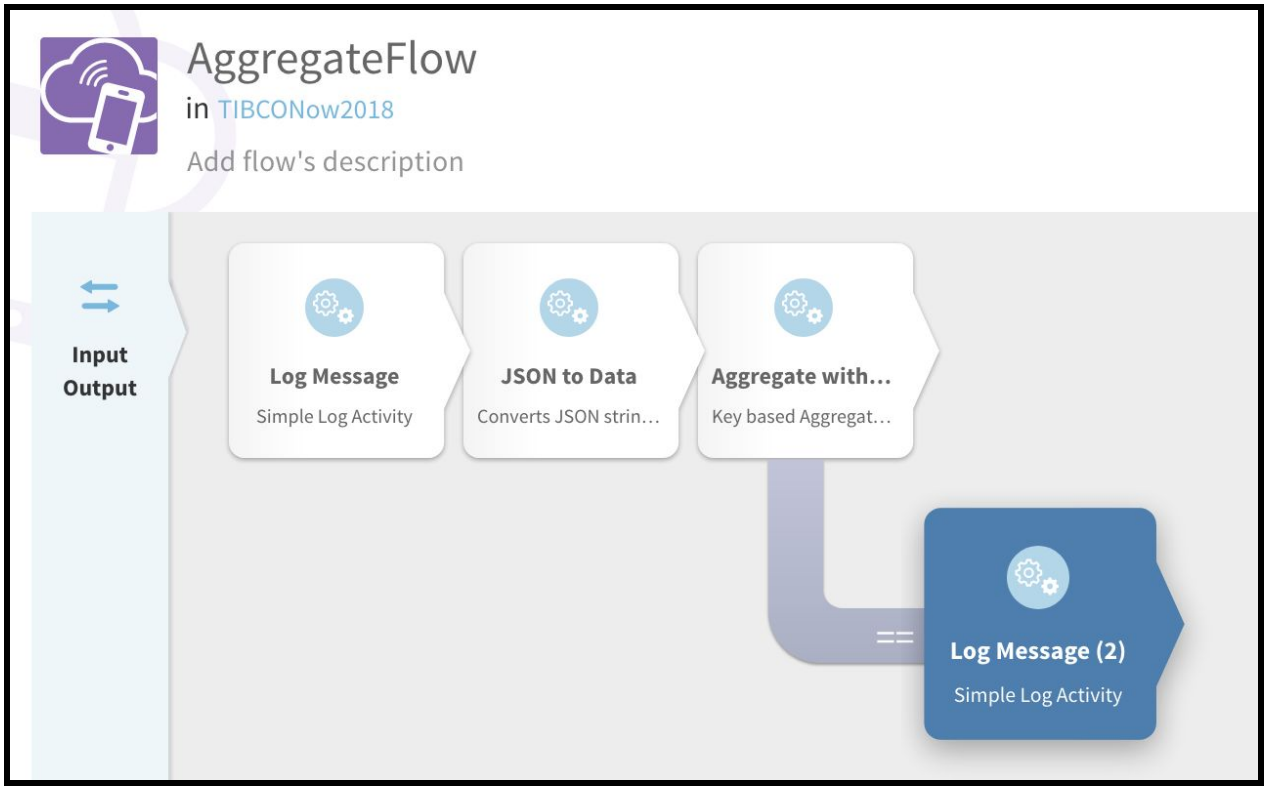


Click on the branch and add the condition. In this case the activity id is aggregate_4 so we will use that in our condition:

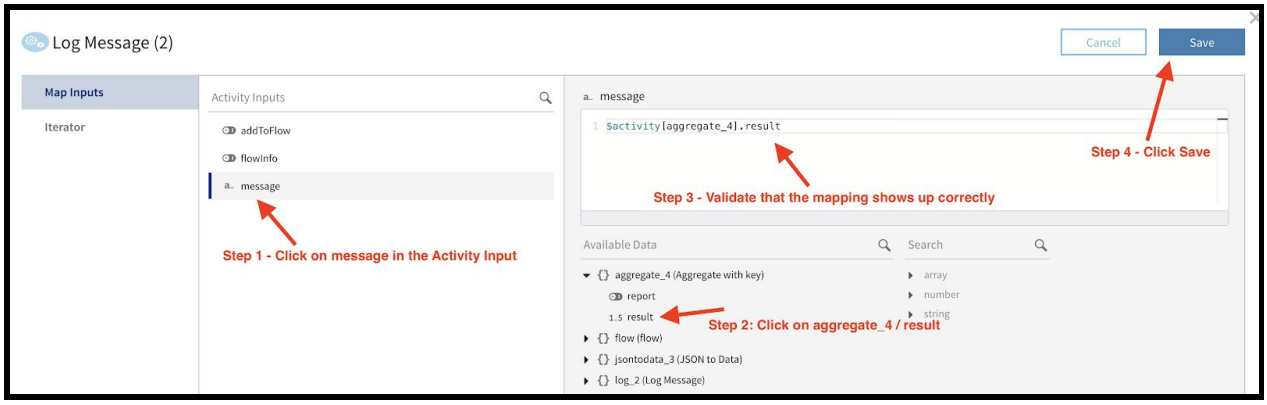
```
$activity[aggregate_4].report == true
```



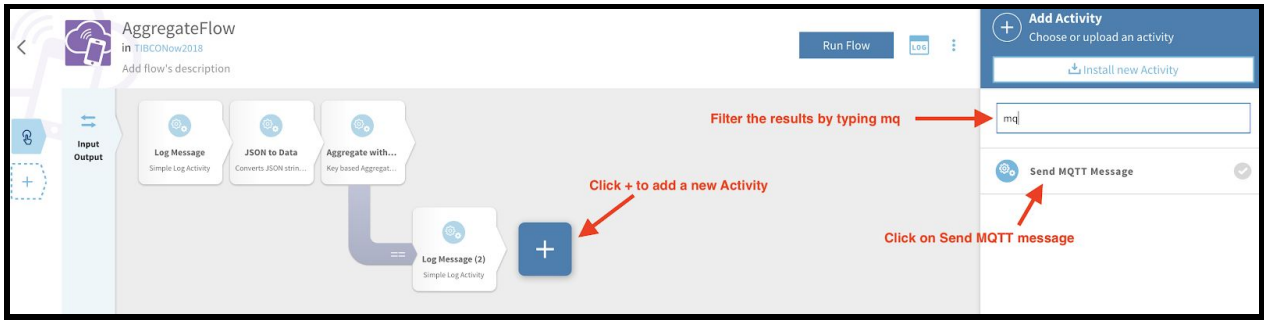
19) At this point let's log the output from the aggregator. First, add a Log Message activity in the branch. You have added enough activities to know how to add an activity. Once you add the log activity it should like below. Note the name of the Log Message activity . It says Log Message (2) because it's our second log message activity in this flow.



20) Configure the Log Message (2) activity and map the message field from result data.

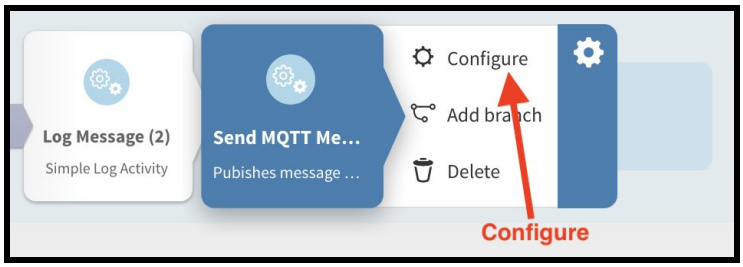


21) Now, that we have the aggregate function defined in the flow it's time to send it out on MQTT so we can consume the aggregated data in other flows. We will now add a new activity to send a MQTT message. So, we have used receive MQTT, Log Message and an aggregate function. Time to explore a new activity.



Click on the + button and add a Send MQTT Message activity.

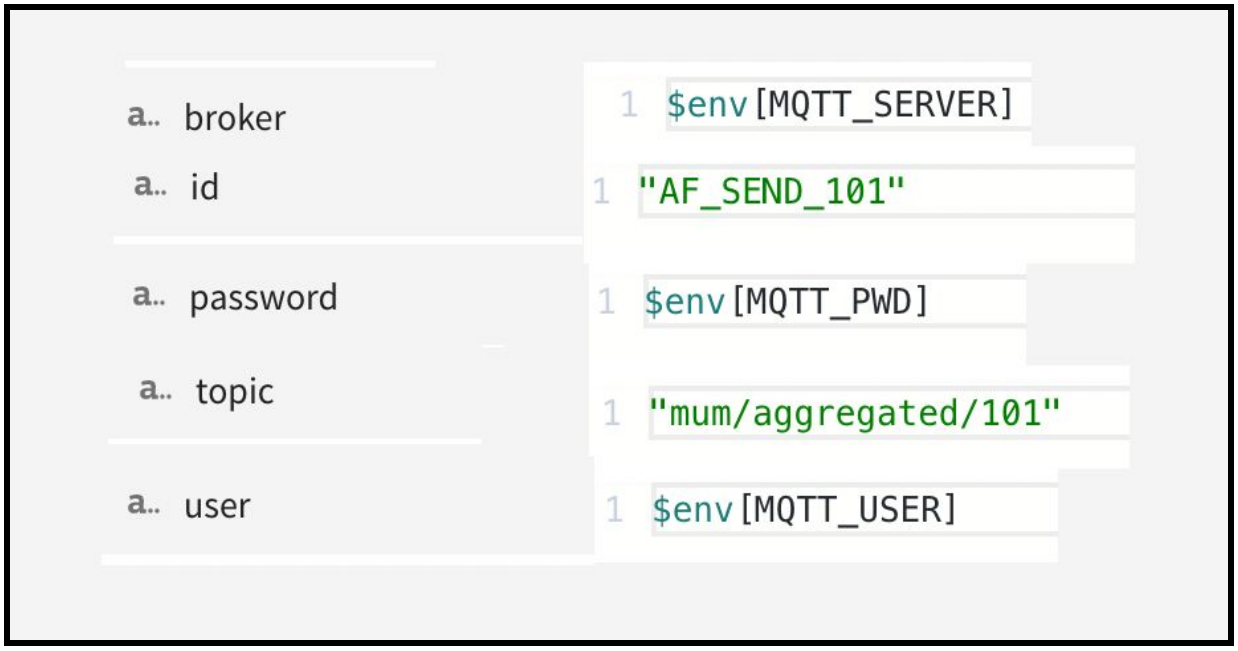
Once added note the activity id as always and click on the configure button to start the configuration.



22) In the configuration panel, we will use the same MQTT settings as before, except that we want to send the data to a different topic and with an unique client ID. This time we will send the messages on AF_SEND_<participantId>.

broker = $env[MQTT_SERVER]
id = "AF_SEND_*nnn*" where nnn is your participant ID. You should know your participant Id by now. The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. AF_SEND_*nnn* represents the MQTT message send from Aggregate Flow , hence AF_SEND in the beginning.
user = $env[MQTT_USER]
password = $env[MQTT_PWD]
topic = "mum/aggregated/*nnn*" - Where nnn is your ParticipantId. This topic name is important. Remember the actual value typed. We need it for the next challenge where this message will become the input
message = << You will be constructing the message payload in the text editor based on sample payload>>

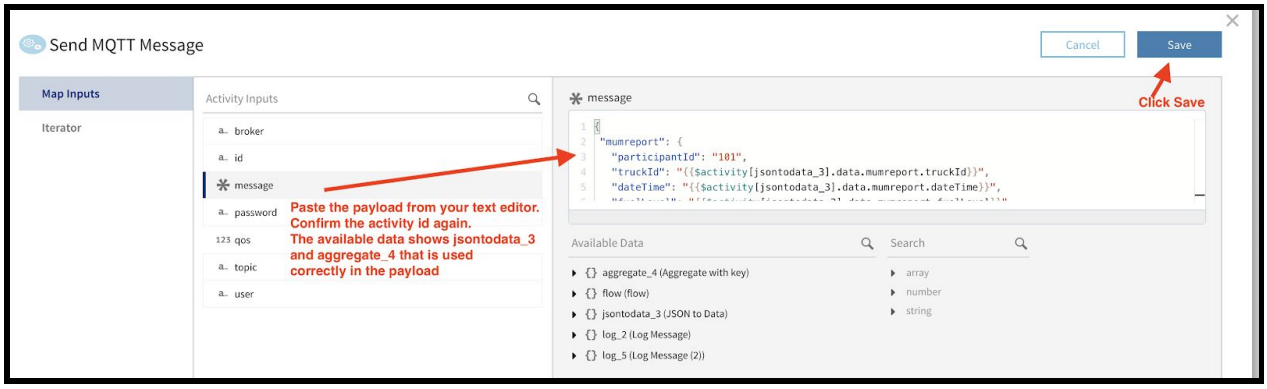
The above mapping should be straight forward. See screenshot for values mapped. Notice that the literal strings wrapped around quotes and participant id is replaced with an actual value. You should use your own participant id provided to you.



Let's construct the payload to send in the message field.
Here's the sample payload message. This sample payload is available in the *app-challenge-examples* folder on the instance. Open **payload-send-uc1** in the text editor to modify. There is one thing that you certainly need to change and that is the participant id. The references to jsontodata_3 and aggregate_4 is something that you should confirm by looking at the configuration window of the Send MQTT message. The Available Data panel will tell you the exact id's.

```
{
 "mumreport": {
   "participantId": "nnn",
   "truckId": "{{$activity[jsontodata_3].data.mumreport.truckId}}",
   "dateTime": "{{$activity[jsontodata_3].data.mumreport.dateTime}}",
   "fuelLevel": "{{$activity[jsontodata_3].data.mumreport.fuelLevel}}",
   "position": {
     "lat": "{{$activity[jsontodata_3].data.mumreport.position.lat}}",
     "long": "{{$activity[jsontodata_3].data.mumreport.position.long}}",
     "speed": "{{$activity[aggregate_4].result}}"
   }
 }
}
```

In the mapper we need to build the output message as a JSON object. Copy the payload from your text editor to use as the mapping on the field message.



Now, let's test this.
Use MQTT.fx as to in Appendix B .

Second Challenge: Flogo Microservice to detect low fuel

The second challenge will involve building a microservice in Flogo that will detect low fuel and on detection uses a REST service to find the closest gas/petrol station. Additionally, it will also report this information on MQTT so it can be consumed anyone tracking all the truck data.

The second challenge will use the MQTT message that you are sending after running the aggregate function. The Send MQTT message from first challenge will be used as an input in the second microservice we will develop. A lot of things in Flogo UI should be familiar to you by now. We will be creating a new flow and add new activities similar to what we did in the first challenge.

