

# Workshop 1: Raspberry Pi and IoT

# Objective:

The purpose of this workshop is to get you started working with Bluemix, Node-Red, Watson IoT, and a Raspberry Pi. You will be defining a Raspberry Pi as an IoT device in the Watson IoT service. Following that, you will use Node-Red to create an application on the Pi that reports sensor information (temperature, humidity, barometric pressure, etc.) to another Node-Red application running on Bluemix. Two sets of instructions are provided. If you have prior experience and would like to try and complete the exercise on your own, follow the Jedi Master path. Otherwise, if you prefer a more guided approach, choose the Jedi Padowan path. Choose wisely and have fun.

### Total time to complete: One Hour

1. Connect to Raspberry Pi and Start Node-Red (5 minutes)
2. Create the Bluemix Application Space (17 minutes)
3. Define IoT devices to Internet of Things Service (10 minutes)
4. Create dashDB table for environment data (8 minutes)
5. Build Raspberry Pi Node-Red flows
6. Build Bluemix Node-Red flows
7. Deploy and validate success

# Prerequisites:

This workshop assumes that you have completed the STSA event prerequisite activities. Specifically, you should have:

* An SSH terminal program for connecting to the Raspberry Pi. If you are using a MacOS or Linux based system, you are ready to go. If using Windows, you should have installed the PuTTY application available on the IBM Standard Software installer.

If you do not have these, please request assistance ASAP in order to get your system prepared as you will not have time to complete the exercise.

# Jedi Master Path:

### PART I: Start Node-Red on Raspberry Pi

* Connect to your Raspberry Pi using SSH. Your Pi can be reached at IP address 192.168.32.xx. Replace the xx with your specific team number. For example, team 30 would use:   
  **ssh pi@192.168.32.30**
* The Raspberry Pi credential are:

User ID: pi  
Password: raspberry

* Start Node-Red on the Pi with the command: node-red-start.

***FYI****: When the Node-Red app starts, the last action it performs is to start a logging function. You can exit this logging, if needed, by pressing Ctrl-C. This will not stop Node-Red itself. If/When you want to stop Node-Red, you need to issue the command: node-red-stop.*

### PART II: Create the Bluemix Application Space

* Using one of your team’s Bluemix accounts, create a new Node-Red application from the Node-Red boilerplate. Instructions in these workshops assume that the name of the application is **STSAWorkshops-xx** (where xx is your team number).
* Connect the following services to your new application. The names for the services are in parenthesis:
  + Internet of things (**STSAWorkshops-IoT**)
  + Watson Conversation (**STSAWorkshops-Conversation**)
  + dashDB for Analytics (**STSAWorkshops-dashDB**)

### Define IoT devices to Internet of Things Service

* In your newly created Internet of things service, device a new device type called **PiGateway**.
* Add a new gateway device using the newly defined PiGateway device type. For these workshop exercises, the device ID is assumed to be: **STSAGateway**.

***Note****: The workshops will be treating the Raspberry Pi as a Gateway and the attached Sense Hat as a downstream sensor device. However, it is not necessary to define the Sense Hat device to the IoT service as the gateway device will do it for you you when the Sense Hat connects through it.*

*.*

### Part IV: Create dashDB table for environment data

* Add a new table to your dashDB service using the following DDL:

CREATE TABLE "SENSEDATA"

(

"SENSORID" VARCHAR(20),

"TEMPERATURE" DOUBLE,

"HUMIDITY" DOUBLE,

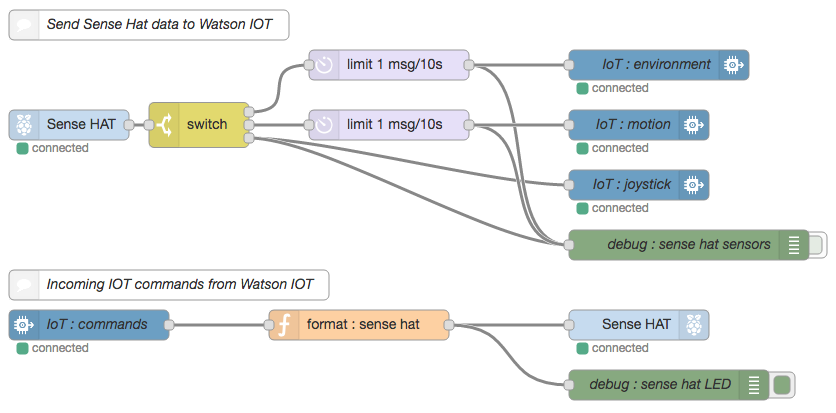
"PRESSURE" DOUBLE,

"TIMESENT" TIMESTAMP

);

***Note****: The name of this table and the names of the rows will be an important element of later workshops so be sure to double check your spelling.*

### Part V: Create Raspberry Pi Node-Red Flows

* Create the following:  
  
