



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 Padawan path. Choose wisely and have fun.

Total time to complete: One Hour

- I. Create the Bluemix Application Space (17 minutes)
- II. Define IoT devices to Internet of Things Service (10 minutes)
- III. Create dashDB table for environment data (8 minutes)
- IV. Build Bluemix Node-Red flows
- V. Connect to Raspberry Pi and Start Node-Red (5 minutes)
- VI. Build Raspberry Pi Node-Red flows
- VII. 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: 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**)
 - dashDB for Analytics (**STSAWorkshops-dashDB**)

Part II: 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 when the Sense Hat connects through it.

Part III: Create dashDB table for environment data

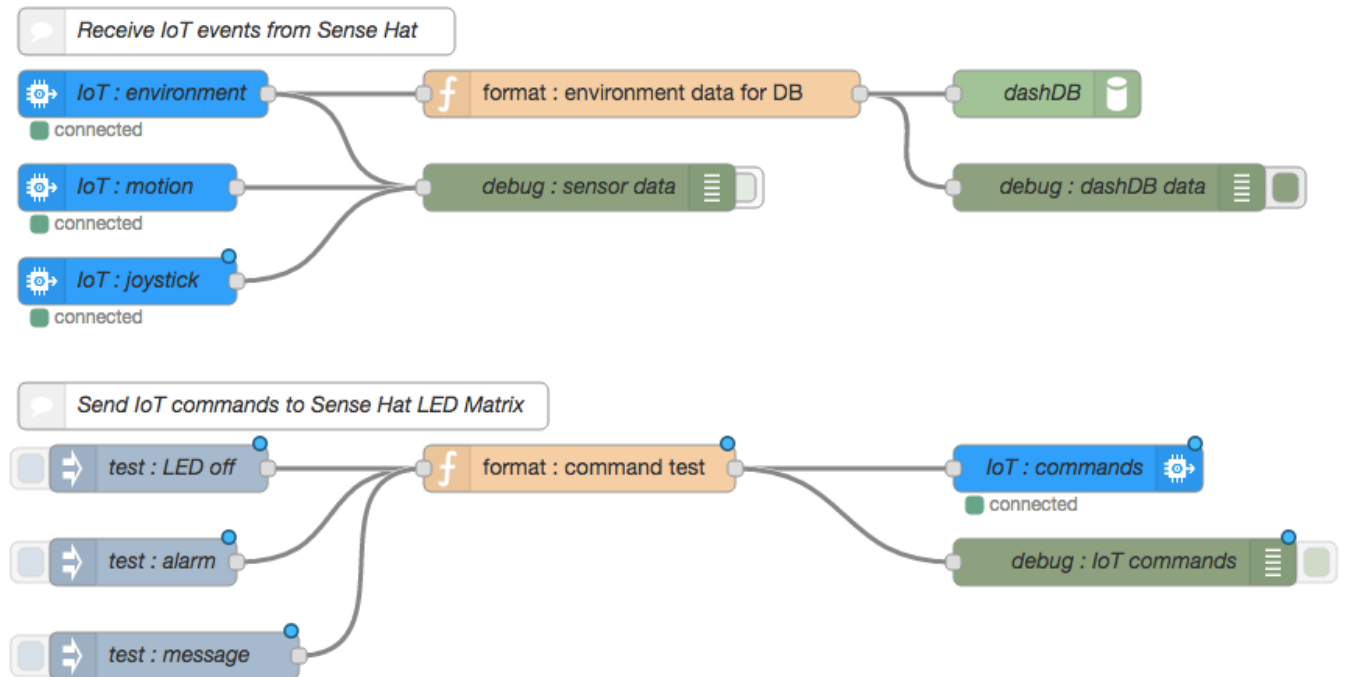
- Add a new table to your dashDB service using the following DDL:

```
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 IV: Create Bluemix Node-Red Flows

- Create the following:



- Receive the three different event types (environment, motion, & joystick).
- Format the incoming **environment** data into the appropriate format for the dashDB node.

The incoming environment event will have the following payload:

```
msg.payload: {d:{temperature: 35.21
                  humidity: 38.31
                  pressure: 994.84}}
```

The dashDB node will need the following payload based upon the table created earlier:

