

Workshop 1: Transforming Products & Services with IoT

The Internet of Things (IoT) links physical things, such as weather sensors and smartphones, to applications. The connection allows applications to provide functionality based on the information it can derive from the things that are connected. IoT has applications across many industries, including government, insurance, energy, and smart homes.

The purpose of this tutorial is to work with IBM® Bluemix®, Node-RED, Watson® IoT, and a Raspberry Pi to build the foundation of an application that uses weather data, such as temperature, humidity, and pressure, that is captured by weather sensors.

Prerequisites

- You must have a [Bluemix account](#)
- You must have an instance of the [Internet of Things Platform service](#).
- You must have an instance of the [Db2 Warehouse on Cloud service](#).
- You must have a Raspberry Pi and a Sense HAT.
- You must be able to connect to your Raspberry Pi on a network and you need to know its IP address.
- You should have an SSH terminal program for connecting to the Raspberry Pi. If you are using a MacOS or Linux system, you are ready. If you are using a Windows system, install the PuTTY application, which is available at <http://www.putty.org>.

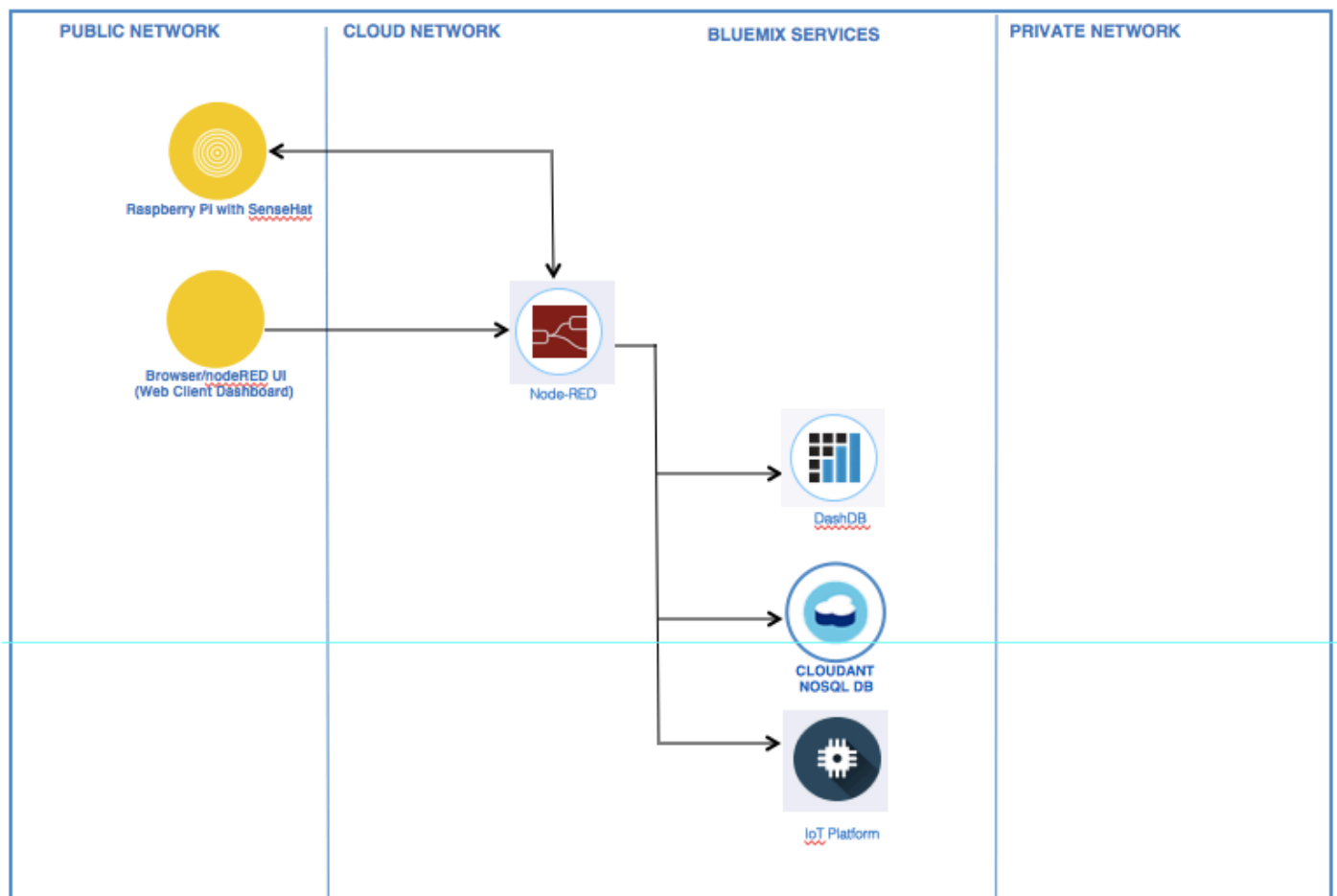
Objective:

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 IoT Platform 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: 1.5 - 2 Hours

- I. Create an Internet of Things Platform service
- II. Define the Raspberry Pi device to the IoT Platform service
- III. Create a Db2 Warehouse on Cloud service
- IV. Create a Db2 Warehouse table for environment data
- V. Create a Bluemix Node-RED Application space
- VI. Create the Raspberry Pi Node-RED flows
- VII. Create the Bluemix Node-RED flows
- VIII. Deploy and validate success

Architecture Overview:



Jedi Master Path:

Part I & II: Create an Internet of Things Platform service and define the Raspberry Pi device

1. If not already done, create an Internet of Things Platform service in your Bluemix space.
2. In your newly created IoT Platform service, define a new gateway device type called **PiGateway**.
3. Add a new gateway device using the newly defined PiGateway device type. For these workshop exercises, the device ID is assumed to be: **myPiGateway**.

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 Platform service as the gateway device will do it for you when the Sense Hat connects through it.

Part III & IV: Create a Db2 Warehouse on Cloud service and a table for environment data

1. If not already done, create a Db2 Warehouse on Cloud service in your Bluemix space.
2. Create a table called: **SENSEDATA**
3. The table should contain the following columns and data types:

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 a Bluemix Application Space

1. 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 **myWorkshop-xxx** (replace xxx with your initials to help ensure a unique name).
2. Connect your IoT and Db2 Warehouse services to your new application

PART VI: Create Raspberry Pi Node-RED Flows

1. Apply power to your Raspberry Pi by attaching a standard microUSB cable between the microUSB connector on the Raspberry Pi and a power source such as a laptop or USB power block. (booting takes less than a minute).
2. Connect to your Raspberry Pi using ssh. Your Pi can be reached at via it's IP address by using the following format: **ssh pi<ip address>**.