* Break the Sense Hat sensor data into three different event types (environment, motion, & joystick).
* Limit the number of environment and motion events that are sent to the IoT nodes to 1 every 10 seconds. Otherwise you will quickly overwhelm the data transfer limits imposed by our free IoT Service accounts.
* Send the data to the IoT service as one of three event types.
* Receive incoming IoT commands called **alarm** and **message**. The alarm command should light the entire 8x8 LED matrix on the Sense Hat to a solid color provided in the incoming IoT command. The message command should scroll a message across the LED matrix. The message, the text color, and the background color are all provided in the incoming IoT command.
* Format the incoming command data into the appropriate format for the Sense Hat node.  
  The incoming alarm command will have the following payload:

msg.command: "alarm"  
msg.format: "json"  
msg.deviceType: "SenseHat"  
msg.deviceId: "sensehat-xx”  
msg.payload: {d:{color:msg.payload}}

The incoming message command will have the following payload:

msg.command: "message"  
msg.format: "json"  
msg.deviceType: "SenseHat"  
msg.deviceId: "sensehat-xx”  
msg.payload: {d:{color:"blue",  
 background:"green",  
 message:”message text”}}

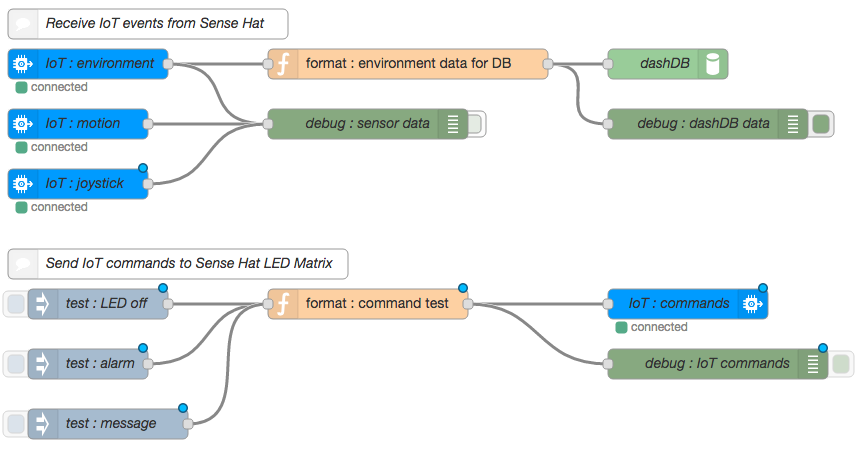
In order to set the entire 8x8 Sense Hat LED matrix to a specific color, you need to have the following string in the msg.payload:

msg.payload = “\*, \*, *color*” (replacing *color* with a color choice like red, blue, green, etc)

To have a message scroll across the LED matrix, the msg format is a bit more detailed:

msg.color = “*color*” (again, replacing *color* with a color choice like red, blue, green, etc)  
msg.background = “*color*” (again, replacing *color* with a color choice like red, blue, green, etc)  
msg.payload = “message to display”

### PART VI: Create Bluemix Node-Red Flows

* Create the following:  
  

### Part VII: Deploy and validate success

* Check your IoT service and validate the automatically created SenseHat device.
* Check your dashDB SENSEDATA table and confirm records are being stored.
* Inject an alarm command from Bluemix and confirm that the LED matrix on the Sense Hat lights to the desired color.
* Inject a message command from Bluemix and confirm that the LED matrix on the Sense Hat scrolls the message in the correct color and background.

# Jedi Padowan Path

### PART I: Start Node-Red on Raspberry Pi

Mac OS:

1. Start the Terminal application from the Mac OS Launchpad.
2. Issue the command: **ssh pi@ 192.168.32.xx**. Replace the xx with your team specific team number. For example, team 30 would use: **ssh pi@192.168.32.30  
   *Note:*** *When you connect for the first time, you may see a message indicating that the authenticity of the host could not be established. Simply answer with “yes”.*

Windows:

1. Start the PuTTY application from the Windows start menu.
2. Enter **pi@ 192.168.32.xx**. (Replace the xx with your team specific team number. For example, team 30 would use: **ssh pi@192.168.32.30**)
3. Click “Open”.  
   ***Note:*** *When you connect for the first time, you may see a message indicating that the authenticity of the host could not be established. Simply answer with “yes”.*

Everyone:

1. At this point, you will be prompted for the user password. The Raspberry Pi credential are:

User ID: **pi**  
Password: **raspberry**

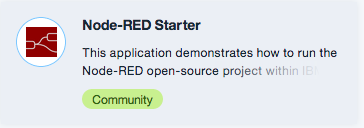
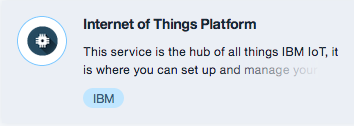
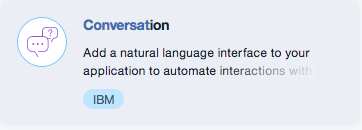
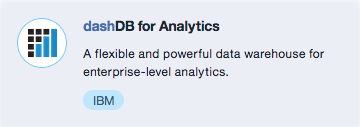
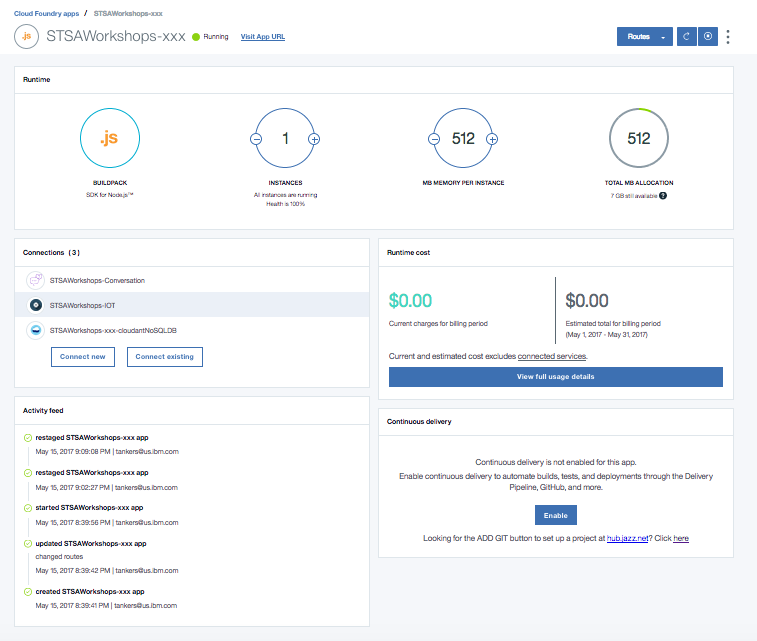
***Note:*** *Once logged in, you will see a message reminding you to change the password of your Pi. Use the* ***passwd*** *command to change it in order to ensure the security of your work. Be sure to make a note of your new password.*

1. Start Node-Red on the Pi: with the command: **node-red-start**.

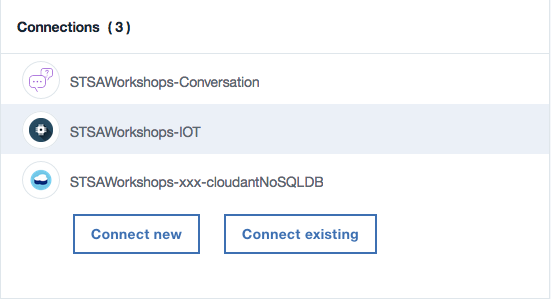
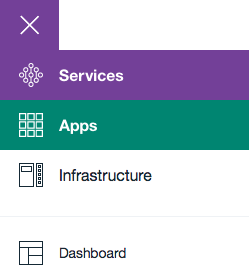
***FYI****: When the Node-Red app starts on the Pi, the last action it performs is to start a logging function. You can exit this logging, if needed, by pressing Ctrl-C. This will not stop Node-Red itself. If/When you want to stop Node-Red, you need to issue the command: node-red-stop.*

### PART II: Create the Bluemix Application Space

Here we will create the Bluemix side of the application. This will be a Node-Red application that will receive sensor events from, and send commands to a Raspberry Pi. We will start with a Node-Red boilerplate application and connect the IoT, Watson Conversation, and dashDB services to it.

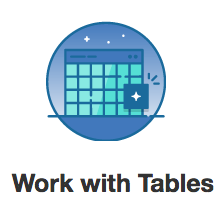
1. Open a browser and sign in to one of your team’s Bluemix accounts at [**www.bluemix.net**](http://www.bluemix.net). You will be using only one of your Bluemix accounts for this workshop.
2. From here, click on:  and navigate to and click on: .   
   You can use the type-ahead feature of the search bar to quickly find this boilerplate.
3. On the resulting screen, you need to give your application a name. You can name you application anything you like provided it is unique. For consistency, these workshops will use the name **STSAWorkshops-xx** (where xx is your team number). Enter **STSAWorkshops-xx** in the application name field, and click ****. There is no need to modify any of the other fields on this page. At this point, your application will be created and, after a few moments, you will be taken to the Getting Started page for your application.
4. As you work through the workshops over the new 2 days, you will need connections to additional services in your application. Let’s make those connections now. Go to the **Connections menu** of your application page. Note that as part of the boilerplate, a Cloudant NoSQL database has been created automatically. You will add three additional services: Internet of Things, Watson Conversation, and dashDB.
5. Click  and select the .   
   Again, you can use the type-ahead feature of the search bar to find this service quickly.  
   **Important**: Make sure that you select the IoT Platform Service, not the IoT Platform Boilerplate app.
6. You will need to give your IoT service a name. A default name is suggested but may be typed over. While it can name the service anything you like, the instructions throughout will assume a name of **STSAWorkshops-IoT** for the IoT service.
7. Click **** and, after a moment, you will be asked to restage the app. Say **yes** to the restaging request. At this point you have created a new IoT service and connected it you your application.
8. Repeat previous steps to add the **STSAWorkshops-Conversation** and **STSAWorkshops-dashDB** services to your application using the following service options.  
       