```
msg.payload: {SENSORID : deviceId,
              TEMPERATURE : temperature,
              HUMIDITY : humidity,
              PRESSURE : pressure,
              TIMESENT : 'TIMESTAMP'}
```

- Send test commands called **alarm** and **message** to the Sense Hat device on the Raspberry Pi. The three inject nodes should send a string payload with a topic that identifies the specific command being sent as follows:

Payload	Topic	Name
String: off	alarm	Turn off LED
String: color (red, blue, etc)	alarm	Send Alarm
String: any text string	message	Send Message

- Set the outbound msg.eventOrCommandType to either alarm or message based upon the incoming topic type.

- Format the injected data into the appropriate format for the Raspberry Pi application based upon the incoming topic type:

The alarm command will need to have the following payload:

```
msg.payload: {d:{color:"desired color or off"}}
```

The message command will need to have the following payload:

```
msg.payload: {d:{color:"desired color",
                  background:"desired color",
                  message:"desired message"}}
```

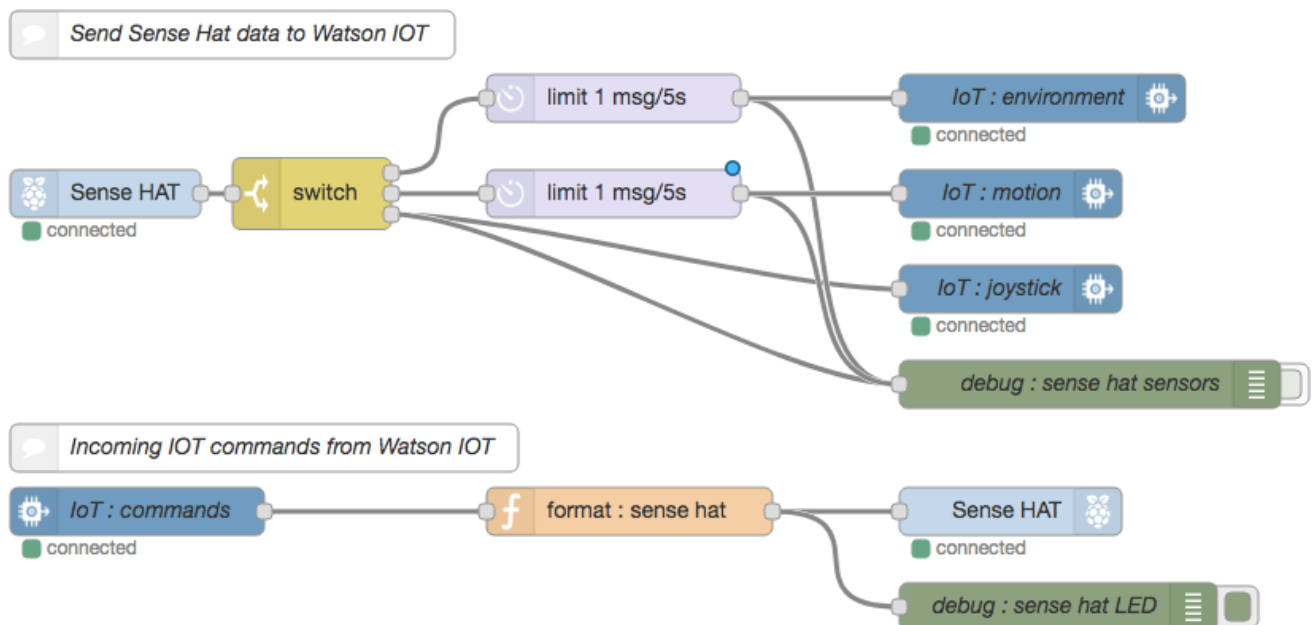
PART V: Start Node-Red on Raspberry Pi

- 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 VI: 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 5 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 VII: Deploy and validate success


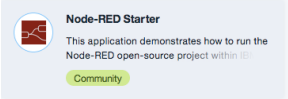

- Please skip ahead to the common Part VII Section.


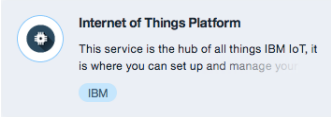
Jedi Padawan Path

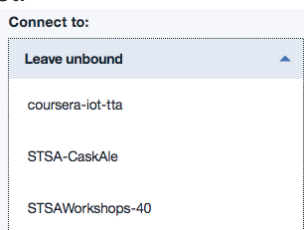
PART I: Create the Bluemix Application Space


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**. You will be using only one of your Bluemix accounts for this workshop.

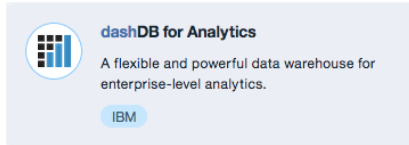
2. From here, click on:  and navigate to and click on: . You can use the type-ahead feature of the search bar to quickly find this boilerplate.
3. On the resulting screen, you need to give your application a name. You can name your application anything you like provided it is unique. For consistency, these workshops will use the name **STSAWorkshops-xx (Replace xx with your team number)**. Enter your application name 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:** As part of the boilerplate, a Cloudant NoSQL database has been created automatically. You will add two additional services: **Internet of Things and dashDB for Analytics**.

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. Next, you need to tell this new service what application to connect to. On the left side of the page, you will see the connect to drop-down menu. Be sure to select your Bluemix application from this list.

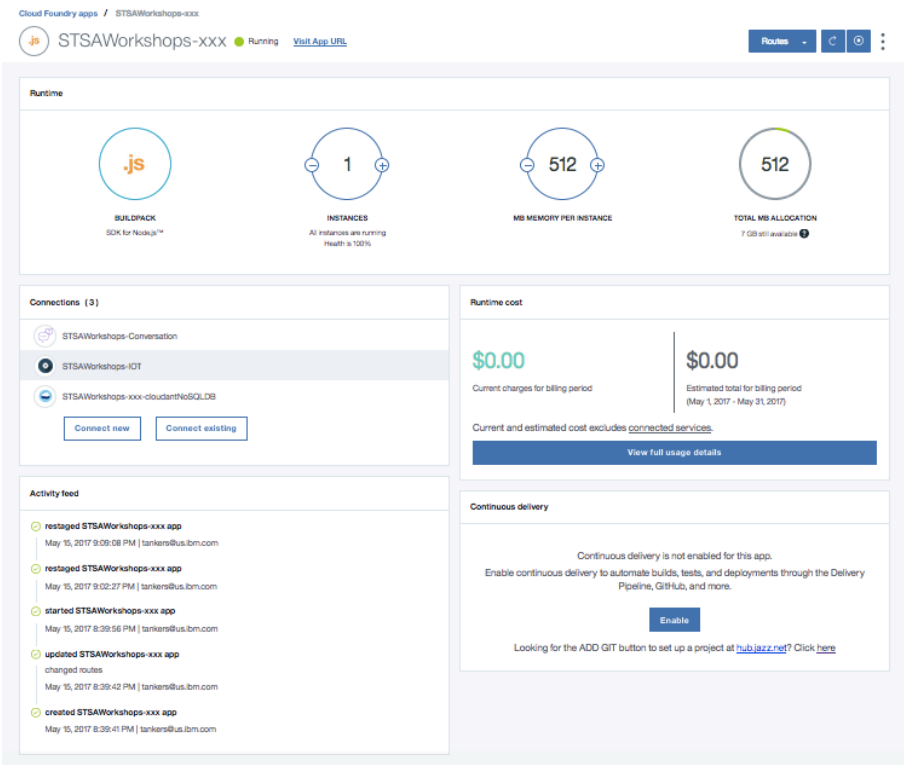