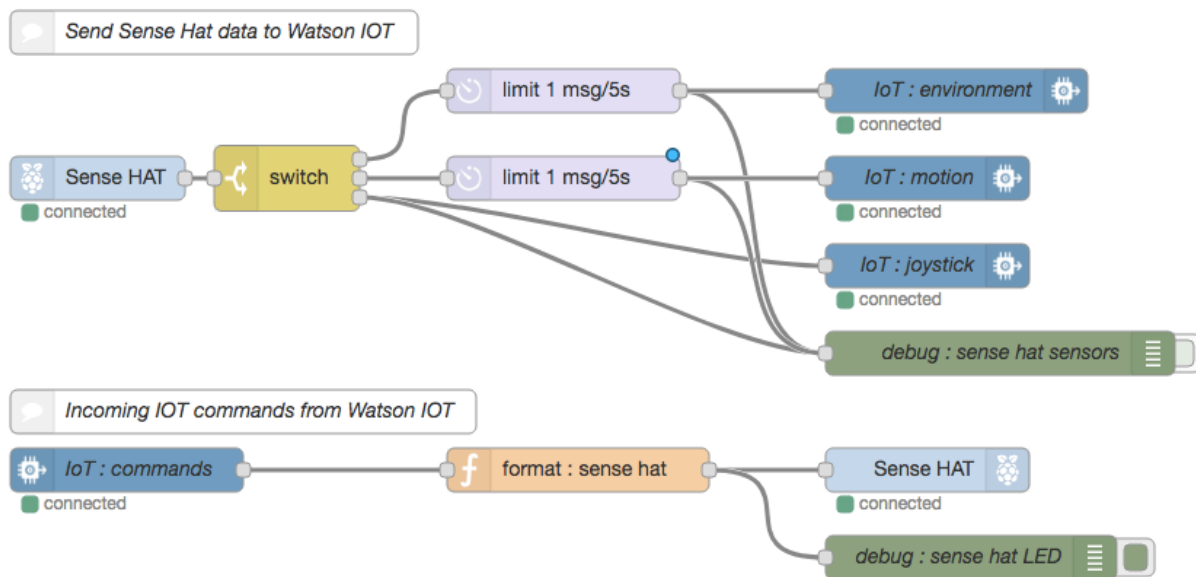
Note: Depending on your system configuration, you may be able to connect to your laptop by name rather than by ip address. To do this, you append ".local" to your Raspberry Pi hostname. For example: **ssh pi@raspberrypi.local** (pi is the default username on the Raspberry Pi).

3. The Raspberry Pi credential are:
User ID: pi
Password: raspberry

4. In order to ensure that Node-RED will restart automatically in the event of reboots and crashes, you should enable the Node-RED service with the command:
`sudo systemctl enable nodered.service`
5. Finally, start Node-RED on the Pi with the command:
`node-red-start`

Note: 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`.

6. Create the following Node-RED flow on the Raspberry Pi:



- 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 Platform service accounts.
 - Send the data to the IoT Platform 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:

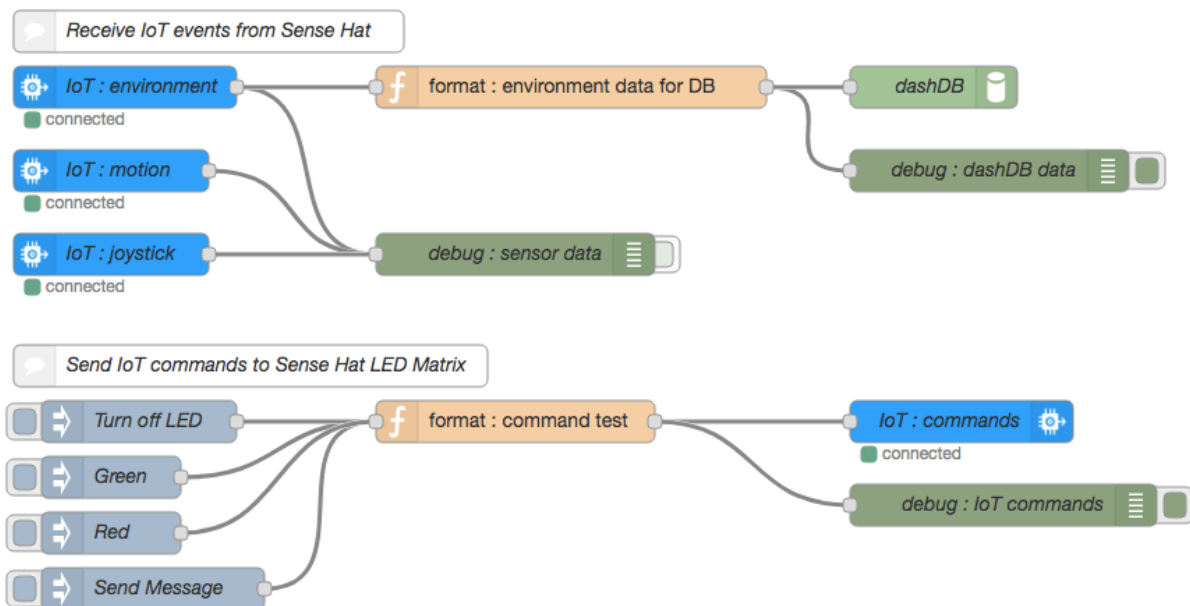
```
msg.payload = "* , * , color"
(replacing color with a color choice like red, blue, green, etc)
```