As you execute this challenge you will do the following:

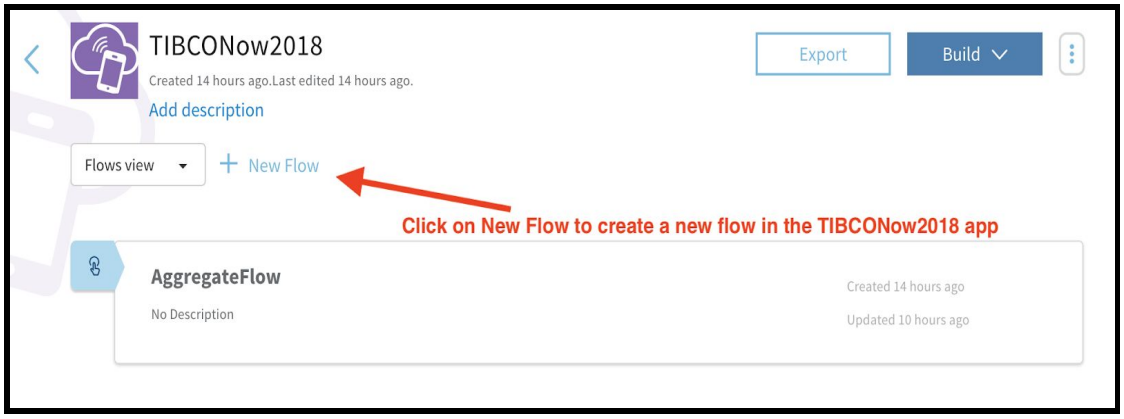
1) Receive aggregated msg on MQTT from the previous challenge (1)
2) If fuel level < 25% then use REST service to find closest gas/petrol station
3) Send to MQTT nearest station in the following format

```
{
  "mumreport": {
    "participantId": "100",
    "truckId": "1",
    "dateTime": "2018-07-1221:39:49.356+0200",
    "fuelLevel": "20.0",
    "position": {
      "lat": "47.34",
      "long": "23.34",
      "speed": "35.0"
    },
    "fuellocation": {
      "name": "Petrom",
      "lat": "47.25817",
      "long": "23.25653",
      "phone": "260-644331"
    }
  }
}
```

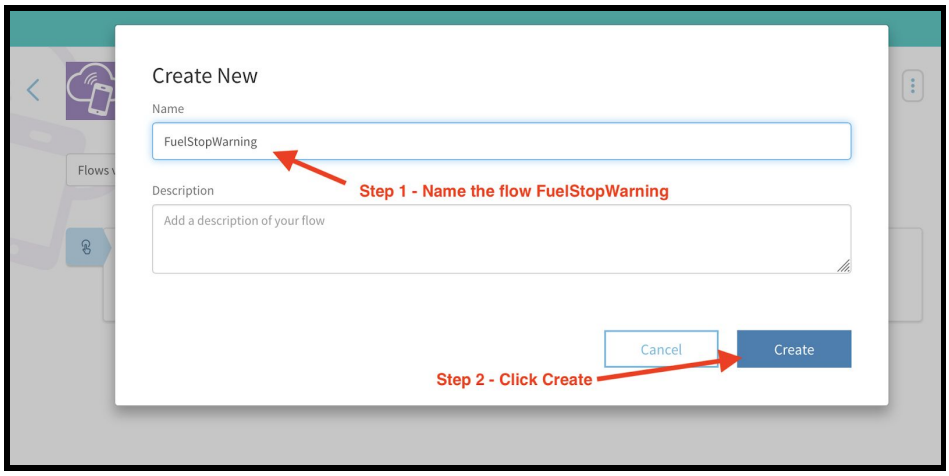
Now, let's create a flow that reads the aggregated data message and checks for a low fuel level. On low fuel, the flow needs to send a warning as MQTT message.

NOTE: We want you to use the what you have learned so far. Some of the instructions may just tell you to perform a task without specific instructions. It's intentional !

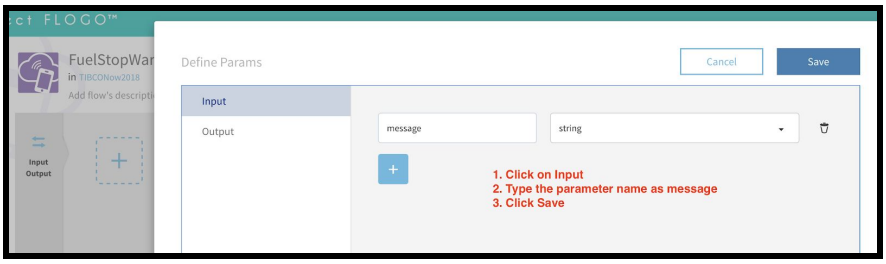
1) Create a new flow. The flow should be created in the app that you have already created. This will be the second flow in the app.



Name the flow **FuelStopWarning** and click Create



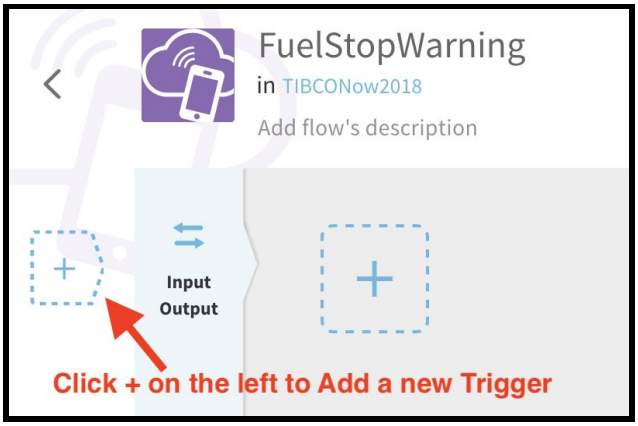
2) Click on the newly created Flow and configure the Input/Output parameters. You did this already during the first challenge.

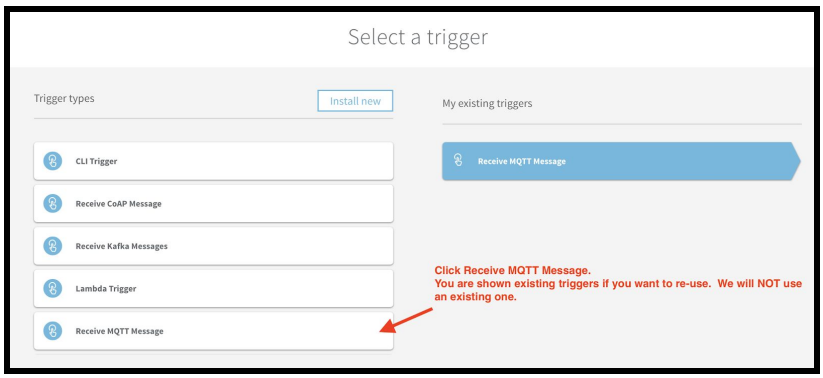


As you have probably noticed, this is pretty much a standard way of working in Flogo. Create a flow, define the inputs and next if you guessed it right - you said trigger !. We added a MQTT trigger during the first challenge. We will do the same again.

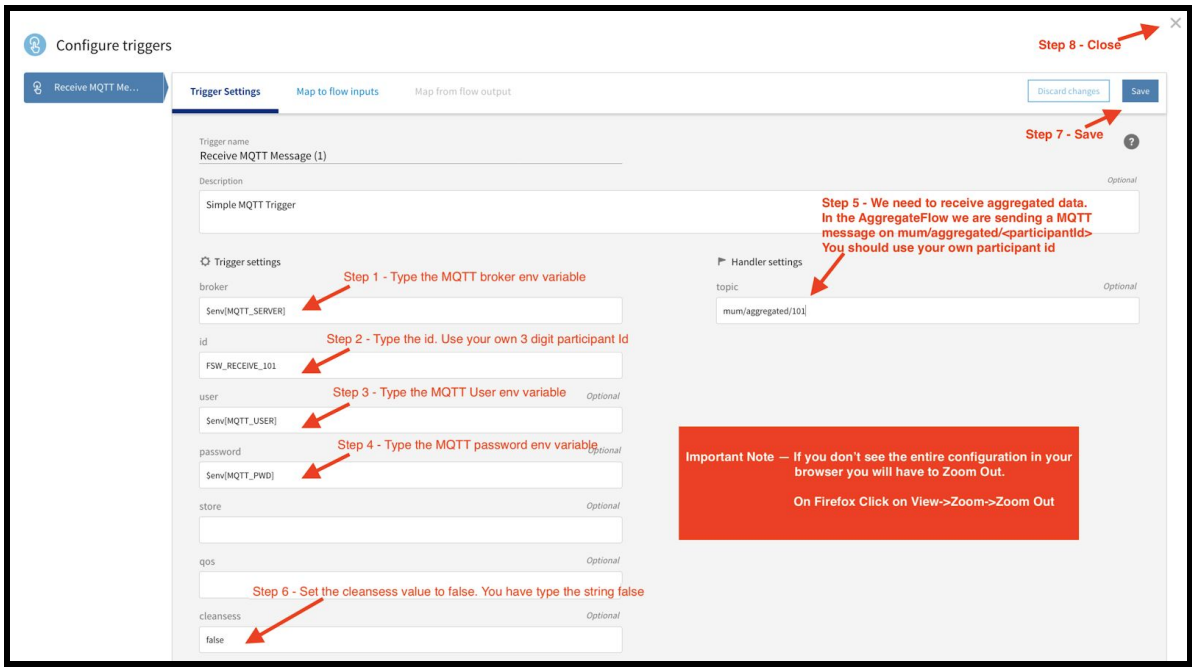
3) Adding a trigger to your flow is very simple. Click on the + button on the left most side to open the trigger view.

Here you will add a new Receive MQTT message. You will notice that the flow will provide you an option to reuse an existing trigger. Ignore it and add a new one as we have a couple of different settings.
Once added, click on Configure





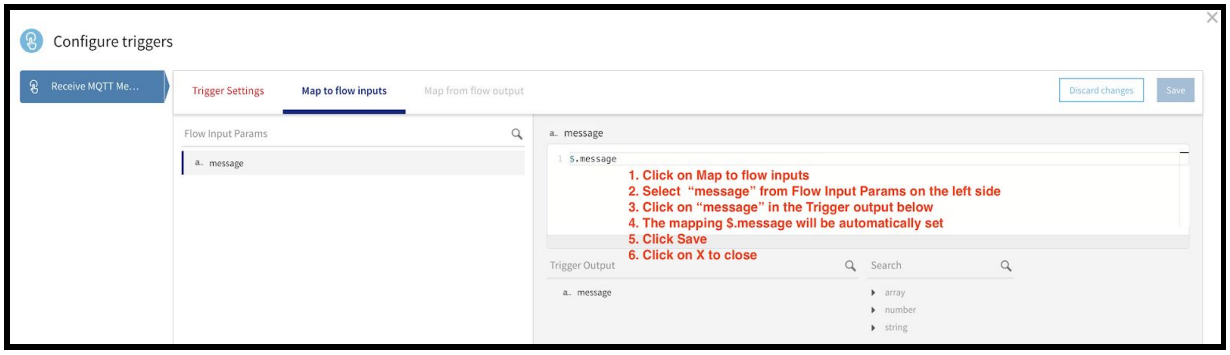
4) Let's configure the MQTT trigger now.



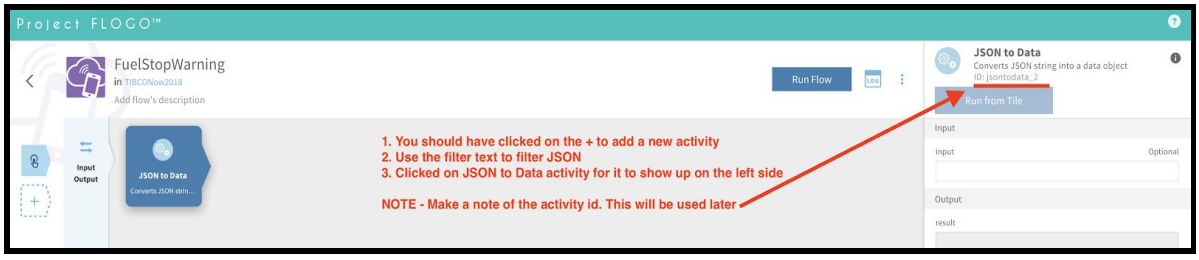
The configuration is similar to what we already did in the first challenge except for a couple of things. First, one is the client ID and second one is the subscription topic.

broker = $env[MQTT_SERVER]
id = FSW_RECEIVE_*nnn*
nnn is your participant ID. You should know your participant Id by now.
The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. FSW_RECEIVE_*nnn* represents the MQTT message receive on Fuel Stop Warning Flow , hence FSW_RECEIVE in the beginning.
user = $env[MQTT_USER]
password = $env[MQTT_PWD]
topic = mum/aggregated/<participantId>
This topic should be the exact same value that you used in the first challenge to send aggregated data.
cleansess = false
NOTE: You should be using your own participantId**.**

5) In the Configure after completing the Trigger Settings, click on the Map to flow inputs to setup the flow inputs.

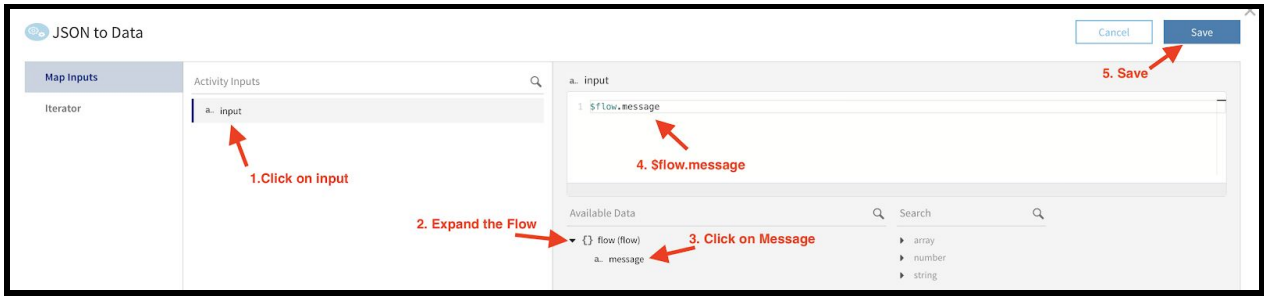


6) Add a JSON to Data Activity to FuelStopWarning flow. You have already added several activities during the previous challenge. You should be familiar. Once added , your screen show like what's in the screenshot. **NOTE** - Make a note of the activity id. You don't need to write it down. Just remember where to look for it when needed.



7) Configure the JSON to Data activity by clicking on the Configure button. You should hover on JSON to Data and then get to the Configure button.

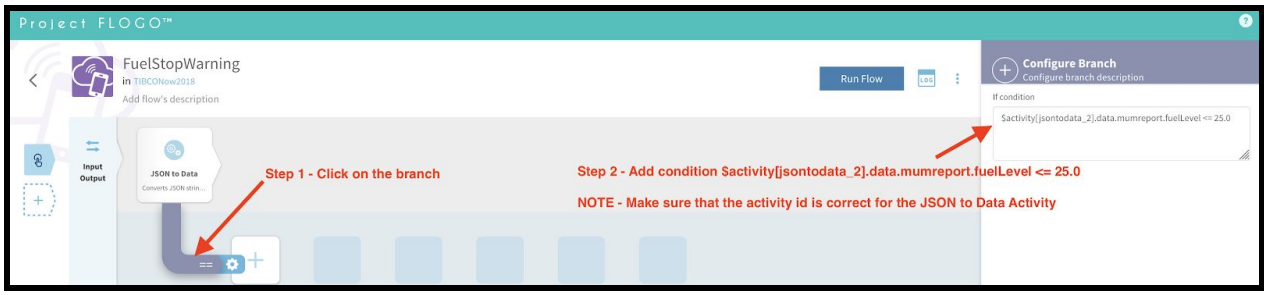
8) Let's map the activity input message. Again, we have done this before in the first challenge. Nothing new here



9) Now we are going to add some logic to check the fuel level. We need to check if the fuel level is below 25 (gallons / liters - call it whatever you want)
We will create a branch and add a condition to check the fuel level.

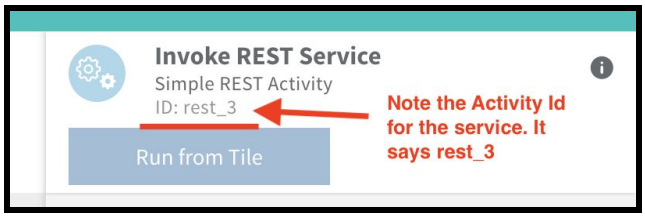
```
$activity[jsontodata_2].data.mumreport.fuelLevel <= 25.0
```

You already know how to identify the activity id for JSON to Data activity. Click on the JSON to Data tile to find the activity id. If it is different than jsontodata_2, use what you find in the condition.