8. Click  and, after a moment, you will be asked to restage the application. Say **yes** to the restaging request. At this point you have created a new IoT service and connected it to your application.

- Repeat steps 4 - 8 to add the **STSAWorkshops-dashDB** services to your application using the following service.

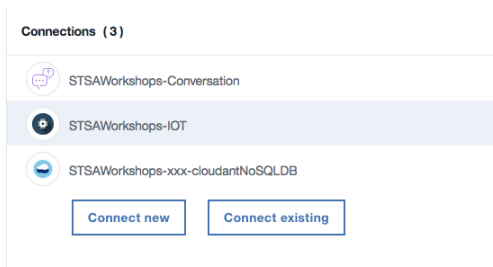


- At this point, you should be in the Connections section of your application. Click on Overview in order to return to the Overview page which should look similar to this:

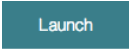


PART II: Define IoT devices to Internet of Things Service


- 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.
- Scroll down to the:













And click on the **STSAWorkshops-IOT** service to start it.

- 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 it 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 **Next**.

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 **Add**.
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.

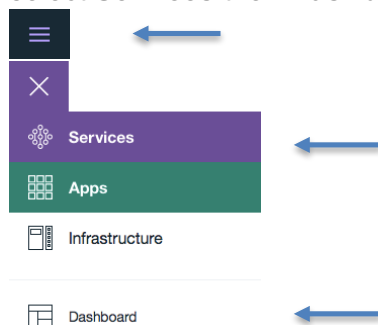
Organization ID: _____

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 should leave this browser tab open for validation testing 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 III: 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 it 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 **OPEN** .

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 it will make returning to the Bluemix dashboard much easier.



- Click on **Work with Tables** and then **Add Table** in order to create a new table.
- 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.

- Click **Run DDL** 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.
- Select the “SENSEDATA” table from the table name drop down **Table Name** and 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.

Create, drop, and work with tables

[Quick tour](#)

For existing tables, you can view details, browse data, and export data. [Learn more](#)

Add Table

Delete Table

Schema

DASH12664

Table Name

SENSEDATA

Table Definition

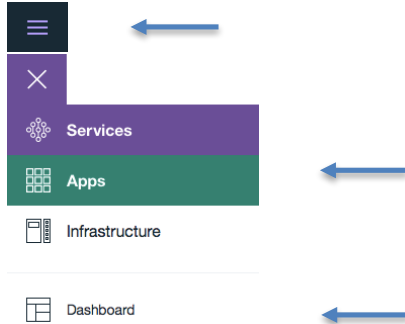
Browse Data

Primary key

	Column Name	Data Type	Length	Scale	Allow Nulls
	SENSORID	VARCHAR	20	0	Yes
	TEMPERATURE	DOUBLE	8	0	Yes
	HUMIDITY	DOUBLE	8	0	Yes
	PRESSURE	DOUBLE	8	0	Yes
	TIMESENT	TIMESTAMP	10	6	Yes

- You should leave this browser tab open for validation testing later, but, for now, switch over to the dashDB Service Details browser tab so you can return to your Bluemix dashboard.

8. In order to return to the Bluemix Dashboard, click the hamburger menu in the upper left of the page, select **Apps** then **Dashboard**.



Part IV: 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, select your application from the Bluemix Dashboard.
2. From there, you get to the Node-Red programming space, by selecting **Visit App URL**

Cloud Foundry apps / STSAWorkshops-40