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

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

PART VII: Create Bluemix Node-RED Flows

1. Create the following Node-RED flow in your Bluemix Node-RED application:



- 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 IoT 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:

Type	Payload	Topic	Name
string	off	alarm	Turn off LED
string	green	alarm	Green
string	red	alarm	Red
string	enter any message you like here	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 VIII: Deploy and validate success

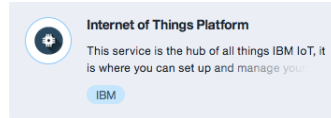
- Please skip ahead to the common Part VII Section at the bottom of this document.

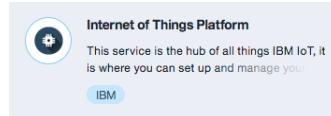
Jedi Padowan Path

PART I: Create an Internet of Things Platform service


If you completed the Internet of Things Platform service creation pre-requisite, skip ahead to Part II and proceed with defining the Raspberry Pi gateway.

1. Open a browser and sign in to one of your team's Bluemix accounts at **www.bluemix.net**.
2. You will be taken to your **Apps Dashboard** and from there you need to scroll down to **All Services** and click the  button.



3. Find and then click on the  service. You can use the type-ahead feature of the search bar to find this service quickly.

Note: Make sure that you select the IoT Platform Service, not the IoT Platform Boilerplate.

4. You will need to give your IoT Platform service a name. A default name is suggested but may be typed over. While you can name the service anything you like, these instructions will assume a name of **myWorkshop-IoT** for the IoT Platform service.
5. Click  and, after a moment, you will be taken to the “Service Details” page of your new IoT Platform service.

Part II: Define the Raspberry Pi device to the IoT Platform service


At this point, we will define the Raspberry Pi to the IoT Platform service you have created previously.

1. If you have not done so already, you need to go to your IoT Platform service. To do this, open a browser and sign in to one of your team's Bluemix accounts at **www.bluemix.net**.
2. You will be taken to your **Apps Dashboard** and from there you need to scroll down to **All Services** and click on your **myWorkshop-IoT** service.
3. This will land you on the “Service Details” page of your IoT Platform service where you should click



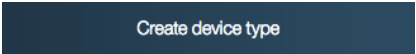









Note: The IoT Platform service should launch in a separate browser tab leaving the IoT Service Details tab open as well. Keep the Service Details tab open as it will make returning to the Bluemix dashboard much easier.

4. Now you will create a new IoT Gateway device in the IoT Platform service to represent the

Raspberry Pi. 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. However, as you will see, will not be necessary to define the Sense Hat to the IoT Platform 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 yet. So, we need to 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, so select .
8. Call the new device type **PiGateway** and give it any description you like. Leaving it empty is fine too.
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 Platform 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: **myPiGateway**. 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 one gateway and one downstream device, 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, 8 - 36 character mix of alphanumeric characters and symbols. The other option is a Self-provided token giving you the option to 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 .
- IMPORTANT: 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 Platform service.**

Organization ID: _____

Device Type: _____


Device ID: _____

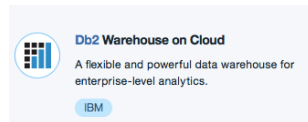
Token: _____


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 on the words  and you will be taken there with **All Services** listed front and center.

Part III: Create a Db2 Warehouse on Cloud service

If you completed the Db2 Warehouse on Cloud service creation pre-requisite, skip ahead to Part IV and proceed with defining the Db2 Warehouse table.