10) The condition we added checks if the fuel level is < 25. What do we do if it is less than 25? Let's call a REST service that will look up fuel station. The fuel station lookup service is deployed on the **TIBCO cloud and written using TIBCO Cloud Integration Web Flows that are based on Flogo.** This particular service is front ending a **very complex Bing maps API to expose a simple interface** that works with just the Latitude and longitude.

a



11) As always, the new activity needs to be configured. You should now how to open the configuration page for an activity. **Hover the mouse over Invoke REST Service and click on Configure.** You can refer the screenshot as well but here's what goes into various inputs.
If the activity id for JSON To Data is not jsontodata_2, make sure to update it. DO NOT MODIFY the api_key. That's the access key to make the REST call.

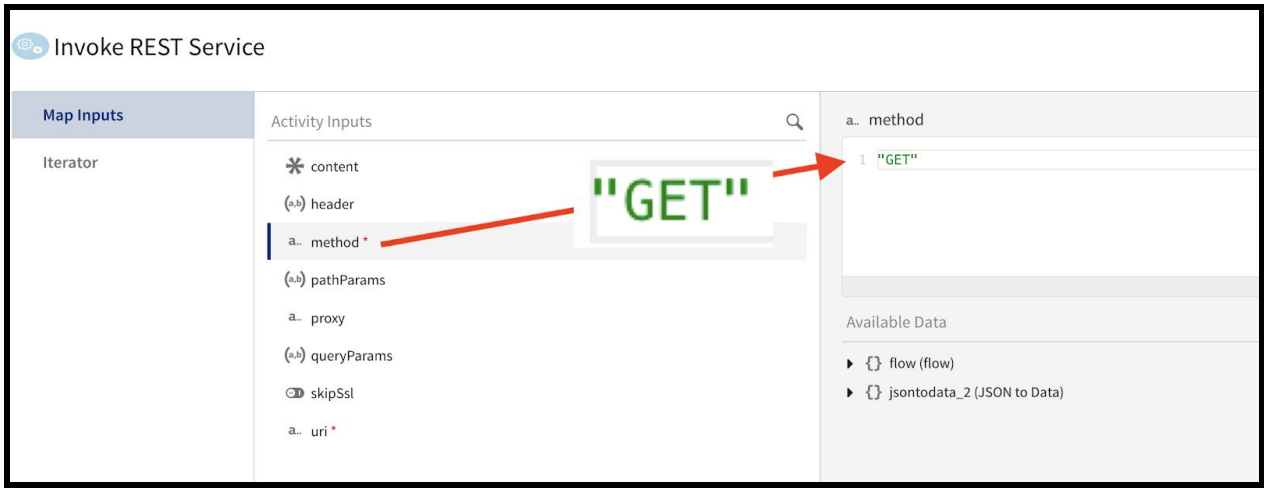
Method = "GET"

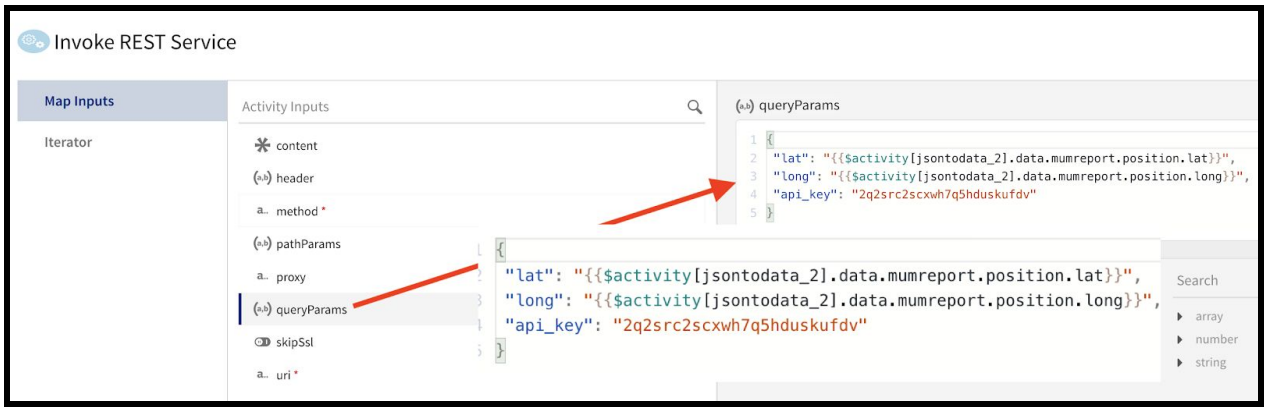
Queryparms =

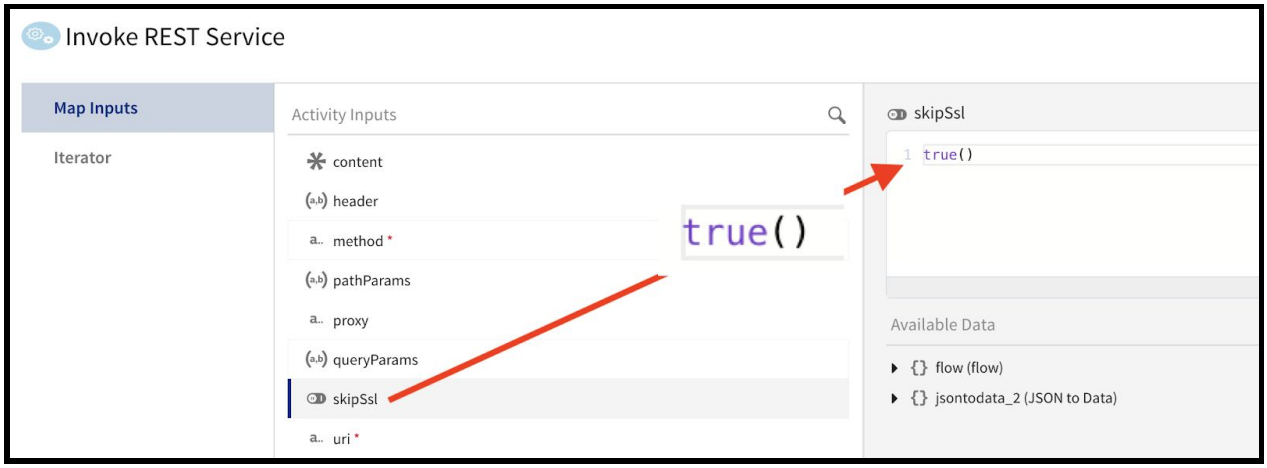
```
{
 "lat": "{{$activity[jsontodata_2].data.mumreport.position.lat}}",
 "long": "{{$activity[jsontodata_2].data.mumreport.position.long}}",
 "api_key": "2q2src2scxwh7q5hduskufdv"
}
```

skipSsl = true()

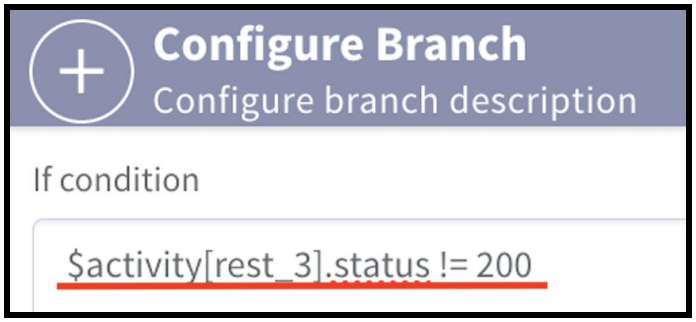
uri = "http://ahampshi.api.mashery.com/tn18/locatefuel"







12) The fuel service API run on BING is wrapped around in TIBCO Cloud Integration to simplify and managed through TIBCO Mashery as an API. Mashery also provides the caching capabilities that is useful when you are using the same lat, long to query. However, we want to proceed and implement the next set of business logic only if we have a response from the API call. Let us now introduce you to the idea of adding 2 branches to an activity. In this case, we will add one branch to Invoke REST activity with condition that says status != 200 and log that there was a error in API call. We will add another branch that checks if the status is 200 OK and send the locations of fuel stops to the truck driver.



NOTE - Activity id for REST activity is rest_3 and condition is `$activity[rest_3].status != 200`

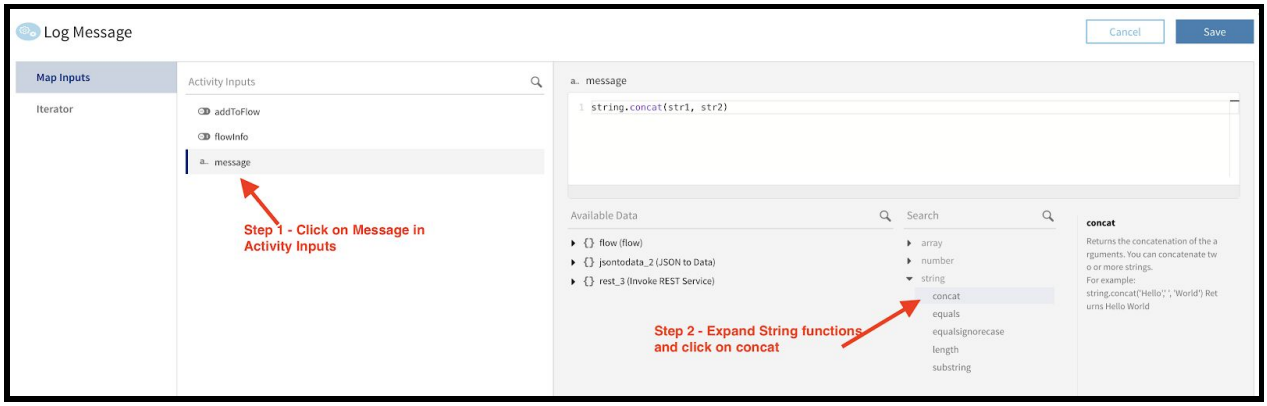
13) Activities in the flow with condition status !=200 will only be invoked if there was an error calling the API. Let's add a log message activity to see errors (if any).

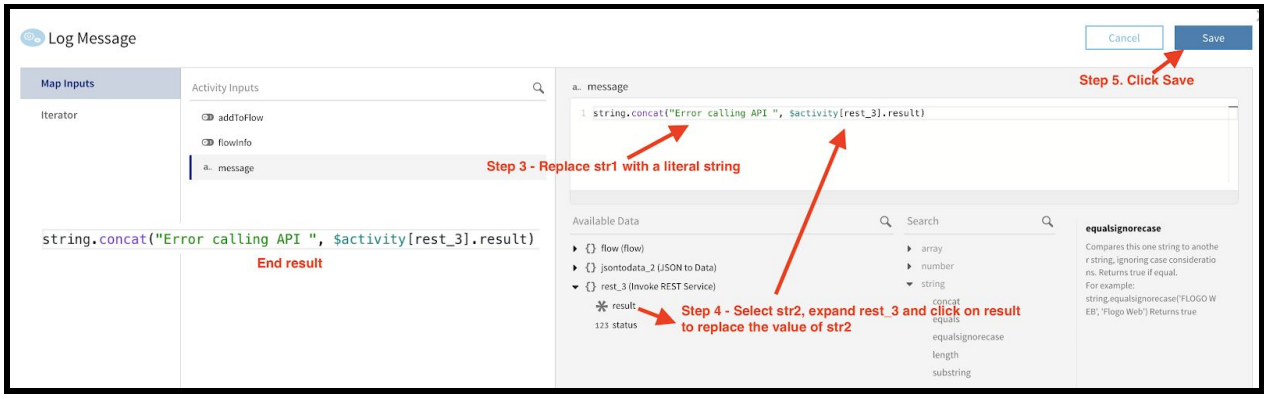
Add by clicking + , filter to find the Log Message Activity and then start configuring Log Message Activity

14) In the configuration for Log Message, let's look at some out of the box functions and use them. We will use String concat function to report an error message whenever the API call to locate fuel stations fail.
Here's the mapping we need to you set in the mapper.

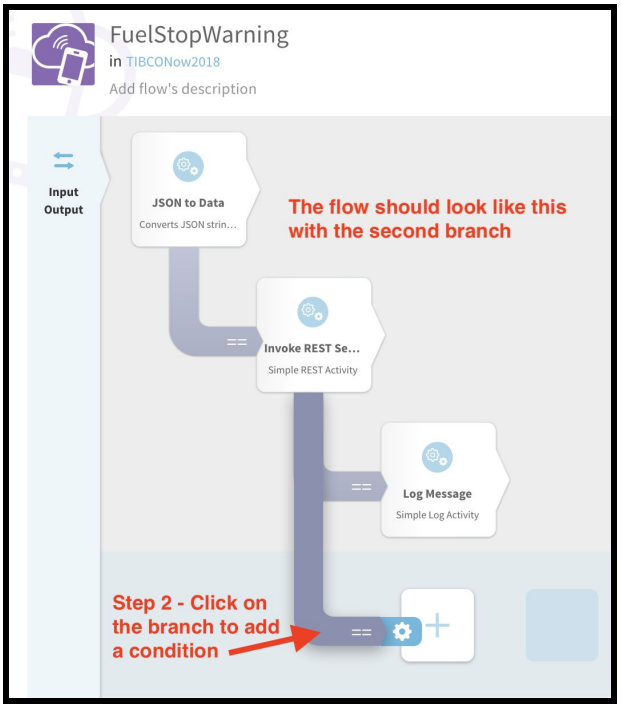
```
string.concat("Error calling API ", $activity[rest_3].result)
```

NOTE - The Invoke REST Service activity id is rest_3. If it is different in your case use the correct one. Take a look at the screenshots for specific instructions for the mapper.



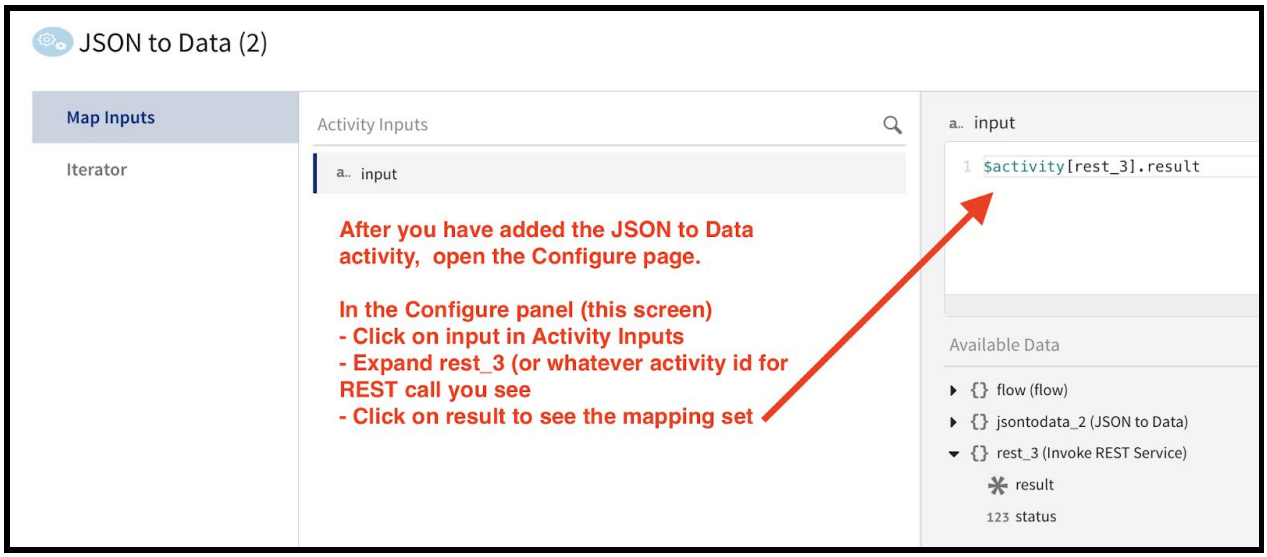


15) Now, let's consider the case of a successful API call. For that, we will first add a second branch with condition that checks if the status is 200.
Note: the condition to add will be `$activity[rest_3].status == 200`

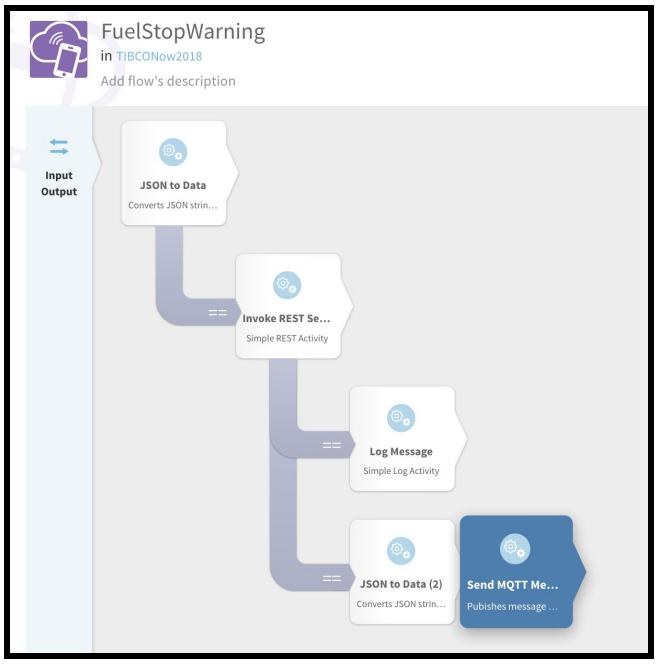


16) The response from the API call is a JSON string. We will now use the JSON to Data activity to map the JSON string from the REST service and create a JSON object. So, what do we need?
Let's start by creating a JSON to Data activity by clicking on the + on the branch that will be executed when we have a successful API call.