   ***Note****: After creating the dashDB service, you will not be asked to restage the application, and will be redirected to your Bluemix main dashboard.*
9. Scroll down to the application section of the dashboard and click on the **name** of your new application. Be sure to click on the name and not the route for your application. This will re-open the application to the Overview page which should look similar to this:  
   

### PART III: Define IoT devices to Internet of Things Service

1. At this point you should be in the “Overview” section of your Bluemix application. If you are not, simply click on your application name (**STSAWorkshops-xx**) in the Bluemix dashboard.
2. Scroll down to the:  
      
   And click on the **STSAWorkshops-IoT** service to start it.
3. This will land you on the “Manage” page of your IoT service where you should click . ***Note****: The IoT service should then launch in a separate browser window or tab leaving the service details tab open as well. Keep this tab open as it will make returning to the Bluemix dashboard much easier.*
4. You will need to create a new IoT device. On the Left side of the window, you should see an icon that looks like this:. Click on this icon to get to the devices view.   
   ***Note****: These workshops will be treating the Raspberry Pi as a Gateway and the attached Sense Hat as a downstream sensor device. However, it is not necessary to define the Sense Hat device to the IoT service as the gateway device will do it for you when the Sense Hat connects through it.*
5. Click new on the .
6. The first step of device creation is to select the device type. There is a drop-down menu to select from but you will notice that there are no device types in the list. So, we will create the device type as a part of the device creation by selecting .
7. We will be configuring the Raspberry Pi as a Gateway device select .
8. Call the new device type **PiGateway** and give it any description you like.
9. Then, in the lower right corner, find and select the  button.
10. At this point, you arrive at the “Define Template” page. The options on this page select attributes for the device type. All of these attributes are optional. They will be used as a template for new devices that are assigned this device type. Attributes you do not define may still be edited individually on devices that are assigned this device type. We will not define any additional attributes, so just click .
11. Next, you come to the “Submit Information” page. If you had selected any attributes in the prior page, they would be listed here for verification. Select  again to proceed to the “Metadata” page.
12. The “Metadata” page is where you would define any custom attributes that might be needed for your environment. We have no need for any Metadata so just click  to create the new PiGateway device type.
13. Now you will be returned to the “Add Device” page. However, now you will see the new device type “PiGateway” in the list of device types. Go ahead and select it and click .
14. Every device needs to have a unique id within your specific IoT service instance and it is defined on this Device Info page. You can use any unique name that you like, these workshops assume the device id is: **STSAGateway**. Enter your device id and click .  
    ***Note****: There are also several other attributes that can be defined here that can help you to identify a device and its purpose. These can be very useful in a multi device environment but, since we will only be dealing with a gateway and a downstream sensor, these additional attributes won’t be needed.*
15. Just as ii was during Device Type creation, the “Metadata” page is where you would define any custom attributes that might be needed for your environment. We have no need for any metadata so just click  to move to the “Security” page.
16. Regarding security, there are two options. The Auto-generated token is a random 18 character mix of alphanumeric characters and symbols. The other option is a Self-provided token where you provide your own authentication token for the device. While in the real world, security is paramount, for these workshops, please scroll down and enter an easily remembered, **self-provided** token (we will use **raspberry**) and then click .  
    ***Note****: Authentication tokens are encrypted and cannot be recovered if lost. Be sure to make a note of your token.*
17. Finally, you come to the “Summary” page where you can verify all of your entries before you complete the device creation by clicking .
18. Your device is now defined and you are presented a summary page of the result. Make a note of the following items from the summary page as you will need them later when you actually connect your device to the IoT service.  
      
    Organization ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
      
    Device Type: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
      
    Device ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
      
    Token: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
19. Close the summary window by clicking  and you will be taken back to the device dashboard where you will see your new device. You can leave this browser tab open for reference later, but, for now, switch over to the IoT Service Details browser tab so you can return to your Bluemix dashboard.
20. In order to return to the Bluemix Dashboard, click the hamburger menu in the upper left of the page, select Services then Dashboard.  
      
    

### Part IV: Create dashDB table for environment data

At this point, we will create a table in your dashDB service to hold the environment data that is sent to your application from the IoT sensors. This will be a fairly simple table that will store the Temperature, Humidity, and Barometric Pressure values into a row of the table each time an environment event is received. We will also store the ID of the sensor that sent the data along with a time stamp.

1. To get started, find the **STSAWorkshops-dashDB** Service in your Bluemix dashboard and click on its name to launch the service.
2. Like the IoT Service, this will land you on the “Manage” page of your dashDB service where you should click .   
   ***Note****: The dashDB service should then launch in a separate browser window or tab leaving the service details tab open as well. Keep this tab open as if will make returning to the Bluemix dashboard much easier.*
3. Click on and then in order to create a new table.
4. Replace the text in the “Edit the DDL statements box with the following:

**CREATE TABLE "SENSEDATA"**

**(**

**"SENSORID" VARCHAR(20),**

**"TEMPERATURE" DOUBLE,**

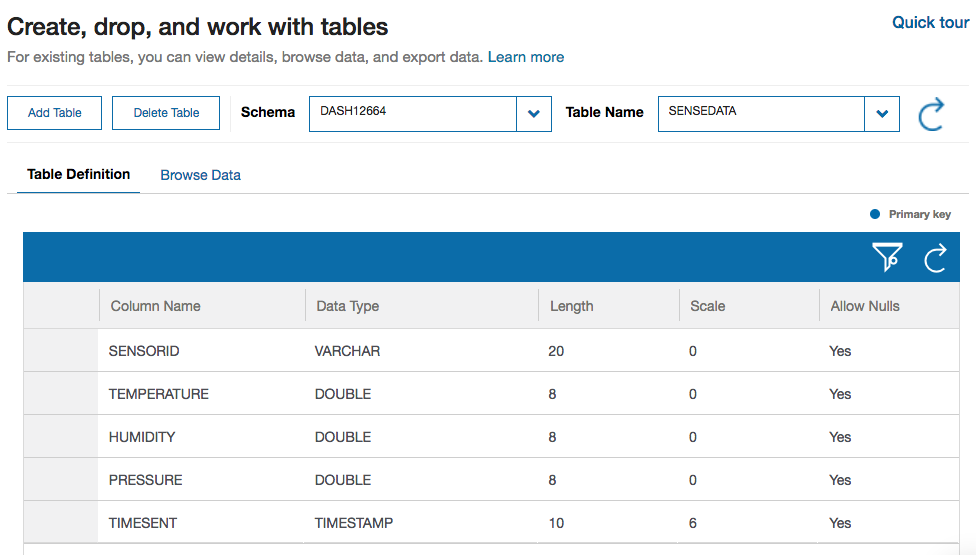
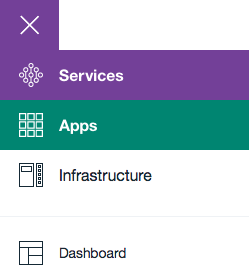
**"HUMIDITY" DOUBLE,**

**"PRESSURE" DOUBLE,**

**"TIMESENT" TIMESTAMP**

**);**

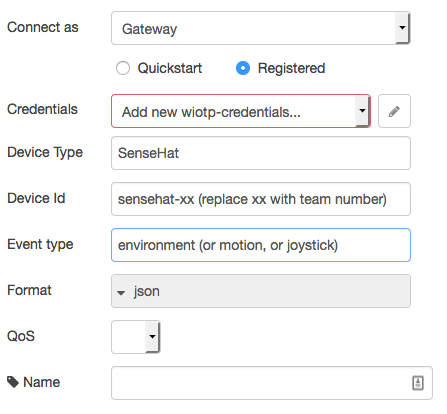
***Note****: The name of this table and the names of the rows will be an important element of later workshops so be sure to double check your spelling.*

1. Click and your table will be created. You should see a message that the table was created successfully. Dismiss that message and you will be returned to the “Create a Table” page. Dismiss that as well and you will be back to the “Create, drop, and work with tables” page.
2. Select the “SENSEDATA” table from the table name drop downand you will be able to see the table definition. You can also click on “Browse Data” in order to see the actual data stored in the table. This will be handy later.  
   
3. You can leave this browser tab open for reference later, but, for now, switch over to the dashDB Service Details browser tab so you can return to your Bluemix dashboard.
4. In order to return to the Bluemix Dashboard, click the hamburger menu in the upper left of the page, select Apps then Dashboard.  
     
   

### Part V: Create Raspberry Pi Node-Red Flows

In this portion of the workshop you will create a Node-Red application on the Raspberry Pi that will collect sensor data from a device called a Sense Hat that is attached to the Pi. You will then forward that data to your IoT Service so that it can be used by a corresponding Node-Red application you will create in Bluemix. There are three different types (environment, motion, & joystick) of sensor data and each type will be sent to the IoT service with a specific event type so that different actions might be taken depending on the event type. Additionally, this application will be able to receive commands sent from the Bluemix application that will control the 8x8 LED matrix that is part of the Sense Hat device. One command (alarm) will turn the entire matrix into a solid color that is provided as a part of the message payload. The other command (message) will scroll a text message across the matrix. The message, the text color, and the background color will all be provided as a part of the message payload. Ready? Let’s begin.