3. The first time that you start the Node-Red environment, you will be presented with a Welcome wizard. Use the Next button to progress through the wizard. You will be asked to set a Username and Password that will help to secure your Node-Red space. You can choose to bypass this, but you really should add the Username and Password. Be sure to remember them, otherwise you will lose all of your work in the editor. While not normally good practice, make a note of them here:

Username: _____


Password: _____

The next part of the wizard is an opportunity to browse some Node-Red nodes that might be useful. In the interest of time, just press next here to complete the wizard.




4. Once the first-time wizard is complete, you are taken to the Node-Red launch page and can start

the editor by clicking:

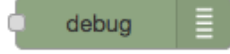
Go to your Node-RED flow editor

- 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.
- 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:** *You can get restrictive and only accept events from specific IDs and types if desired.*
- This node should only accept events of type **environment**.

Authentication		Bluemix Service
Input Type		Device Event
Device Type	<input checked="" type="checkbox"/> All or	device type e.g. armmbed
Device Id	<input checked="" type="checkbox"/> All or	device id e.g. ab12cd231a21
Event	<input type="checkbox"/> All or	environment
Format	<input type="checkbox"/> All or	json
QoS		0
Name		IoT : environment

- Now, add two more ibmiot nodes below the current one that will handle the **motion** and **joystick** events respectively.

- Next, find and add a  node (In the output section) to the right of the ibmiot nodes.
- 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.
- Connect your ibmiot nodes to your debug node by clicking and dragging from one connection point

to the other . **Note:** *You can have several nodes connecting to a single connection point.*

- 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 .
- 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.
- On the right side 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. 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). **Note:** *You can flush the contents of the debug tab by clicking the trash can in the upper right corner.*
- 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. To get started, add a **function** node (function section) to your workspace and connect it to the right side of the **IoT environment** node. ***Note:** the function node has connectors on both the left and right side. The function node is designed to take an incoming message (left side), modify it in some way, and then send the message along its way using the output (right side) connector.*

15. 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. 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;
```