17) We are going to **skip** the **instructions that specifically talk about how to add the JSON to Data activity**. **You will add it on your own.** Once you have the activity, note it's activity id from the top right. You will need this later. Add the activity, hover over it to click on configure and set the input to the result from the REST service.

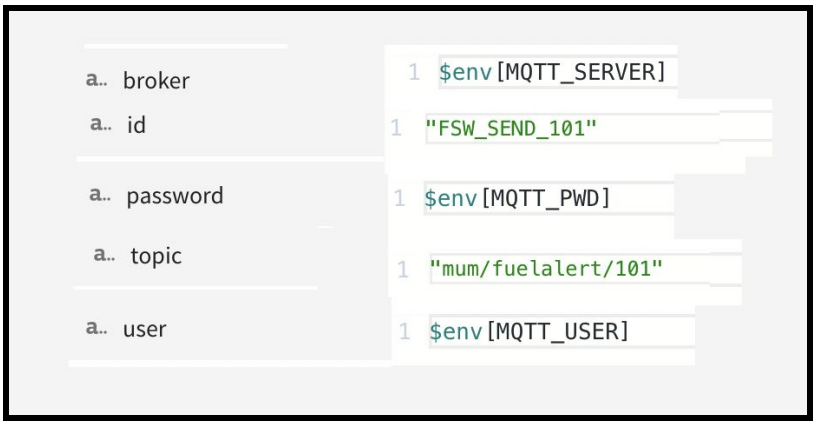


18) One last step is to send a fuel alert on MQTT topic so that truck driver can receive fuel station locations. The first step for this is to add the Send MQTT Message activity. After you add the Send MQTT message activity your flow will look complete as the screenshot shows:

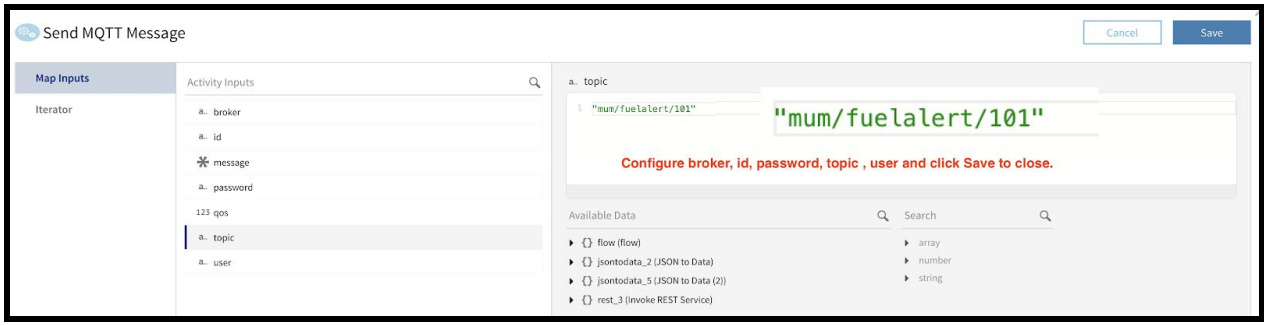


We now have to configure the Send MQTT Message. The settings are the same as before except for the id and topic.

broker = $env[MQTT_SERVER]
id = `"FSW_SEND_nnn"`
where `nnn` is your participant ID. You should know your participant Id by now. The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. FSW_SEND_`nnn` represents the MQTT message send from Fuel Stop Warning Flow , hence FSW_SEND in the beginning.



user = $env[MQTT_USER]
password = $env[MQTT_PWD]
topic = `"mum/fuelalert/nnn"` - where `nnn` is your ParticipantId.



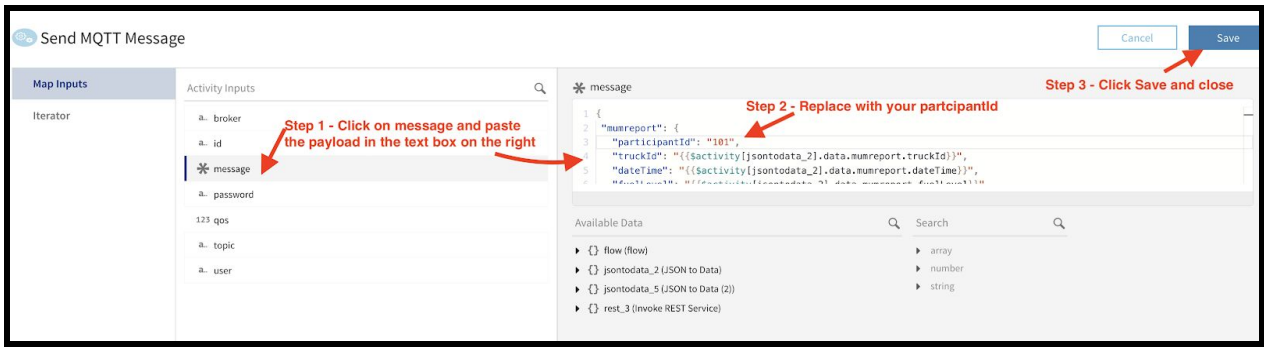
19) Now we will construct the value for the **message** field.

NOTE : The participantId should be set to your participantId before the payload is pasted to the message field.

There are two JSON to Data activities in this flow. Each of them has its own id. The sample payload uses jsontodata_2 for the top level truck information and jsontodata_5 for the fuel station location information. The fuel station location information is something we received from the REST service.

NOTE - The two different JSON to DATA activities and their numbers !! These activity values may be different if you added additional activities!

```
{
 "mumreport": {
   "participantId": "nnn",
   "truckId": "{{$activity[jsontodata_2].data.mumreport.truckId}}",
   "dateTime": "{{$activity[jsontodata_2].data.mumreport.dateTime}}",
   "fuelLevel": "{{$activity[jsontodata_2].data.mumreport.fuelLevel}}",
   "position": {
     "lat": "{{$activity[jsontodata_2].data.mumreport.position.lat}}",
     "long": "{{$activity[jsontodata_2].data.mumreport.position.long}}",
     "speed": "{{$activity[jsontodata_2].data.mumreport.position.speed}}"
   },
   "fuellocation": {
     "name": "{{$activity[jsontodata_5].data.DisplayName}}",
     "lat": "{{$activity[jsontodata_5].data.Latitude}}",
     "long": "{{$activity[jsontodata_5].data.Longitude}}",
     "phone": "{{$activity[jsontodata_5].data.Phone}}"
   }
 }
}
```



To build, run and test the application follow the instructions provided in the <build> section.

Third Challenge: Flogo Microservice for Rest Stop detection

By now, you should be getting comfortable using Flogo and also have a better understanding of the use cases and what we are doing. This challenge involves using data coming from Driver Attention Data (DAD) to decide if the truck driver needs to stop for a break and suggesting rest stops that are closer where the truck is.

As you execute this challenge you will do the following:

1) Receive attention data published in JSON format on a MQTT topic (`mum/attention`)
2) Call a AWS Lambda function that will decide if driver needs to rest (built by us)
3) Suggest the driver to stop if the Lambda function decides that's what the driver needs to do. TIBCO Cloud based service will suggest nearest coffee shops
4) Send to MQTT (topic `mum/attentionalert/<participantId>`) the nearest coffee shop location in the response format

Here's the sample message / structure you will receive on MQTT topic `mum/attention`

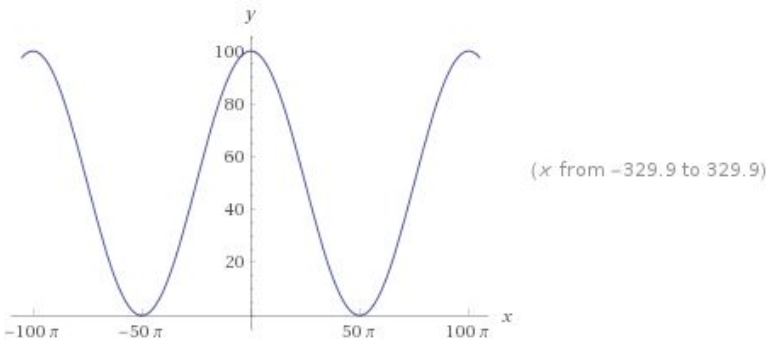
```
{
  "mumreport": {
    "truckId": "1",
    "journeyTime": "3435352424", //how long trip is going
    "dateTime": "2018-07-1221:39:49.356+0200", // current time
    "odometer": "455",
    "fuelLevel": "50.0",    // decreases over time
    "attentionLevel": "20", //  Cycles but slowly decreases over time
[1]
    "position":
{
        "lat": "47.34",
        "long": "23.34",
        "speed": "34.0"
      }
  }
}
```

As we execute this flow, we will send the following response to the truck drivers and suggest them to take a break at the rest stop.

```
{
  "mumreport": {
    "participantId": "100",
    "truckId": "1",
    "dateTime": "2018-07-1221:39:49.356+0200",
    "fuelLevel": "20.0",
    "journeyTime" : "3435352424",
    "attentionLevel" : "10",
    "position": {
      "lat": "47.34",
      "long": "23.34",
      "speed": "34.0"
    },
    "restlocation": {
      "name": "Renata",
      "lat": "47.40451",
      "long": "23.02657",
      "phone": "269-432250"
    }
  }
}
```

Attention level

The attention level is checked when the attention level is < 30 and the journey duration is > 20. The driver starts with 100% attention level and the value typically goes in a decreasing cyclic wave.



For e.g.

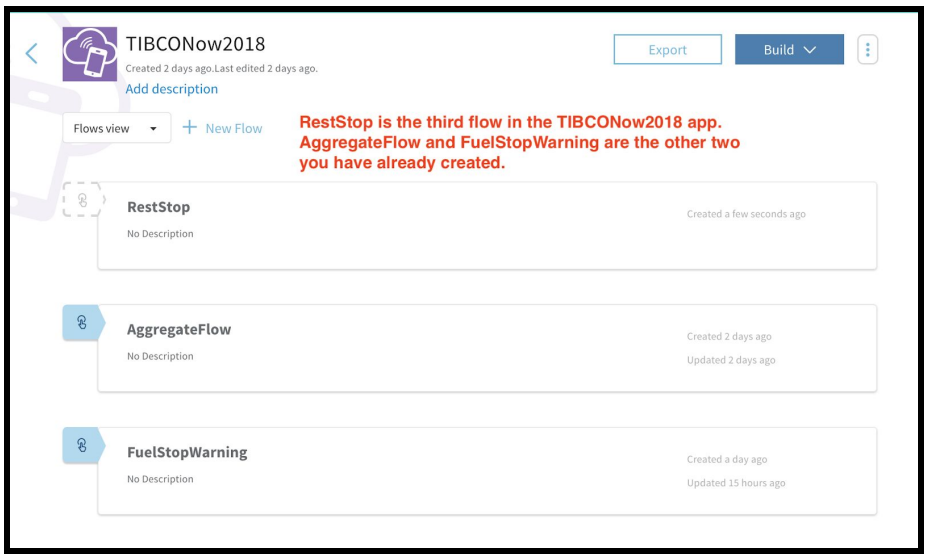
100,90,80,70,60,70,80,90,80,70,60,50,60,70,60,50,40,50,60,70,80…

plot	$\left(\frac{1}{2}(\cos(0.02\,x)+1)\right)\times 100$

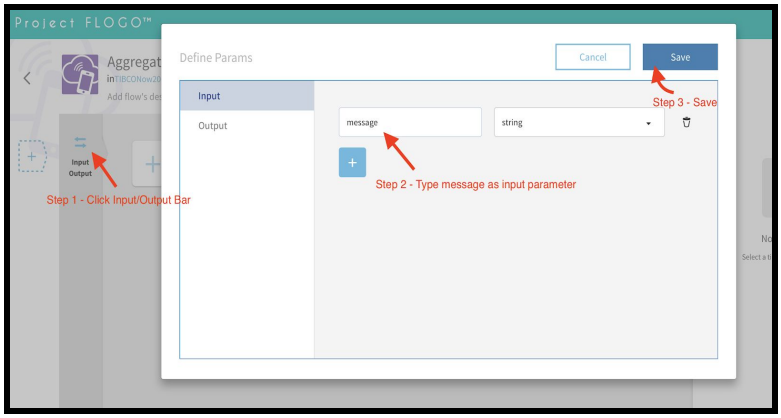
The attention level drops to 0 every 50*π (157 km or ~ 100 miles). We use the period (0.02*x , where x is the odometer value) to compute the attention level.

NOTE - We will now build a new flow called RestStop and will be adding activities just like we did the last 2 challenges. The first challenge was pretty detailed in terms of instructions and second challenge assumed some knowledge. In the third challenge, we will make some assumptions about your knowledge of Flogo and may combine one or more instructions together. Don't worry it's not complex. We are here to help you follow along.