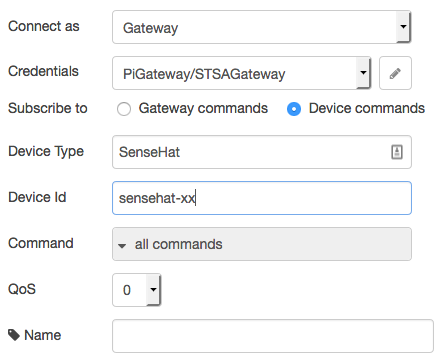
#### **Flow 1**

1. Node-Red programming is done via web browser. In order to get started, fire up one of your team’s laptop browsers and enter **192.168.32.xx:1880** (Replace the xx with your team specific team number. For example, team 30 would use: **192.168.32.30:1880**) into the address field.  
   ***Note****: When you started Node-Red on the Pi, you may have noticed that is said to go to 127.0.0.1:1880. This would work if you were running the browser on the Pi itself (There is actually a full GUI environment available on the Pi and you can run a browser locally). However, because you are connecting to the Pi remotely, you need to use the network address of your team’s system.*
2. From the node palette on the left, find the input node (it should be located in the Raspberry\_Pi section of the palette) and drag it out into your Node-Red workspace.
3. Double click on the new Sense Hat node to open its settings and ensure that all three event types are being reported by checking each box.
4. Next, find and add a  node (In the output section) to the right of the Sense Hat node.
5. Open the debug node and change the output to **complete msg object**. This will allow you to view the entire msg being received by the prior node. It can be very useful in helping to format the msg that needs to be sent to the next node.
6. Connect your Sense Hat node to your debug node by clicking and dragging from one connection point to the other .
7. At this point, verify that your Sense Hat is producing output by clicking .
8. On the right side of the page you should see a tab labeled “debug”. Click on that tab and you should see a tremendous amount of data flowing into your debug node from the Sense Hat.
9. On the very right end of the debug node you will see a green toggle button . This allows you to stop/start the output to that particular debug node. Turn off the output by clicking the toggle and the data will stop scrolling by in the debug panel. Now you can get a closer look at the actual messages that the Sense Hat is sending.
10. Take a minute to examine the debug output. You will see that each Sense Hat message has a topic and a payload. Notice that the topic will match one of the three data types that the Sense Hat reports (environment, motion, & joystick). In order to perform different actions on the different sensor topics, we will need to separate them into unique events. The first step of this process is to find the **switch** node (function section) and add it to your workspace. The switch node routes the application flow to different paths based upon the value of some element of the incoming payload.
11. Open the settings for the switch node and set the property value to **msg.topic**. This is will cause a branch in the application based upon the contents of the topic element of the incoming message.
12. We will want to route the flow based upon whether the topic is equal (==) to “environment”, “motion”, or “joystick”. You may notice that there is only one place to enter selection criteria and you need three. You will need to use the  button to enable the additional fields to accommodate the three topics.
13. Notice that there are now three separate connection points for the switch node. This is what allows you to route the flow based on the contents of the msg.topic. The connection points are in the same order as they were added to the switch node.
14. One thing that you may have noticed when viewing the debug output is that there are a lot of events being generated by both the environment and the motion topics. If you were to send every one of these events to the IoT service, you would quickly use up the 200MB limit that is imposed on a free account. This is also a consideration for your clients as the IoT service does have a charge that is based upon data transmission. To deal with this, connect two **delay** nodes to the back end of the switch nodes. One for the environment topic and one for the motion topic.
15. For each delay node, open the settings and set the action to **Limit rate to.**
16. Limit the rate to 1 message every 10 seconds and check the box to **drop intermediate messages**. This will cause the application to discard all messages except for one every 10 seconds.
17. Delete the connection between the Sense Hat node and the debug node by selecting the line connecting them and hitting the delete key.
18. Now, connect the delay nodes and the final switch connection (joystick topic) to the debug node. Note that you can have several nodes connecting to a single connection point.
19. Redeploy the application by once again clicking deploy and you should see a dramatic decrease in the number of messages received. Also, notice that you will only see joystick topics when you use the Sense Hat joystick. For that reason, there is no need to delay them like the other topics.
20. Finally, it is time to send these messages to your IoT Service so that they can be accessed from your Bluemix application. For this, you will need to pull three **Watson IoT** output nodes into your workspace and position them to the right of your delay nodes.
21. Each Watson IoT node will send a separate event type to the IoT service. You will need to configure them by opening settings and configuring as follows:

* Because we defined the Raspberry Pi as a Gateway device, you need to connect as a **Gateway**.
* When configuring the first of these nodes, you will need to **Add new wiotp-credentials.** You do this by clicking on the  in the credentials line. Here you will identify the gateway device that you defined earlier. Use the values that you recorded during the IoT device definition to fill in the appropriate fields. Do not modify any other fields in this section.
* The device type and Device ID can be any value that helps to identify their purpose. We will use the values provided for consistency.
* The value entered into Event type will define the actual event that this message comprises. Use one of the three values in each node.

1. Connect the Watson IoT nodes to their respective delay or switch nodes.
2. This concludes flow 1.

#### **Flow 2**

1. The second flow will receive commands that will control the Sense Hat LED. In order to build the flow, you will need just four nodes. A **Watson IoT** input node will connect to a **function** node, which in turn will connect to both a **Sense Hat** output node, and a **debug** node. In some space below your first flow, drag these nodes into your workspace and connect them as described
2. Like before, set the debug node to show the **complete msg object**.
3. The Watson IoT input node will receive the commands that are sent from the Bluemix application. You will need to configure the node by opening settings and configuring as follows:  
   

* Again, because we defined the Raspberry Pi as a Gateway device, you need to connect as a **Gateway**.
* However, this node is listening for commands that are destined for a downstream sensor, not the gateway itself. So, you need to subscribe to **Device commands**.
* Unlike the prior flow, here you will catch **all commands** with a single Watson IoT node. In this case, you will still take different action based on the command type, but will determine the command type by code. You could do this by using two separate IoT nodes (one for each command) like before, but this way you will see that there are many different ways to accomplish the same task with Node-Red.