16. Drag another debug node into the workspace and connect it to the right side of the function node.
17. Deploy the app and verify that the output from your function node appears correct by examining the payload output in the debug tab.
18. Once the data is being formatted properly, it's time to store it into your database. Find a **dashDB** output node (storage) and connect it to the right side of the function node. ***Note:** There are two dashDB node types and they look very similar. Be sure to select the output node. It is the one that has only one connector located on the left side.*
19. Edit the dashDB node by selecting your database service from the drop-down list and specifying the table name of **SENSEDATA**.
20. 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.
21. 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 trigger test messages in order to ensure that you can control the LED from your Bluemix application. Pressing the button on the left side of the inject node allows a message on a topic to be injected into the flow. You will use the topic field to identify the type of command (alarm or message) you wish to send. The payload should be a string that will be sent on to the function node. The string will differ for each node. The inject nodes should look as follows:

Payload	Topic	Name
String: off	alarm	Turn off LED
String: color (red, blue, etc)	alarm	Send Alarm
String: any text string	message	Send Message

4. Now let's move on to the function node. As you now already have experience with the function node, enter the following code snippet into your new function node. This code will create the appropriate java script object (d) that needs to be sent, depending on the topic type. The incoming msg.payload will be used as input to the object creation. **Note:** The message command has two additional parameters, color and background. You can set them as you prefer in this code snippet.

```
// begin code
msg.eventOrCommandType = msg.topic;
if (msg.topic == "alarm") {
  msg.payload={d:{color:msg.payload}};
}
else if (msg.topic == "message") {
  msg.payload={d:{color:"navy",
                  background:"black",
                  message:msg.payload}};
}
else
  msg = null;
return msg;
// end code
```

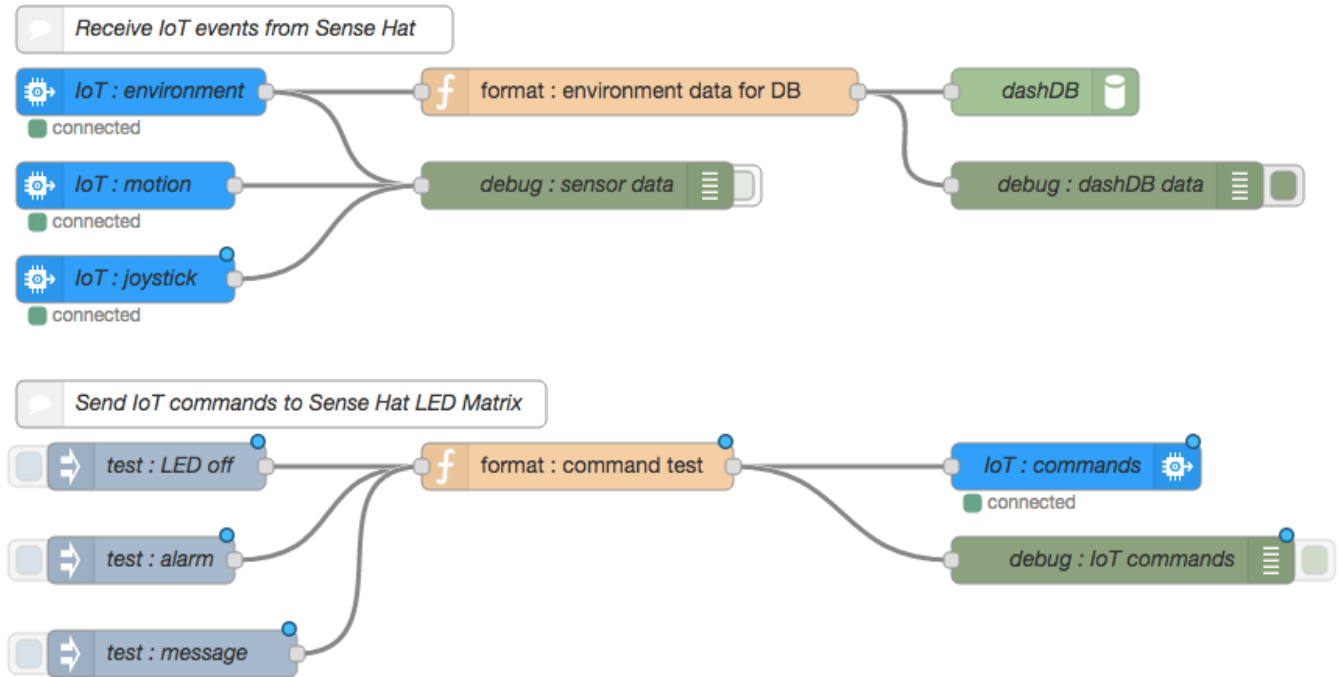
Note: Feel free to substitute any color you like for the color and background.