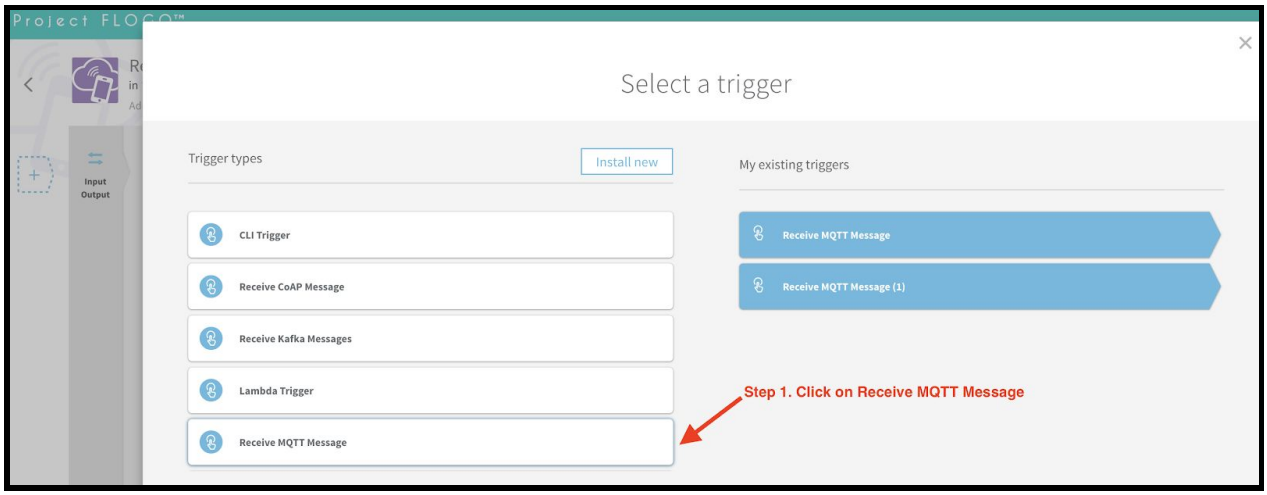
1) Click on New Flow to add a new flow and call it RestStop.



2) Start by creating the flow input/output settings.
Click on the input output bar enter the parameter name of "message" (only entering the input param is needed). This will be used to pass data from the trigger to the flow.. **Click Save**

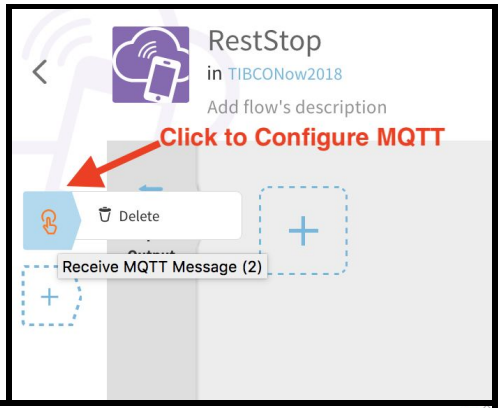


3) Next, we are going to add a trigger to the flow. From the last two challenges, we know that adding a trigger to your flow is very simple. First click on RestStop to enter the flow and then Click on the + button on the left most side to open the trigger view.

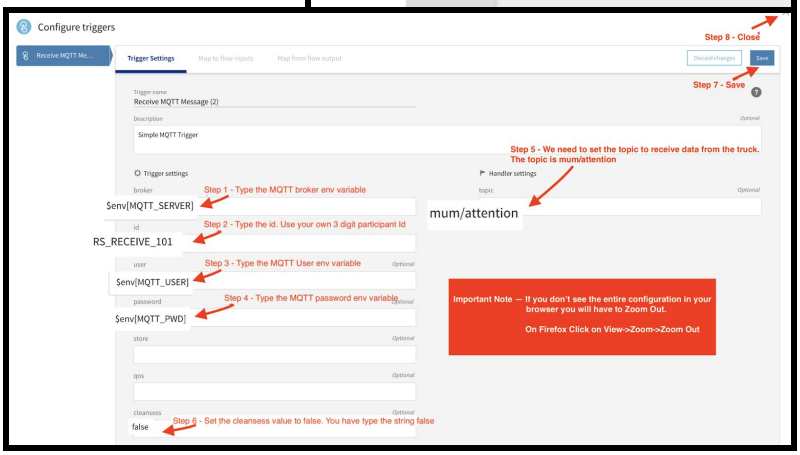


Here you will add a new Receive MQTT message. You will notice that the flow will provide you an option to reuse couple of existing triggers. These were the ones we created in the previous flows. Ignore it and add a new one as we have a couple of different settings to change.

Once added, click on Configure



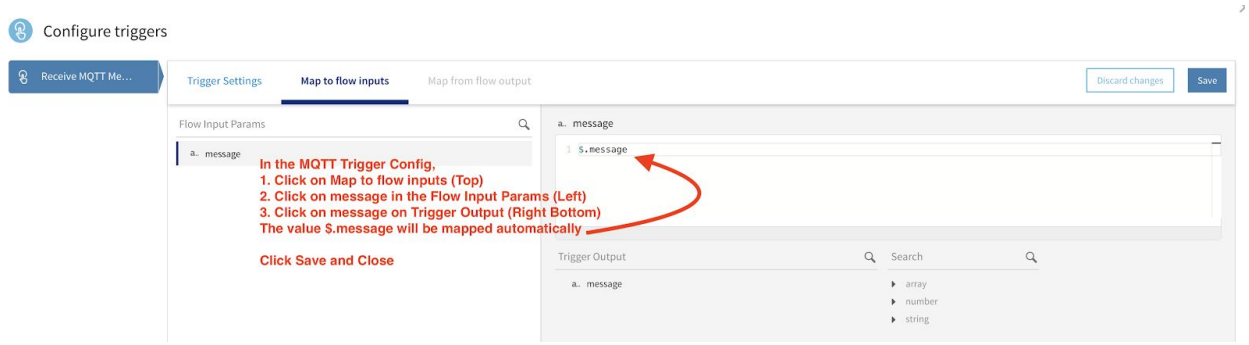
4) Set the values for the MQTT Trigger.



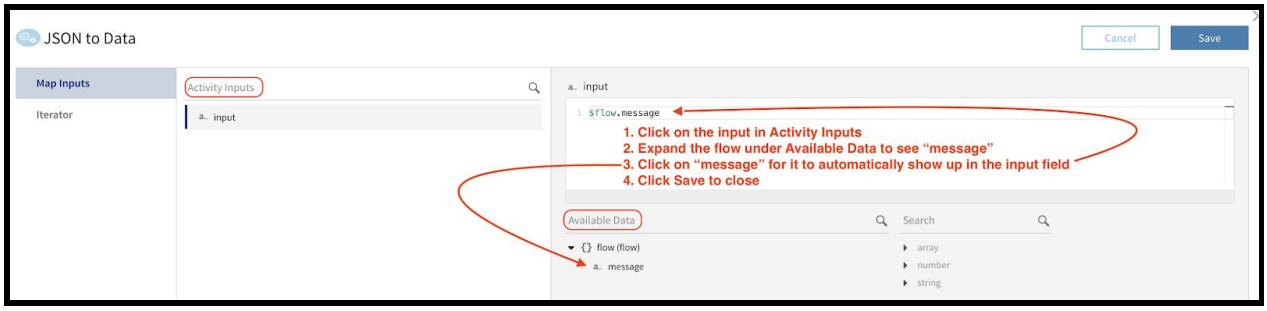
broker = $env[MQTT_SERVER]
id = RS_RECEIVE_*nnn* nnn is your participant ID. You should know your participant Id by now.
The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. RS_RECEIVE_*nnn* represents the MQTT message receive on Rest Stop Flow , hence RS_RECEIVE in the beginning.
user = $env[MQTT_USER]
password = $env[MQTT_PWD]

topic = mum/attention
cleansess = false
NOTE: You should be using your own participantId**.**

5) Continue to configure the MQTT Trigger's Map to flow inputs.



6) Add a new JSON to Data Activity, go to the Configure panel and map the message payload to the input.



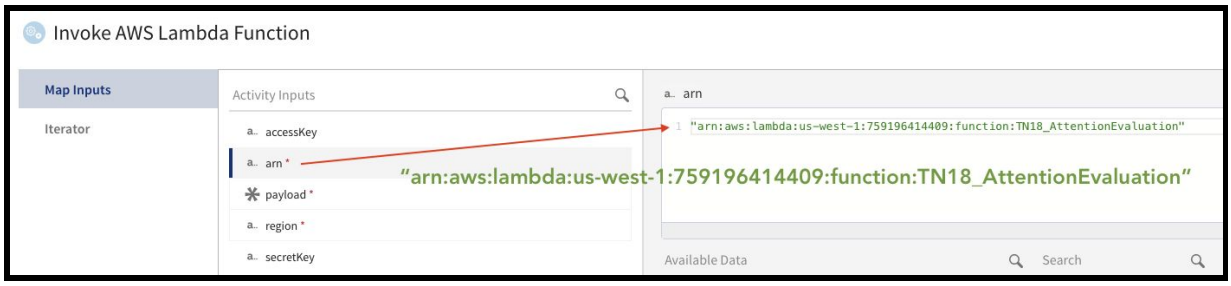
7) Now we are going to add a new activity that we haven't used before. **Invoke AWS Lambda** . We will call a serverless function that is going to run our algorithm that uses data about the journey and the driver to determine if our driver should take a rest.

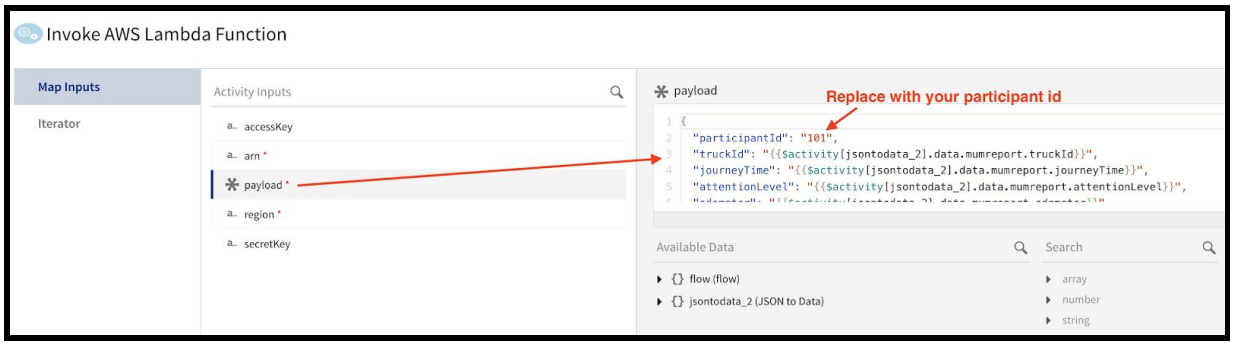
To call the Lambda function that has been deployed for you, you need to specify the AWS region it's deployed to and the unique id or ARN. The AMI has the your access key and secret access key already configured for the lambda function we are going to call.
Here are the values handy for you to paste. Always exercise caution when pasting values.
Arn: `"arn:aws:lambda:us-west-1:759196414409:function:TN18_AttentionEvaluation"`

Region: `"us-west-1"`

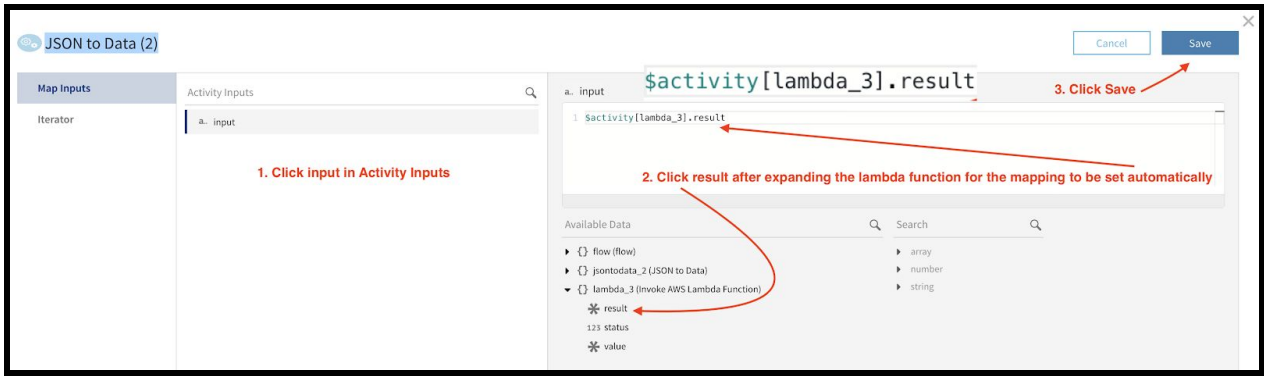
Payload: < Change the participant Id in the text below !>

```
{
 "participantId": "nnn",
 "truckId": "{{$activity[jsontodata_2].data.mumreport.truckId}}",
 "journeyTime": "{{$activity[jsontodata_2].data.mumreport.journeyTime}}",
 "attentionLevel": "{{$activity[jsontodata_2].data.mumreport.attentionLevel}}",
 "odometer": "{{$activity[jsontodata_2].data.mumreport.odometer}}"
}
```



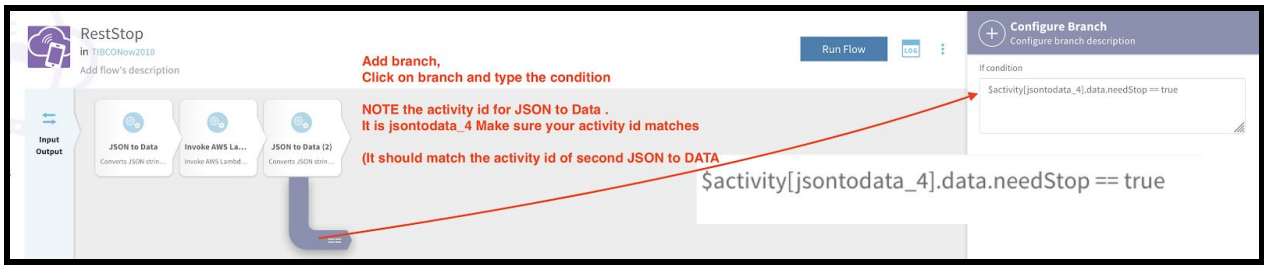


8) Add another JSON to Data activity (No instructions to add. You already know how to add an activity). Note the activity id. You know how to find it. This time map the result from the Lambda invoke to the input



9) Add a branch from the JSON to Data activity and add a condition that checks if the driver needs to stop. Here's the condition to copy

```
$activity[jsontodata_4].data.needStop == true
```



10) If our serverless compute function running on AWS Lambda recommends that the driver needs to stop, then we want to invoke a REST service that can help us find nearest coffee shops. Here's the values handy for you to configure the REST service. Note that the activity id for JSON to Data used in the payload is jsontodata_2. Make sure you are using the activity id from the first JSON to Data activity. You can look at the screenshot too.