1. The function node is a powerful node that allows you to incorporate your own program logic into a Node-Red application. The message is passed in to the function node as a JavaScript object called msg. By convention it will have a msg.payload property containing the body of the message. You can act on the content of this payload, modify it, and pass it along to the next node in your application. The function node in this flow will receive the message from the IoT service and put it into the format that the Sense Hat expects. The commands that come into the Watson IoT node have the following format:

The “alarm” command:

msg.command: "alarm"  
msg.format: "json"  
msg.deviceType: "SenseHat"  
msg.deviceId: "sensehat-xx”  
msg.payload: {d:{color:msg.payload}}

The “message” command

msg.command: "message"  
msg.format: "json"  
msg.deviceType: "SenseHat"  
msg.deviceId: "sensehat-xx”  
msg.payload: {d:{color:"blue",  
 background:"green",  
 message:”message text”}}

In order to set the entire 8x8 Sense Hat LED matrix to a specific color, you need to have the following string as the msg.payload:

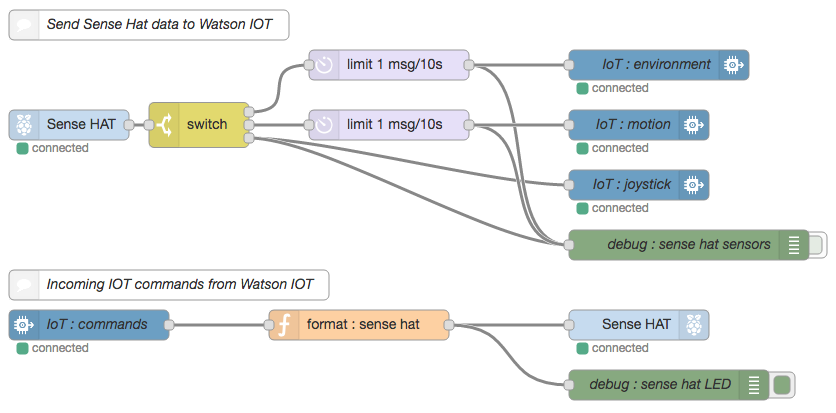
msg.payload = “\*, \*, *color*” (replacing *color* with a color choice like red, blue, green, etc)

To have a message scroll across the LED matrix, the msg format is a bit more detailed:

msg.color = “*color*” (again, replacing *color* with a color choice like red, blue, green, etc)  
msg.background = “*color*” (again, replacing *color* with a color choice like red, blue, green, etc)  
msg.payload = “message to display” ***Note:*** *If the message string is only a single character, it will not scroll. Rather, it will stay lit on the LED.*

With the knowledge of the contents of the incoming command message as well as the requirements of the Sense Hat, a trivial snippet of JavaScript code can check for the command type and make the transformation. Enter the following code into the function field of the function node.

// begin code  
  
d = msg.payload.d;  
  
if (msg.command == "message") {  
 msg.background = d.background;  
 msg.color = d.color;  
 msg.payload = d.message;  
}  
else if (msg.command == "alarm") {  
 msg.payload = "\*,\*," + d.color;  
}  
else  
 msg = null;  
  
return msg;  
  
// end code

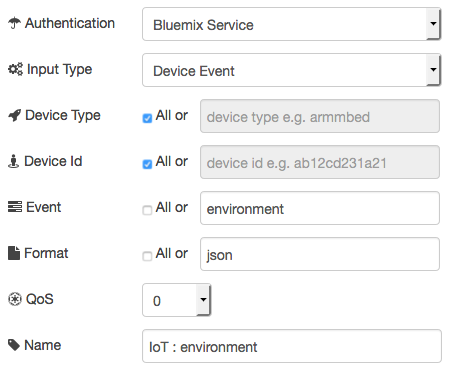
1. With that, the Raspberry Pi side of this application is complete. Click on **Deploy** to start the application. Once the Bluemix side is complete, you will be able to test the flows and ensure that everything is working as expected. Part VII of this workshop has some verification tests you can use.
2. In the end, your Raspberry Pi application should look something like this:  
   

### Part VI: Create Bluemix Node-Red Flows

In this portion of the workshop you will create a Node-Red application on Bluemix that will receive sensor data sent to it from a corresponding application on the Raspberry Pi. You will then store some of that data in your dashDB Service so that it can be used in later workshops. There are three different types (environment, motion, & joystick) of sensor data that will be received. You will only store the environment data in the database. Additionally, this application will be able to send commands to the Raspberry Pi application that will control the 8x8 LED matrix that is part of the Sense Hat device. One command (alarm) will turn the entire matrix into a solid color that is provided as a part of the message payload. The other command (message) will scroll a text message across the matrix. The message, the text color, and the background color will all be provided as a part of the message payload. Ready? Let’s begin.

#### Flow 1

1. Node-Red programming is done via web browser. In order to get started, return to your Bluemix Dashboard browser tab*.*
2. From there, you get to the Node-Red programming space, by clicking on your application’s **Route** (not its name as you have done before).It will look like this **STSAWorkshops-xx.mybluemix.net.**
3. 
4. From the node palette on the left, find the input node (it should be located in the input section of the palette) and drag it out into your Node-Red workspace.
5. The ibmiot input node will receive the events that are sent from the Raspberry Pi application. You will need to configure the node by opening settings and configuring as follows:



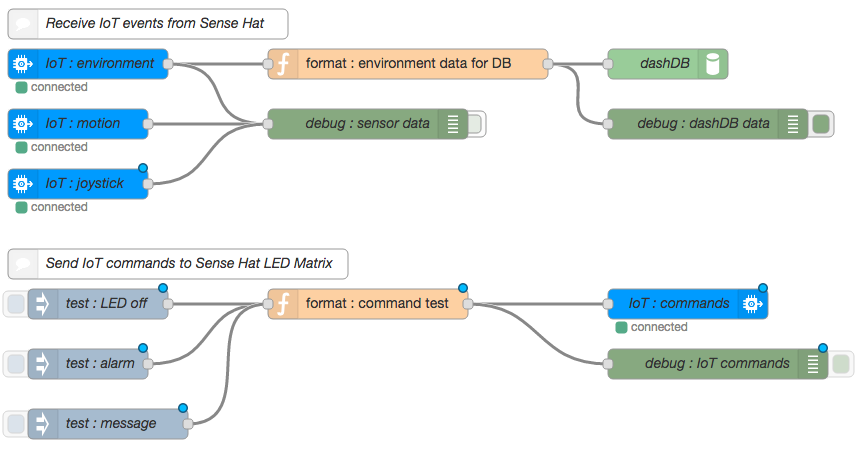
* Because the IoT service is connected to your application, all the authentication can be handled right through Bluemix and no other information needs to be provided for authentication.
* Set this node to accept events from all Device Types and Device Ids. Note however, that you can get restrictive and only accept events from specific IDs and types if desired.
* This node should only accept events of type **environment**.

1. Now, add two more ibmiot nodes below the current one that will handle the **motion** and **joystick** events respectively.
2. Next, find and add a  node (In the output section) to the right of the ibmiot nodes.
3. Open the debug node settings and change the output to **complete msg object**. This will allow you to view the entire msg being received by the prior node. It can be very useful in helping to format the msg that needs to be sent to the next node.
4. Connect your ibmiot nodes to your debug node by clicking and dragging from one connection point to the other . Note that you can have several nodes connecting to a single connection point.
5. Once the Raspberry Pi side of this application is complete, you will be able to verify that you are receiving events from the Raspberry Pi by clicking .
6. On the right side of the page you should see a tab labeled “debug”. Click on that tab and you should see data flowing into your debug node from the ibmiot nodes.
7. On the very right end of the debug node you will see a green toggle button . This allows you to stop/start the output to that particular debug node. Turn off the output by clicking the toggle and the data will stop coming into the debug panel. Now you can get a closer look at the actual event messages that are being received as they won’t continually scroll by.
8. Take a minute to examine the debug output. You will see that each message has a payload. Notice that each message will have an event type that matches one of the three events that the Raspberry Pi forwards (environment, motion, & joystick).
9. Now you will store the incoming environment event data into your dashDB table. First you will need to put the incoming data into the correct format for the database. You will do that with a function node. The function node is a powerful node that allows you to incorporate your own program logic into a Node-Red application. The message is passed in to the function node as a JavaScript object called msg. By convention it will have a msg.payload property containing the body of the message. You can act on the content of this payload, modify it, and pass it along to the next node in your application. The function node in this flow will receive the message from the IoT service and put it into the format that the dashDB expects. To get started, find the **function** node (function section) and add it to your workspace.
10. The msg.payload for the dashDB node should include a value for each column in your table. To make this happen, open the function node and enter the following code into the function field of the function node to transform the incoming Sense Hat data into that which is expected by dashDB.

// Format Sensor Data for dashDB  
msg.payload = {  
 SENSORID : msg.deviceId,  
 TEMPERATURE : msg.payload.d.temperature,  
 HUMIDITY : msg.payload.d.humidity,  
 PRESSURE : msg.payload.d.pressure,  
 TIMESENT : 'TIMESTAMP'  
};  
return msg;

1. Connect the function node to your **ibmiot** **environment** node.
2. Drag another debug node into the workspace and connect it to the other side of the function node.
3. Deploy the app and verify that the output from your function node appears correct by examining the payload output in the debug tab.
4. Once the data is formatted properly, its time to store it into your database. Find a **dashDB** node (storage) and place it to the right of the function node.
5. Edit the dashDB node by selecting your database service from the drop-down list and specifying the table name of **SENSEDATA**.
6. Deploy the application and your environment data will now be stored into the SENSEDATA table. Each row of the table will correspond to an incoming environment event.
7. This concludes flow 1.

#### Flow 2

1. The second flow will inject test data that will be sent to the Sense Hat device to control the LED. In order to build the second flow, you will need six nodes. Three **inject** nodes will connect to a **function** node, which in turn will connect to both a **ibmiot** output node, and a **debug** node. In some space below your first flow, drag these nodes into your workspace and connect them as described.
2. Like before, set the debug node to show the **complete msg object**.
3. The three inject nodes will be used to send
4. With that, the Bluemix side of this application is complete. Click on **Deploy** to start the application. Once the Raspberry Pi side is complete, you will be able to test the flows and ensure that everything is working as expected. Part VII of this workshop has some verification tests you can use.
5. Create 3 test nodes that demo the commands to be sent
6. Send two commands to the Sense Hat device (alarm and message)
7. Use inject nodes to test the proper formatting of the data sent to the Sense Hat device
8. In the end, you should have something that looks a bit like this:  
   

### Part VII: Deploy and validate success

1. Check your IoT service and validate the automatically created SenseHat device.
2. Check your dashDB SENSEDATA table and confirm records are being stored.
3. Inject an alarm command test from Bluemix and confirm that the LED matrix on the Sense Hat lights to the desired color.
4. Inject a message command test from Bluemix and confirm that the LED matrix on the Sense Hat scrolls the message in the correct color and background.