5. All that is left is to configure the ibmiot output node:

- The configuration of this node is very similar to the ibmiot input node.
- The output type in this case is a device command as opposed to a device event. We are sending a command to the device rather than responding to an event.
- The command type and the data are being provided in the payload of the prior node. As such, these fields will be ignored here. Put anything you like to indicate this.

Authentication	Bluemix Service
Output Type	Device Command
Device Type	SenseHat
Device Id	sensehat-xx
Command Type	n/a
Format	json
Data	n/a
QoS	0
Name	iot-new-out

- 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.
- In the end, your Bluemix application should look something like this:



PART V: Start Node-Red on Raspberry Pi

Mac OS:

- Start the Terminal application from the Mac OS Launchpad.
- 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:

- Start the PuTTY application from the Windows start menu.
- 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**)
- 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:

- At this point, you will be prompted for the user password. The Raspberry Pi credentials 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.

2. 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 VI: Create Raspberry Pi Node-Red Flows

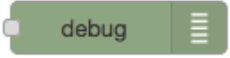
In this portion of the workshop you will create a Node-Red application on the Raspberry Pi that will collect sensor data from a device called a Sense Hat that is attached to the Pi. You will then forward that data to your IoT Service so that it can be used by a corresponding Node-Red application you 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

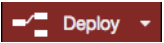
1. Node-Red programming is done via web browser. In order to get started, fire up one of your team's laptop browsers and enter **192.168.32.xx:1880** (Replace the xx with your team specific team number. For example, team 30 would use: **192.168.32.30:1880**) into the address field.



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  **Sense HAT** 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  **debug** 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  **Deploy**.
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 right side 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. 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.
10. Find the **switch** node (function section) and add it to your workspace to the right of your Sense Hat node.
11. Connect the switch node to the Sense Hat node. **Note:** *the switch node has connectors on both the left and right side. The switch node is designed to take an incoming message (left side) and route the application flow to different paths based upon the value of some element of the incoming message. It will then send the message along its way using the output (right side) connector.*
12. Open the settings for the switch node and set the property value to **msg.topic**. The contents of msg.topic is what will be used to branch application flow. Below the Property value is an area in which you can specify what you want to compare the Property to. This application will need to route its flow based upon whether the msg.topic is equal (==) to “environment”, “motion”, or “joystick”.
13. Click the drop-down menu beside the == in order to get an idea of the possible comparison operations that can be performed. Leave the value set to ==.
14. To the right of that is a field where you specify the actual value you will compare to. There is also a drop-down here that allows you to specify what type of data is being compared. Ensure that **String** is selected and then enter the string **environment** into the field.
15. Use the  button near the bottom to add the additional comparison fields for **motion** and **joystick**.

Note: *When you close the switch node settings, you will now see three separate connection points on the output side of the switch node. This is what allows you to route the flow based on the results of the comparison. The connection points are in the same order as they were added to the switch node.*

16. 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 data limit that is imposed on a free IoT service. 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, add and connect two **delay** nodes to the output of the switch node. One for the environment topic and one for the motion topic.
17. For each delay node, open the settings and set the action to **Limit rate to**.
18. Limit the rate to 1 message every 5 seconds and check the box to **drop intermediate messages**. This will cause the application to discard all messages except for one every 5 seconds.
19. Now, let's move the debug node from the Sense Hat over to the output of the switch/delay nodes. First, delete the connection between the Sense Hat node and the debug node by clicking on the line connecting them and then hitting the delete key.
20. Now, connect the output of the two delay nodes and the final switch connection (joystick topic) to the debug node. **Note:** *You can have several nodes connecting to a single connection point on another node.*

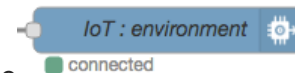
21. Redeploy the application by once again clicking deploy and you should see a dramatic decrease in the number of messages received. **Note:** *You will only see joystick topics when you actually use the Sense Hat joystick. For that reason, there is no need to delay them like the other topics.*
22. 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.
23. 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

Connect as	Gateway
	<input type="radio"/> Quickstart <input checked="" type="radio"/> Registered
Credentials	Add new wiotp-credentials... 
Device Type	SenseHat
Device Id	sensehat-xx (replace xx with team number)
Event type	environment (or motion, or joystick)
Format	▼ json
QoS	▼
Name	

define the actual event that this message comprises. Use one of the three values in each node.