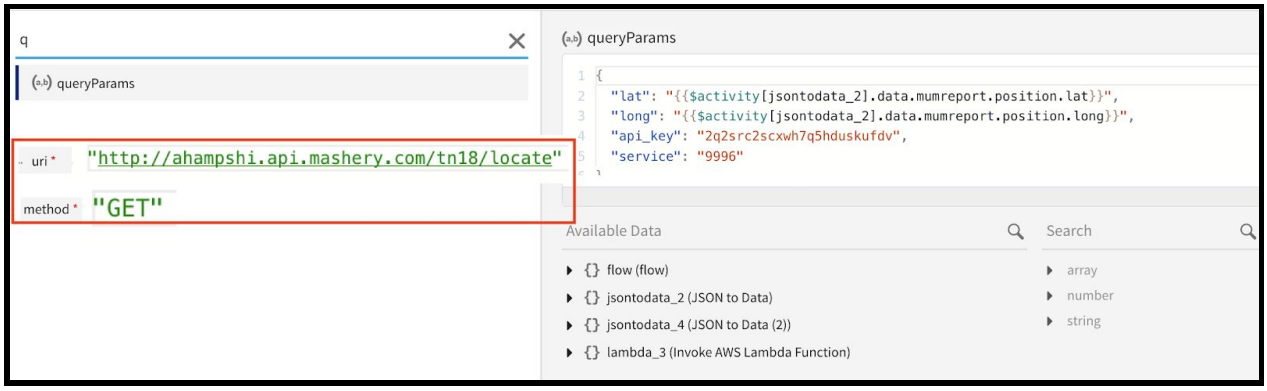
Method: `"GET"`

uri: `"http://ahampshi.api.mashery.com/tn18/locate"`

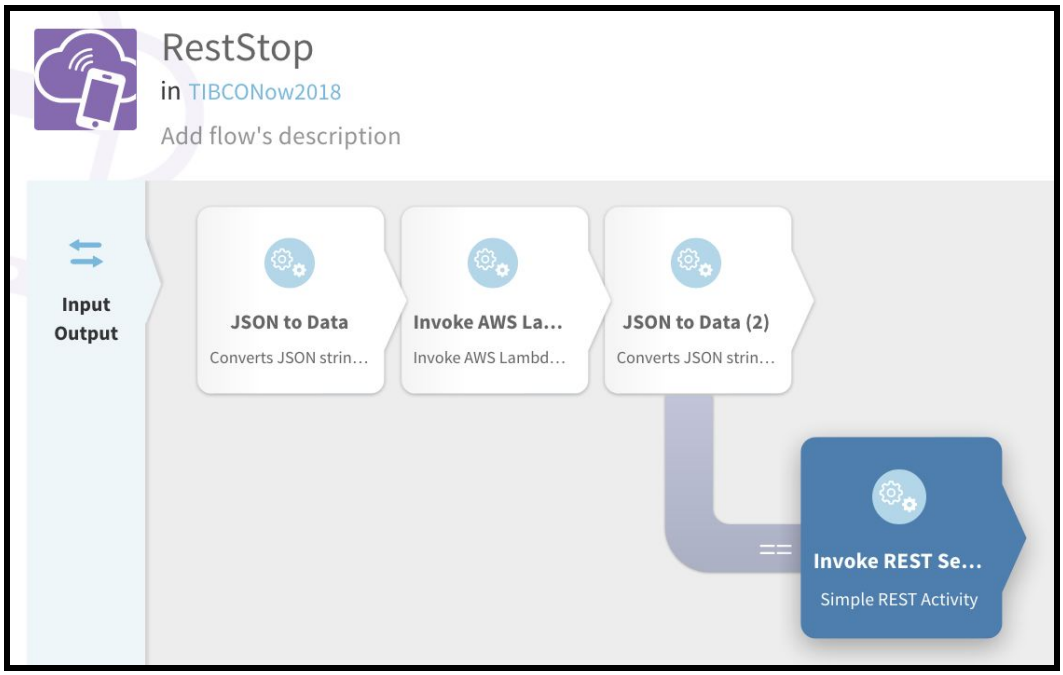
Queryparams:

```
{
 "lat": "{{$activity[jsontodata_2].data.mumreport.position.lat}}",
 "long": "{{$activity[jsontodata_2].data.mumreport.position.long}}",
 "api_key": "2q2src2scxwh7q5hduskufdv",
 "service": "9996"
}
```

NOTE - The required data comes from the FIRST "JSON to Data" activity !!



11) At this time the flow should be looking like this. If it's **NOT** looking like this, you did something wrong. Please retrace your steps.

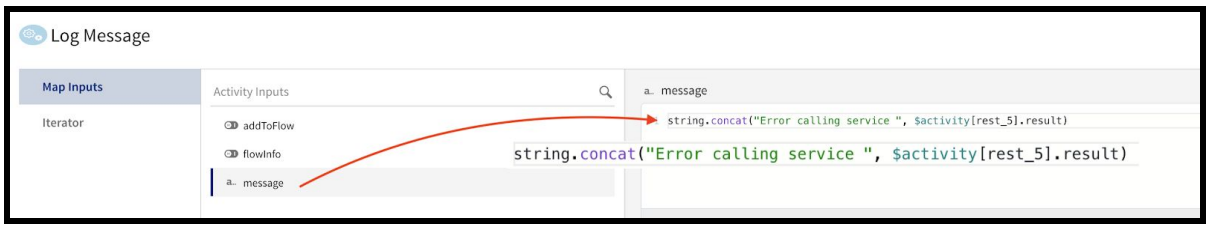


12) We are now going to add two branches from the Invoke REST Service. This will check the status code returned by the service. **Note**: The activity name (rest_5) may be different for you! Check the activity id for the Invoke REST service and use it in the branch condition.
Set one branch condition to: `$activity[rest_5].status!= 200`
And the other condition to: `$activity[rest_5].status == 200`

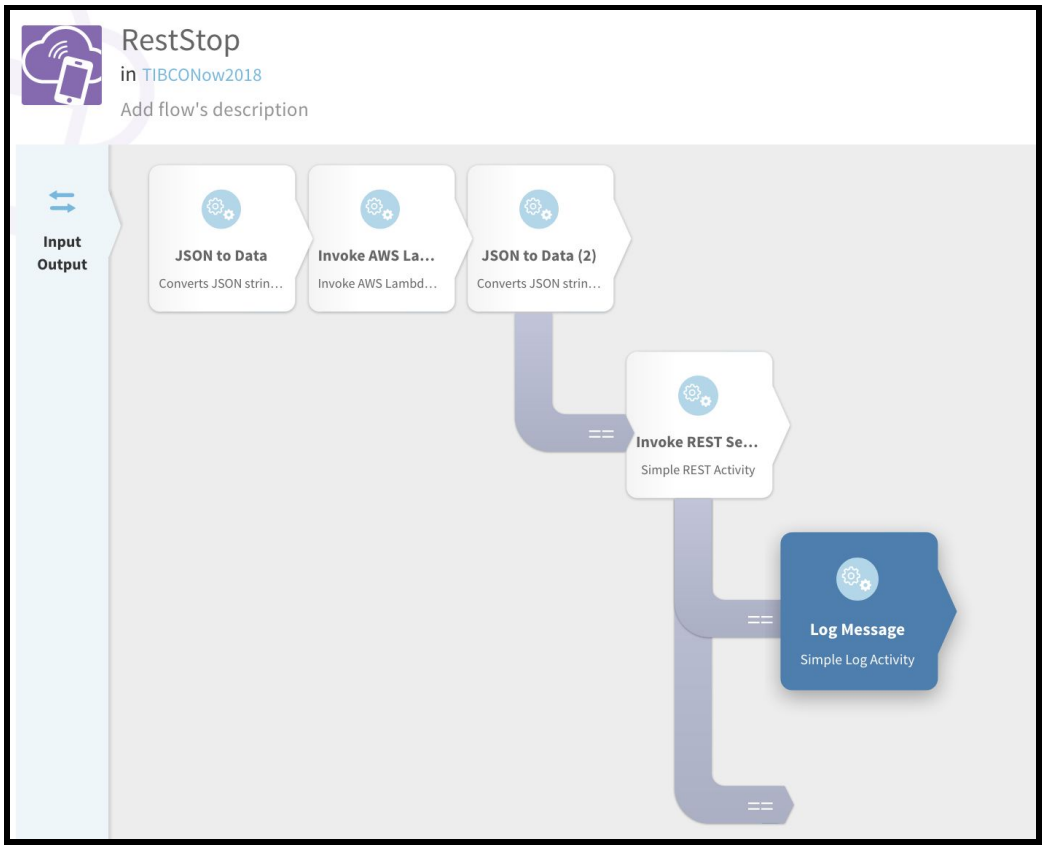
When the status code is not 200 lets use a Log Message to output the error so we can diagnose why it's failed. This is exactly what we did in the second challenge for Invoke REST service.

Set the message to something like:

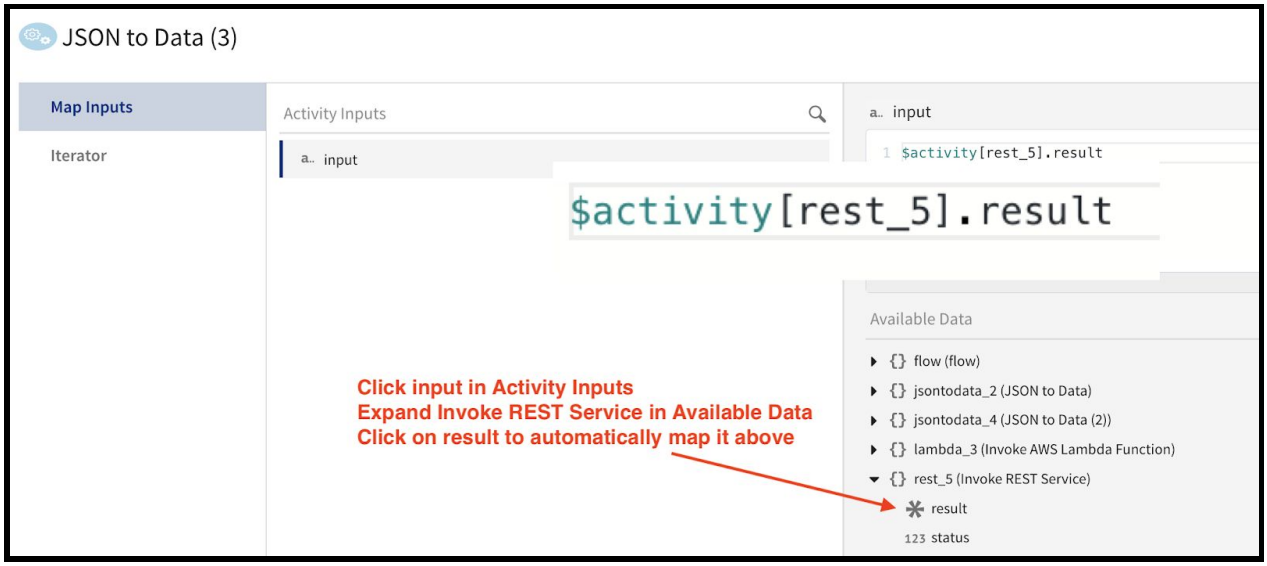
```
string.concat("Error calling service ", $activity[rest_5].result)
```



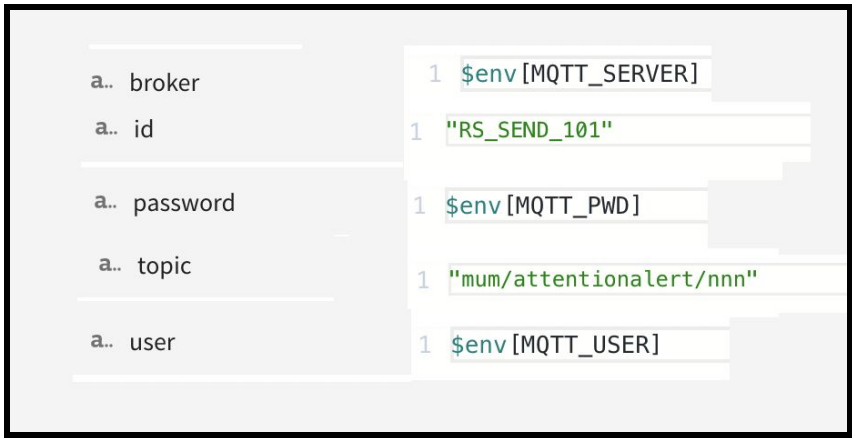
This is how your flow should look at this time.



13) Now, let's handle the case when we have successful invocation of the REST service. Just like the previous challenge, in the second branch, we will first add a JSON to Data activity and then a Send MQTT message activity to notify the driver. First, let's add the JSON to Data Activity. This will be your third JSON to Data activity. Let's note the activity id.



14) On to our very last step for the basic challenge. Let's add Send MQTT message activity and configure it.



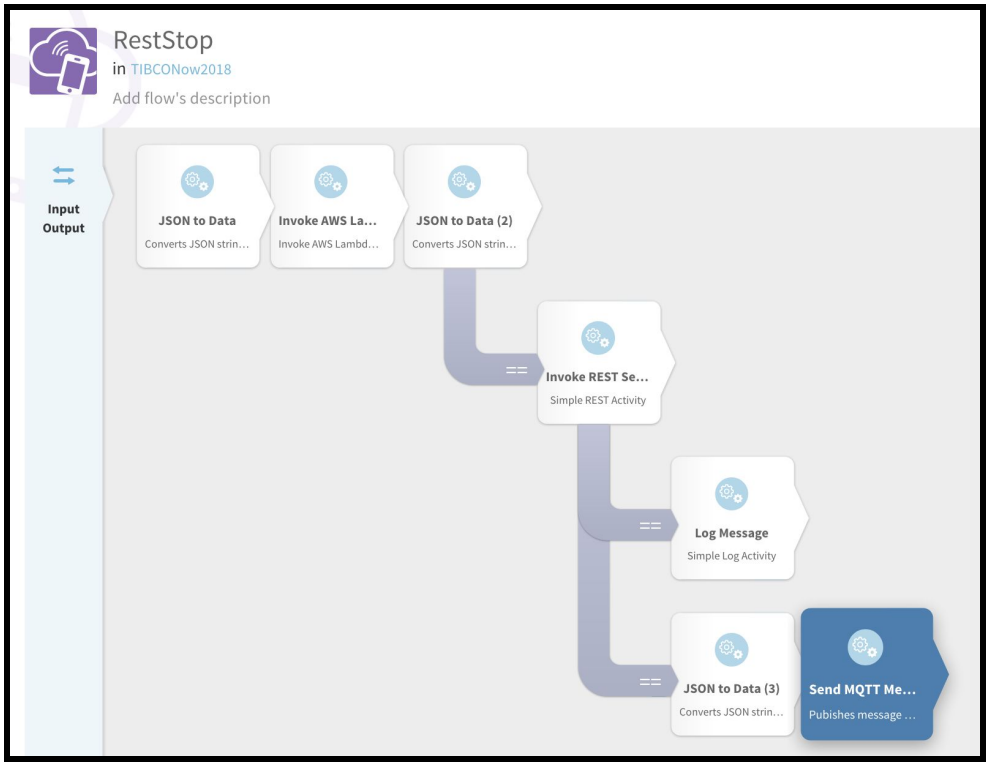
broker = $env[MQTT_SERVER]
id = `"RW_SEND_nnn"` where `nnn` is your participant ID. You should know your participant Id by now. The Id field needs to be a unique value as the MQTT broker tracks users via the client ID. RS_SEND_*nnn* represents the MQTT message send from Rest Stop Flow , hence RS_SEND in the beginning.
user = $env[MQTT_USER]
password = $env[MQTT_PWD]
topic = `"mum/attentionalert/nnn"` - Where nnn is your participantId

To map the message field we will be using the first JSON to Data Activity (jsontodata_2) and the last JSON to Data activity (jsontodata_7)

message: < Change the participant Id in the text below> and copy paste into the message field.

```
{
 "mumreport": {
   "participantId": "nnn",
   "truckId": "{{$activity[jsontodata_2].data.mumreport.truckId}}",
   "journeyTime": "{{$activity[jsontodata_2].data.mumreport.journeyTime}}",
   "dateTime": "{{$activity[jsontodata_2].data.mumreport.dateTime}}",
   "fuelLevel": "{{$activity[jsontodata_2].data.mumreport.fuelLevel}}",
   "attentionLevel": "{{$activity[jsontodata_2].data.mumreport.attentionLevel}}",
   "position": {
     "lat": "{{$activity[jsontodata_2].data.mumreport.position.lat}}",
     "long": "{{$activity[jsontodata_2].data.mumreport.position.long}}",
     "speed": "{{$activity[jsontodata_2].data.mumreport.position.speed}}"
   },
   "restlocation": {
     "name": "{{$activity[jsontodata_7].data.DisplayName}}",
     "lat": "{{$activity[jsontodata_7].data.Latitude}}",
     "long": "{{$activity[jsontodata_7].data.Longitude}}",
     "phone": "{{$activity[jsontodata_7].data.Phone}}"
   }
 }
}
```

If you have managed to complete the steps the flow should look like:



You should now be able to run the full app and validate it. **Note**: This use case may take a few minutes to give an alert, so if you don't see it right away, be patient.

Additional Challenge - Notify an alert on the driver's phone

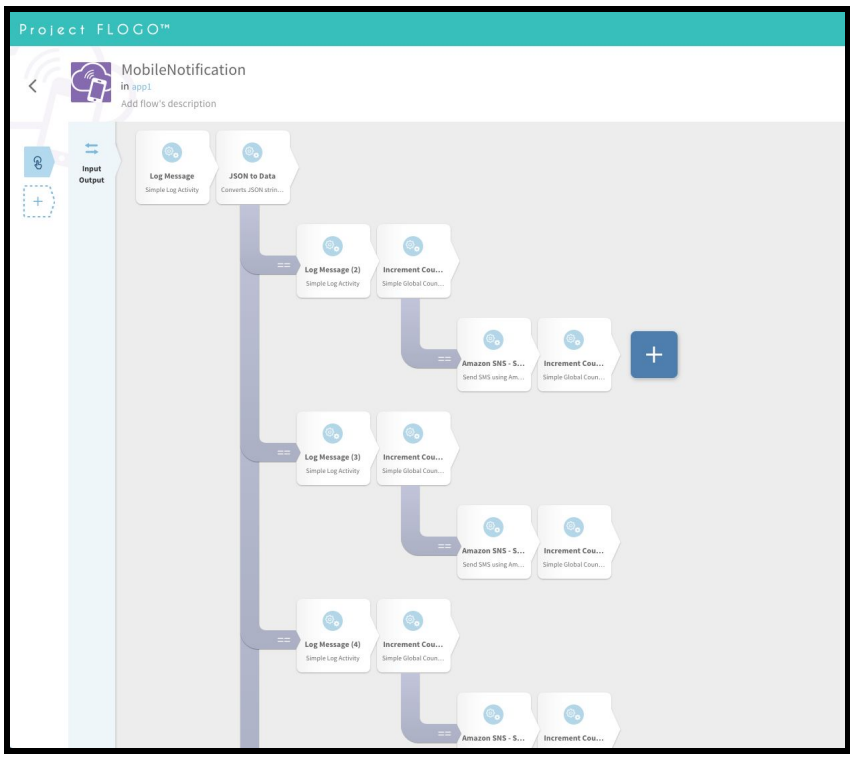
1) Add a step in the Flogo flow or create a new flow to notify an alert on the driver's mobile phone to raise the attention level if below a certain threshold. Hint: use AWS SNS

Instructions - Highlights:

1) Start off by creating a new flow called Mobile Notification.
2) Enter an input variable called message, and create a MQTT trigger.
3) Set the following for the MQTT variables: You have done it few times now.
    a) Broker = $env[MQTT_SERVER]
    b) Id = MN_RECEIVE_nnn
    c) User = $env[MQTT_USER]
    d) Password = $env[MQTT_PWD]
    e) Topic = mum/attention/nnn (*nnn* is your participant ID)
    f) Cleansess = false
4) Map the input within the trigger as well. (Same thing you've been doing in the other challenges)
5) Your first activity should be a log message so that we know that we are actually getting messages sent. Create a log activity and map the flow message to the input message. (similar to what you've been doing)
6) Following the log message, add a "json to data" activity. Map the flow message to the input of this activity.
7) Now, let's add 5 branches out of that json to data activity. The reason to do so is because there are 5 truckid's that are broadcast from that mqtt topic.
8) For the 5 branches you need to add an if condition like: $activity[jstontodata_3].data.mumreport.truckId == VARIABLE
    a) Replace VARIABLE above with one of the following id's in each branch:
        i) 78655
        ii) 78617
        iii) 78604
        iv) 78926
        v) 78615
9) Create log message at the end of each branch, in the message box put something like "TruckID: VARIABLE - Infraction Logged", where VARIABLE in the truckID that matches the branch.
10) Following the log activities in all branches, add an increment counter. On the right hand side, set the counterName input to anything you like (I used something like truckVARIABLE where VARIABLE is one of the 5 truck ids) and set the increment to true. EACH COUNTER NAME SHOULD BE DIFFERENT FOR EACH BRANCH.
11) Now the fun part, add branches to the end of each increment counter. The if condition should be something like $activity[counter_X].value == 2 where X is equal to the ID of the counter activity. That means each branch will have a different counter_X value. You can find the ID of your counter if you click on the activity, and look on the right hand side (where you configured the name and increment earlier), near the name.

12) Add an Amazon SNS activity at the end of each newly created branch (so 5 in total). Note you will have to hard code your access and secret key to the activity. At the moment, this activity does not pick up these values from your .aws folder. On the right hand side there are a few configurations to be made, so let's set those.
    a) accessKey = <ask_us> (Yes actually ask us)
    b) secretKey = <ask_us> (Yes actually ask us)
    c) Region = us-east-1
    d) smsType = Promotional
    e) From 58800
    f) To = PHONE_NUMBER where PHONE_NUMBER is your cell number with +1 added to the front of the area code (US NUMBERS ONLY). Example: +19998789999
    g) Leave the message on the right hand side configuration blank.
13) Now let's configure the message, hover over the SNS activity and click on configure. You will get the configuration window that you are used to. Click on the message input. Add something like this as the message:
```
string.concat($activity[jsontodata_3].data.mumreport.truckId, " - Please consider taking a break at: ",
$activity[jsontodata_3].data.mumreport.restlocation.name)
```
    This should be the same for all the SNS activities.
14) Add an increment counter at the end of each SNS activity. Set the counterName to the same name used previously in this branch. Set the increment value to false and reset value to true.
15) Build your app, and run it along side the basic challenge apps. Now every two recorded instances of low fuel will result in a notification to that individual truck. If you leave it running for a bit, you should see text messages being sent to your phone

How the flow should look like as shown here.



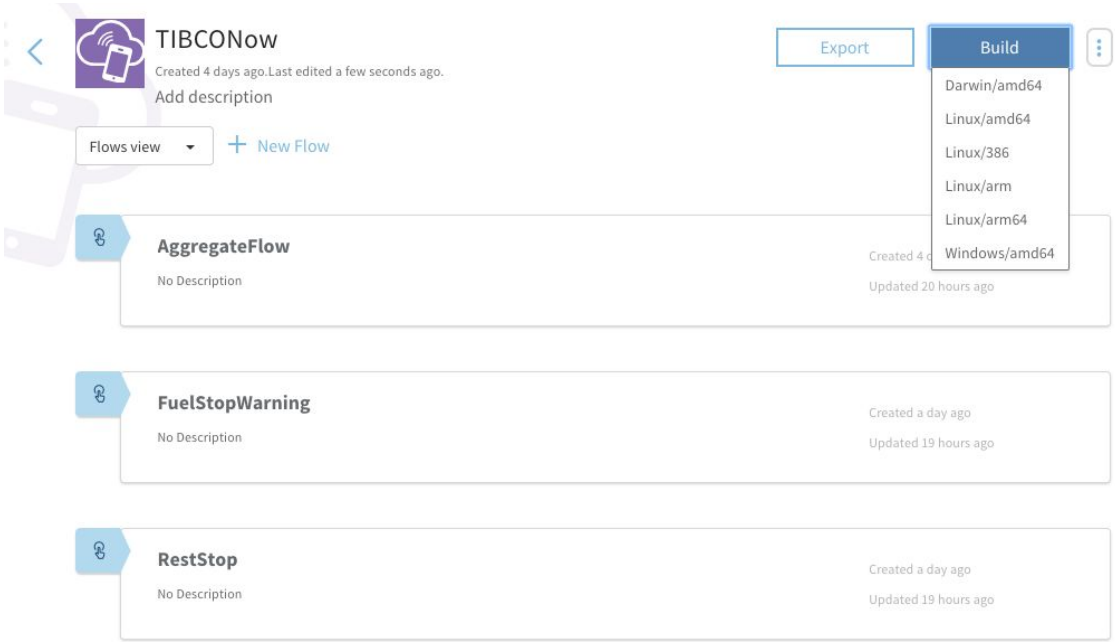
Appendix A - Building, running and testing the Application

At this stage you should test the application and verify that you are getting messages from Flogo on your MQTT topic. If successful, you will start to see number values appear in your terminal (this may take a few seconds). See instructions below to build and run.

```
Criss-MBP:Downloads crmadrig$ ./tibco-now-2018_darwin_amd64
2018-08-20 09:38:07.882 INFO   [engine] - Engine Starting...
2018-08-20 09:38:07.883 INFO   [engine] - Starting Services...
2018-08-20 09:38:07.883 INFO   [engine] - Started Services
2018-08-20 09:38:07.883 INFO   [engine] - Starting Triggers...
2018-08-20 09:38:08.433 INFO   [engine] - Trigger [ receive_mqtt_message ]: Started
2018-08-20 09:38:08.433 INFO   [engine] - Triggers Started
2018-08-20 09:38:08.433 INFO   [engine] - Engine Started
2018-08-20 09:38:09.777 INFO   [engine] - Running FlowAction for URI: 'res://flow:aggregate_flow'
2018-08-20 09:38:09.779 INFO   [activity-flogo-log] - {"mumreport":{"truckId":"78604","dateTime":"2018-08-20 16:38:08.001+0000","fuelLevel":"30","position":{"lat":"51.7007","long":"5.3204","speed":"103.
45"}}}
2018-08-20 09:38:09.782 INFO   [engine] - Flow instance [3b2c55d32bfddb0ec05432b9be38c6ee] Completed Successfully
2018-08-20 09:38:09.782 INFO   [engine] - Running FlowAction for URI: 'res://flow:aggregate_flow'
2018-08-20 09:38:09.782 INFO   [activity-flogo-log] - {"mumreport":{"truckId":"78926","dateTime":"2018-08-20 16:38:08.001+0000","fuelLevel":"99","position":{"lat":"52.3247","long":"5.0173","speed":"93.7
1"}}}
2018-08-20 09:38:09.782 INFO   [engine] - Running FlowAction for URI: 'res://flow:aggregate_flow'
2018-08-20 09:38:09.782 INFO   [activity-flogo-log] - {"mumreport":{"truckId":"78617","dateTime":"2018-08-20 16:38:08.001+0000","fuelLevel":"59","position":{"lat":"51.9937","long":"5.9707","speed":"85.5
5"}}}
```

In the Flogo UI select the App to be Built.

There are two ways that the app can be built, this doc will show using the built in Build function.



Click on Build then select Linux/386 if you are using Linux, Darwin/amd64 if you are using MacOS and Windows/amd64 for Windows. If you chose to run it on some device (for e.g. Raspberry Pi , select the relevant target)

Your executable will be saved to the browser's downloads folder.

For Linux (Centos/RHEL)

First you will need to make the downloaded file executable..

```
chmod +x tibco-now-2018_linux_386
```

Then set the environment vars to pick up the runtime parameters:
(These are set in the AMI already within the .bashrc file)

```
export MQTT_SERVER="tcp://54.153.123.237:1883"
export MQTT_USER="tibco"
export MQTT_PWD="mqttTIBCO18"
export FLOGO_LOG_LEVEL="INFO"

./tibco-now-2018_linux_386
```

For Apple (MacOS)

First you will need to make the downloaded file executable..

```
chmod +x tibco-now-2018_darwin_amd64
```

Then set the environment variables to be picked up for runtime.

```
export MQTT_SERVER="tcp://54.153.123.237:1883"
export MQTT_USER="tibco"
export MQTT_PWD="mqttTIBCO18"
export FLOGO_LOG_LEVEL="INFO"

./tibco-now-2018_darwin_amd64
```

For Windows Cmd

Set the environment variables and run.

```
set FLOGO_LOG_LEVEL = "INFO"
Set MQTT_SERVER = "tcp://54.153.123.237:1883"
Set MQTT_USER = "tibco"
set MQTT_PWD = "mqttTIBCO18"

Run the exe
> tibco-now-2018_windows_amd64.exe
```

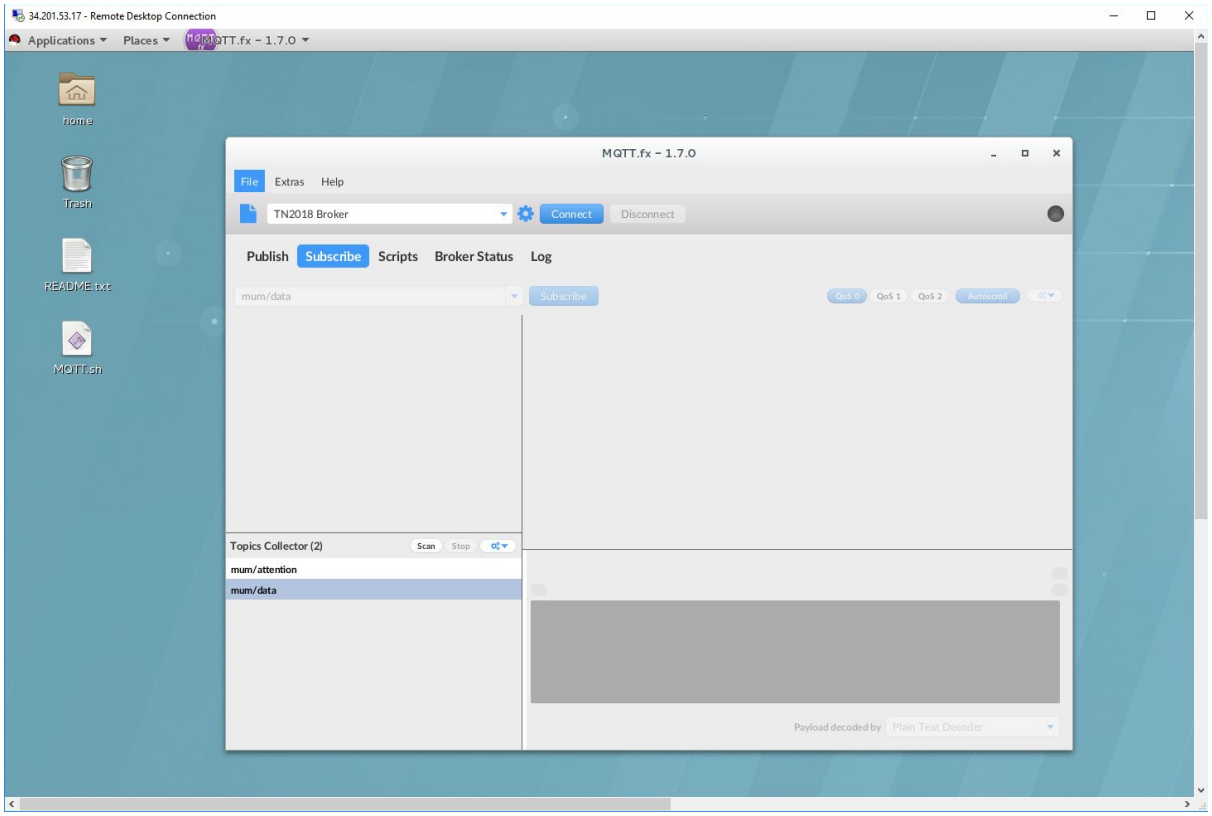
For Windows Powershell

Set the environment variables and run.

```
$env:FLOGO_LOG_LEVEL = "INFO"
$env:MQTT_SERVER = "tcp://54.153.123.237:1883"
$env:MQTT_USER = "tibco"
$env:MQTT_PWD = "mqttTIBCO18"

Run the exe
> .\tibco-now-2018_windows_amd64.exe
```