24. Connect the three Watson IoT nodes to their respective delay or switch nodes.
25. Once again, deploy the application. If everything has gone well, you will see a green dot below the



IoT nodes that indicates they are connected to the IoT service

26. 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 also do this by using two separate IoT nodes (one for each command) like we did with the incoming events, but this way you will see that there are many different ways to accomplish the same task with Node-Red.

The screenshot shows the configuration settings for a Watson IoT node. The fields are as follows:

- Connect as:** Gateway
- Credentials:** PiGateway/STSAGateway
- Subscribe to:** Gateway commands (unselected), Device commands (selected)
- Device Type:** SenseHat
- Device Id:** sensehat-xx
- Command:** all commands
- QoS:** 0
- Name:** (empty field)

4. 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.

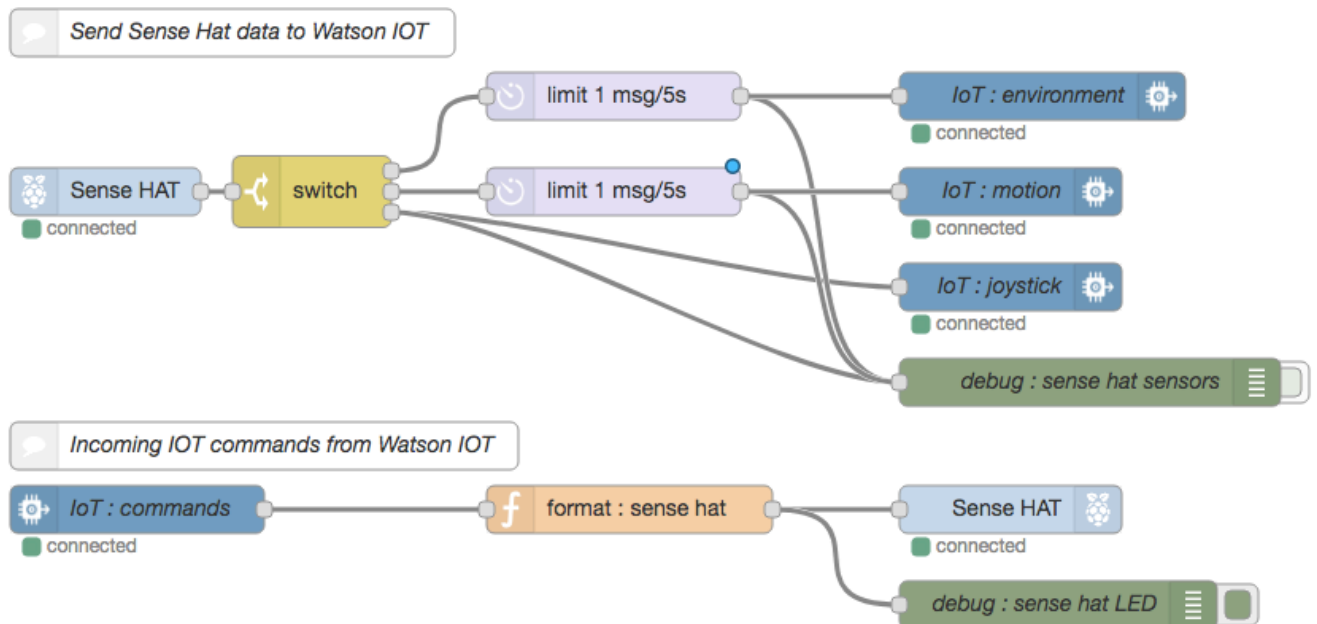
With the knowledge of the contents of the incoming command message as well as the requirements of the Sense Hat, a trivial snippet of JavaScript code can check for the command type and make the transformation. Enter the following code into the function field of the function node.

```
// 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
```

5. 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.
6. In the end, your Raspberry Pi application should look something like this:

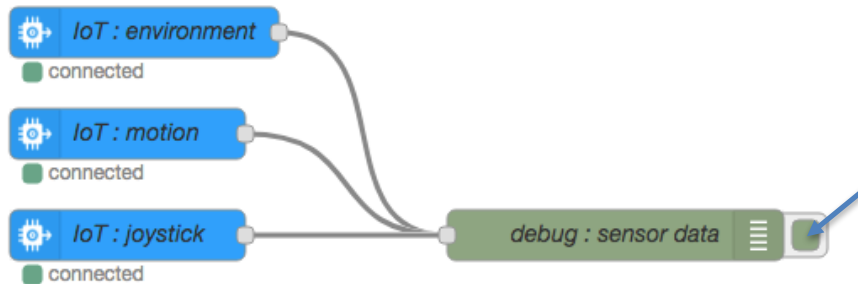


Part VII: Deploy and validate success

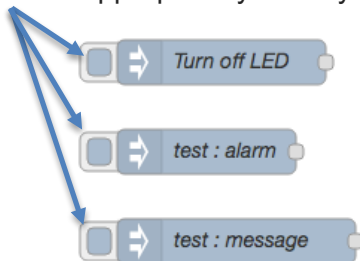
Once you have completed both the Bluemix and the Raspberry Pi Node-Red applications, its time to do some testing to ensure that your application is performing as it should be. Future workshops will be reliant on the successful execution of this workshop so please check that the following items are as expected.

Note: Be sure to deploy the latest version of your applications before beginning. If the deploy button is greyed out, then the application is already deployed.

- In the Bluemix application, use the debug node to verify that all message types are being sent to Bluemix (environment, motion, joystick). You can do this by clicking the start/stop button on the debug node and then looking at the data in the debug tab. For each event that comes in, there is an associated eventType. You should be able to see the eventType as a part of each message. If you do not, you probably do not have the debug node set to display the **complete msg object**. **Note:** You will need to actually move the tiny joystick on the Sense Hat in order to see joystick events.



- In the Bluemix application, use the inject nodes to verify that the Sense Hat LED reacts appropriately when you press the inject buttons. Be sure that all three injections work properly.



If they do not, check the debug output from the function node that formats the data for the Sense Hat. In the output data, you should verify that the **eventOrCommandType** is set to the correct command type (alarm or message). You should also verify that the **payload** looks correct.

For the alarm command, it should look like this:

```
payload: object
  d: object
    color: "navy"
```

For the message command it should look like this:

```
payload: object
  d: object
    color: "navy"
    background: "black"
```

- Recall earlier that you should have left the **IBM Watson IoT Platform Devices** browser tab open. If you switch back to that tab now, you should be able to see your newly created device sensehat-xx of type SenseHat. As a reminder, you never actually created this device or device type. They were created for you by the Raspberry Pi gateway. If you did not leave this browser tab open, you can open it again from your Bluemix dashboard. If you do not see your new device, you may need to press the refresh button.

Devices

Browse | Diagnose | Action | Device Types | Manage Schemas Refresh + Add Device

<input type="checkbox"/>	Device ID	Device Type	Class ID	Date Added	Location	
<input type="checkbox"/>	SeanniePi	RaspberryPi	Device	25 Apr 2017 11:30:56		
<input type="checkbox"/>	SeanniePi-Env	SenseEnvironment	Device	5 May 2017 08:46:30		
<input type="checkbox"/>	SeanniePi-GW	PIGateway	Gateway	5 May 2017 08:34:34		
<input type="checkbox"/>	SeanniePi-LED	SenseLED	Device	5 May 2017 08:40:53		
<input type="checkbox"/>	SeanniePi-Motion	SenseMotion	Device	5 May 2017 08:58:21		
<input type="checkbox"/>	TinyRebel	RaspberryPi	Device	25 Apr 2017 11:26:05		

- Finally, return to your dashDB: Tables browser tab. Once there, if you click on Browse Data, you should be able to see the rows of environment data being added to your table. If you did not leave this browser tab open, you can open it again from your Bluemix dashboard.

Table Definition **Browse Data**

Click a row to see its details.

Maximum number of rows to retrieve: Apply Filter Table Refresh Reset

SENSORID	TEMPERATURE	HUMIDITY	PRESSURE	TIMESENT
TinySense	34.02	39.19	986.83	2017-05-26 19:08:27
TinySense	34.13	38.24	986.77	2017-05-26 19:08:37
TinySense	33.98	39.29	986.85	2017-05-26 19:08:48
TinySense	34.02	38.71	986.84	2017-05-26 19:08:58
TinySense	34.17	39.3	986.85	2017-05-26 19:09:09
TinySense	34.4	39.22	986.77	2017-05-26 19:09:19

- If you do not see any data rows in your table, check the debug output from the function node that formats the data for dashDB. In the output data, you should verify that the **payload** looks like this:

```

Object
  SENSORID: "sensehat-xx"
  TEMPERATURE: 36.34
  HUMIDITY: 34.74
  PRESSURE: 991.11
  TIMESENT: "TIMESTAMP"

```