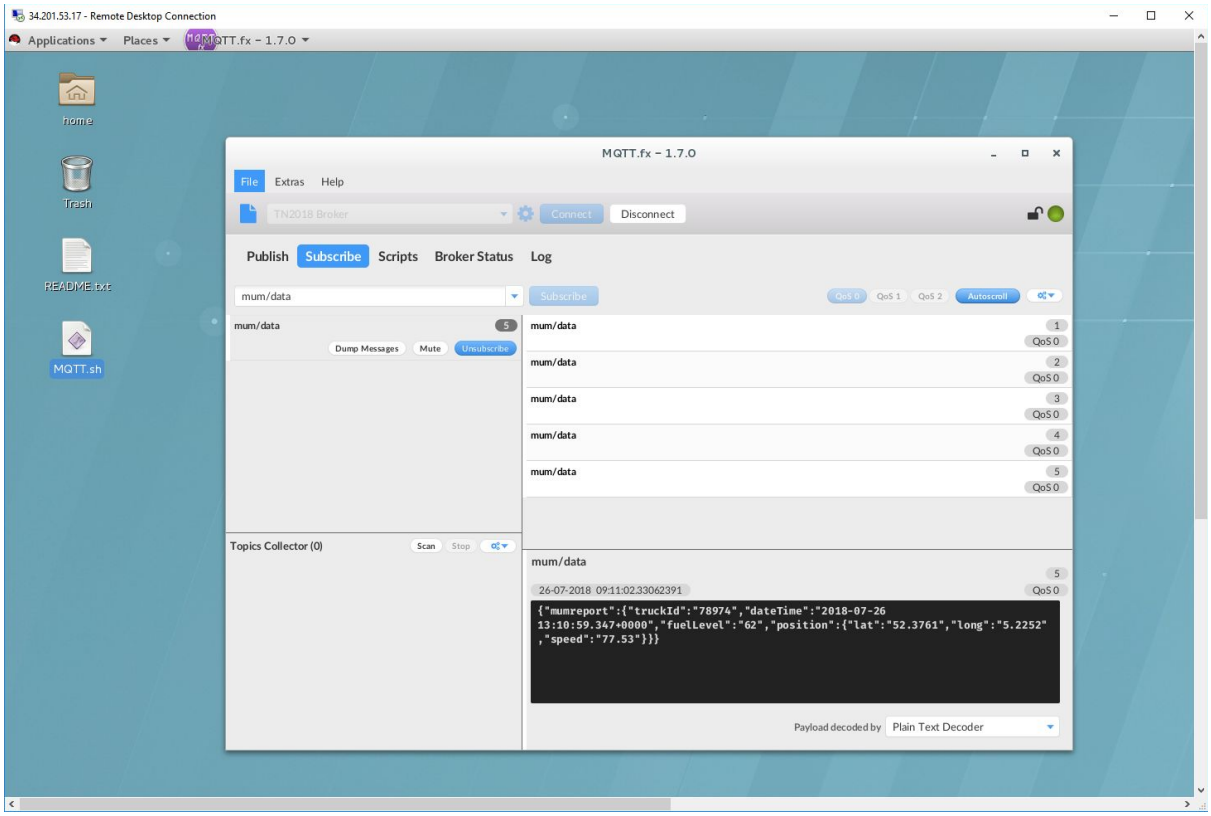
Appendix B - Using MQTT.fx to monitor MQTT messages



The MQTT broker was configured during the install process. If it is not connected, click on connect to connect to the Mosquitto MQTT broker.

On the Subscribe tab enter the name of the topic name. The name of the topic that we are interested is called mum/data. Enter mum/data and click subscribe

You should then see messages appear on the right hand pane

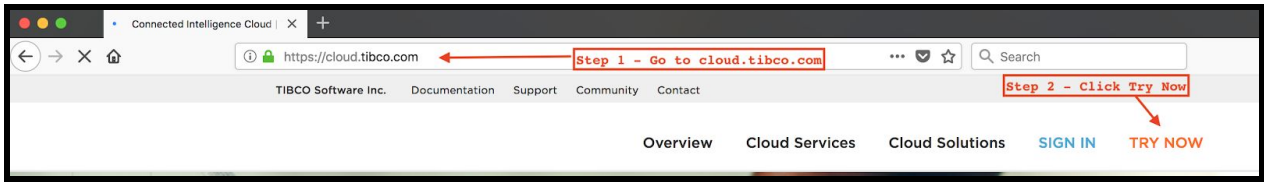


For more information on how to use the tool go to http://mqttfx.jensd.de or search for a MQTT FX Tutorial.

Appendix C - Getting Access to Spotfire

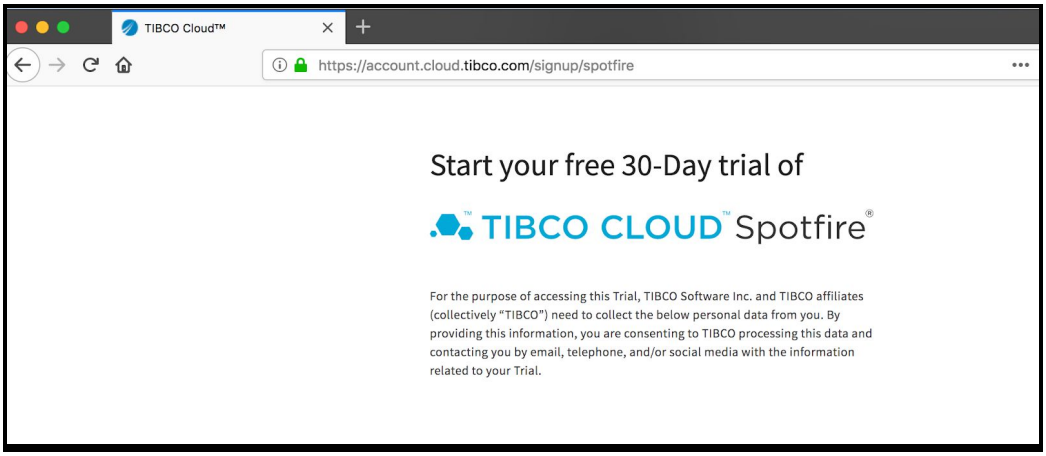
To use the data set that we have provided or any data set that you chose to use for this hackathon, you will need access to Spotfire cloud. It's pretty easy to sign up and get yourself an account.

First, go to cloud.tibco.com and click on "Try Now"

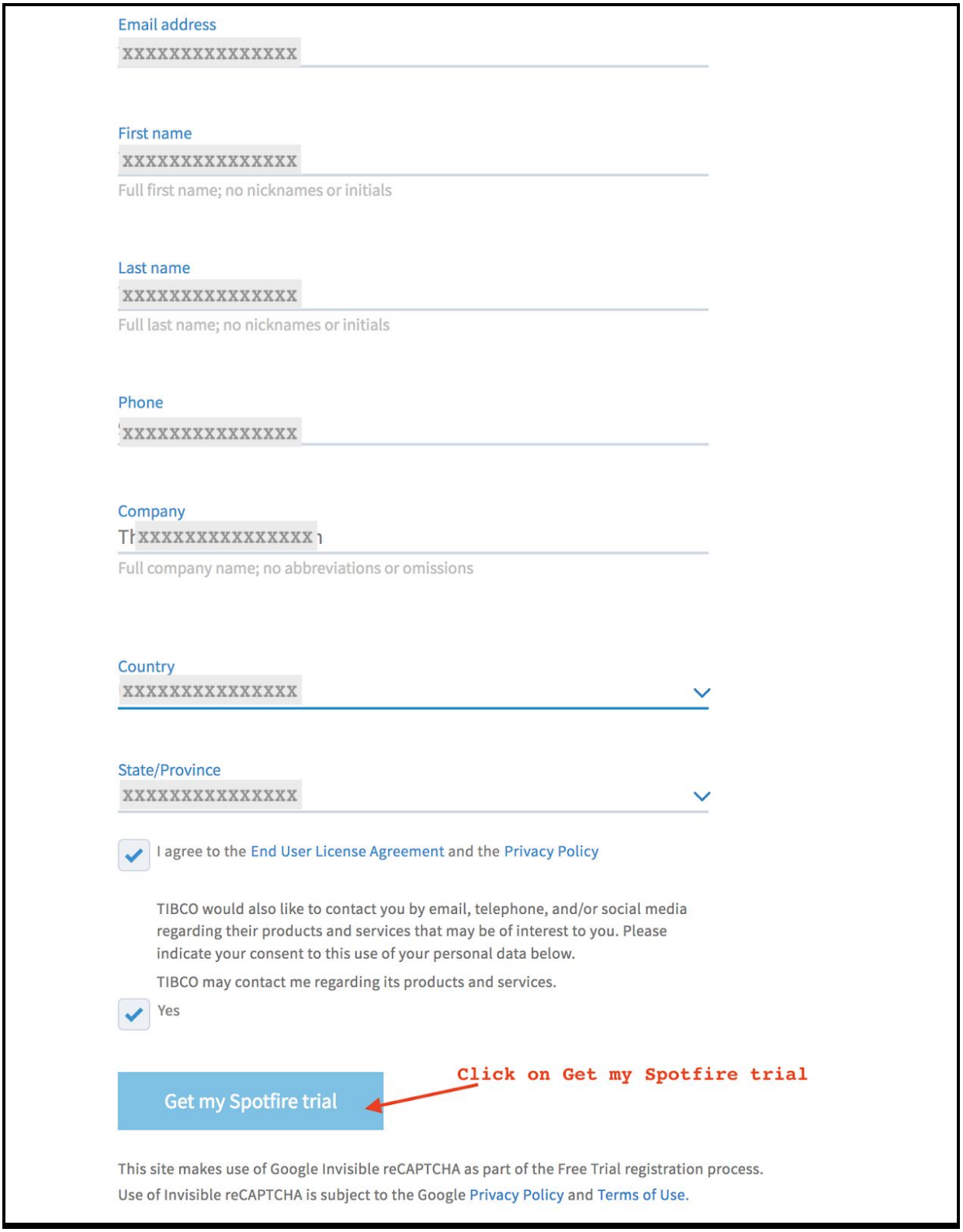


Next, scroll down until you find "TIBCO Cloud Spotfire"

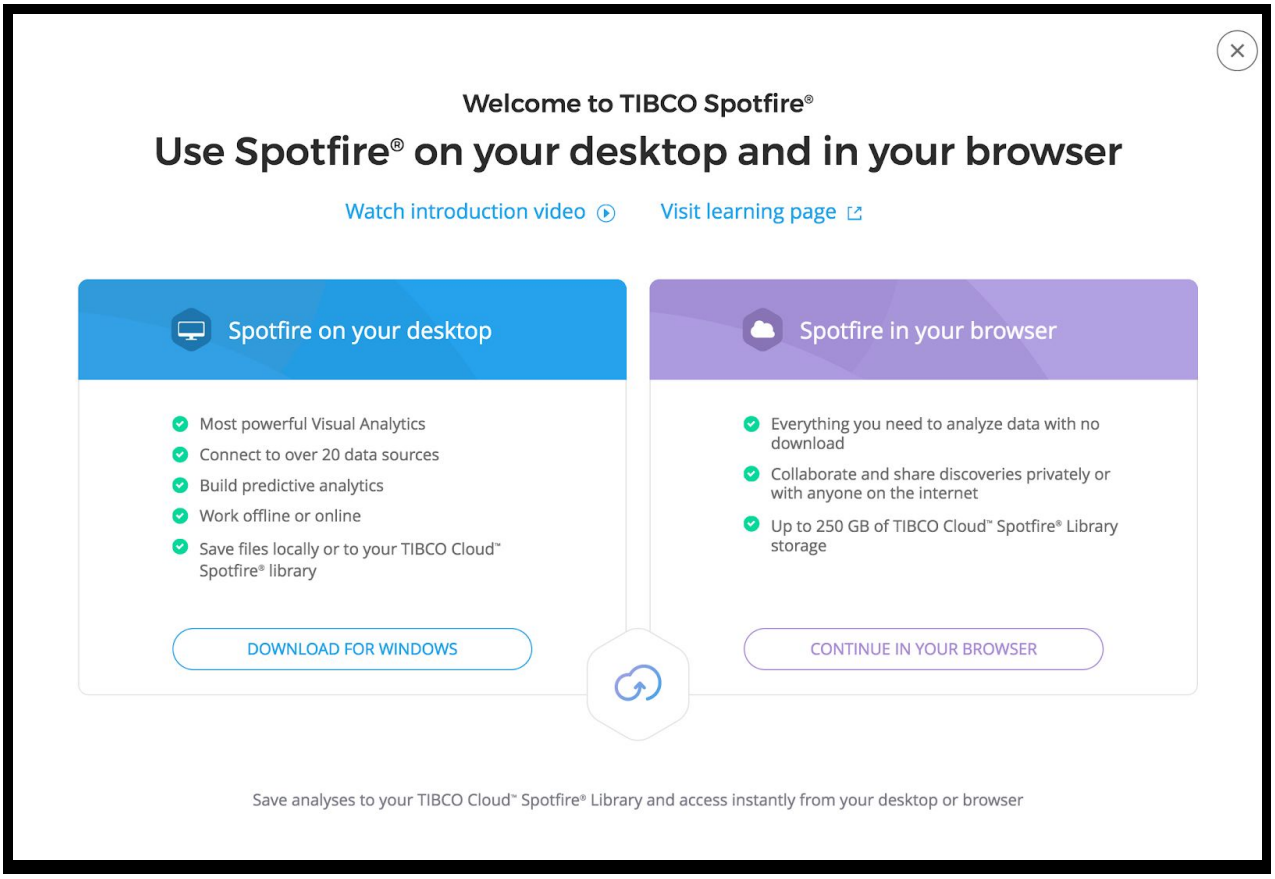
After you click on "Free Trial" you will see a signup screen to start your 30 day trial.



Provide the relevant details to start your Trial.



You will see a status screen and then presented with a choice to download spotfire locally on your windows machine or continue on the cloud. You can download anytime later too if you choose to but for now, let's continue to work on the browser.



From now on, it's pretty straight forward. You will see in the demo that we will show how you can click on "New Analysis", simply drop a data file and start using Spotfire to visualize your data.

Appendix D - Spotfire Tips and Tricks

- Getting Started with Spotfire Tutorials - https://spotfire.tibco.com/learn/videos?field_video_type_target_id=581
- WIKI - https://en.wikipedia.org/wiki/Notebook_interface
- TIBCO Community Exchange - https://community.tibco.com/exchange
- TIBCO Answers Page - https://community.tibco.com/answers
- Spotfire Reference Topics - http://spotfire.tibco.com/learn/quick-reference-topics
- Spotfire Reference Videos - https://www.youtube.com/channel/UCx3agqDZLbfrHNDUaxr0CXA
- Dr Spotfire - https://community.tibco.com/wiki/doctor-spotfire-office-hours
- TIBCO Tips and Tricks - https://community.tibco.com/wiki/tibco-spotfirer-tips-tricks-homepage
- Spotfire Demos - http://spotfire.tibco.com/demos
- Visual Design Best Practices - https://community.tibco.com/wiki/visual-design-best-practices-tibco-spotfirer
- TIBCO Spotfire Support - http://support.spotfire.com/
- See how to replace a value - https://docs.tibco.com/pub/spotfire_server/7.13.0/doc/html/en-US/TIB_sfire_bauthor-consumer_usersguide/GUID-A63A87CF-1923-4971-B338-B317F5AF51D6.html
- Two data sets - http://archive.ics.uci.edu/ml/datasets/Wine+Quality
- Data function - https://community.tibco.com/modules/clustering-variable-importance-data-function-tibco-spotfirer
- Extreme Gradient Boosting - https://community.tibco.com/modules/template-using-xgboost-tibco-spotfirer
- How to use data functions from community exchange - http://youtu.be/35EH8a5NFc8
- Regression - http://documentation.statsoft.com/STATISTICAHelp.aspx?path=MachineLearning/MachineLearning/SupportVectorMachine/SupportVectorMachineExample2Regression
- Classification - http://documentation.statsoft.com/STATISTICAHelp.aspx?path=MachineLearning/MachineLearning/SupportVectorMachine/SupportVectorMachineExample1Classification