1. At this point, you should be in your Bluemix dashboard at **All Services**. From there, click the  button.



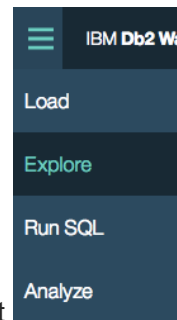
2. Find and then click on the service. You can use the type-ahead feature of the search bar to find this service quickly.
3. You will need to give your Db2 Warehouse on Cloud service a name. A default name is suggested but may be typed over. While you can name the service anything you like, these instructions will assume a name of **myWorkshop-Db2** for the Db2 Warehouse on Cloud service.
4. Click  and, after a moment, you will be returned to your Bluemix dashboard with **All Services** listed front and center.
5. Your Db2 Warehouse on Cloud service is now ready.


Part IV: Create a Db2 Warehouse table for environment data

Now, we need to create a table in the Db2 Warehouse on Cloud service to hold the environment data that is sent to your application from the Sense Hat 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 **myWorkshop-Db2** service in your Bluemix dashboard and click on it to launch the service.
2. Like the IoT Platform service, this will land you on the “Service Details” page of your Db2 Warehouse on Cloud service where you should click .


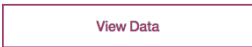
Note: The Db2 Warehouse on Cloud service should then launch in a separate browser window or tab leaving the Service Details tab open as well. Keep this tab open as it will make returning to the Bluemix dashboard much easier.



3. In order to create a new table from here, you need to select **Explore**.
4. From there you will be presented a list of database schemas. You should click on the one beginning with DASH (for example: `DASH10137`) and then click on  **New Table**.
5. Name your new table: **SENSEDATA**
6. Replace the text in the box with the following. Each line represents one column of the table along with its type:

```
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.

7. Click  and your table will be created and you will be able to see the table definition. You can also click on  in order to see the actual data stored in the table. This will come in handy later.

8. In the end, it should look something like this:

The screenshot shows the IBM Db2 Warehouse on Cloud interface. At the top, there's a navigation bar with options: Load, Explore, Run SQL, Analyze, 0%, and a user profile for tankers@us.ibm.com with an Upgrade button. Below this, the 'Database tables' section is active. It features a search bar and a toggle for 'Show system schemas'. The main area is divided into three panels: 'Schema', 'Table', and 'Table definition'. The 'Schema' panel lists various schemas, with DASH10137 selected. The 'Table' panel shows tables within DASH10137, with SENSEDATA selected. The 'Table definition' panel shows the structure of SENSEDATA, including columns like SENSORID, TEMPERATU..., HUMIDITY, PRESSURE, and TIMESENT, along with their data types and nullability. A 'View Data' button is at the bottom right.

Schema	Table	Table definition
DASH10137	SENSEDATA	SENSEDATA 1781 rows (224 KB) Updated on 7/18/2017 at 1:34:59 PM
ERRORSCHEMA	TESTTABLE	
GOSALES		
GOSALESDW		
GOSALESHR		
GOSALESMR		
GOSALESRT		
SAMPLES		
ST_INFORMTN_SCHEMA		

COLUMN NAME	DATA TYPE	NULLA...
SENSORID	VARCHAR	Y
TEMPERATU...	DOUBLE	Y
HUMIDITY	DOUBLE	Y
PRESSURE	DOUBLE	Y
TIMESENT	TIMESTAMP	Y

9. You should leave this browser tab open for validation testing later, but, for now, switch over to the Db2 Warehouse on Cloud Service Details browser tab so you can return to your Bluemix dashboard.

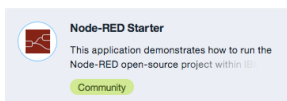
10. In order to return to the Bluemix Dashboard, click the on the words **IBM Bluemix** and you will be taken there with **All Services** listed front and center.

PART V: Create a Bluemix Node-RED 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 from a Node-RED boilerplate application template and connect the previously created IoT and Db2 Warehouse on Cloud services to it.

1. Scroll down in your Bluemix Dashboard until you get to the **All Apps** section. From here, click on:

Create App +



2. Locate the boilerplate template and click on it.

Note: You can use the type-ahead feature of the search bar to quickly find this boilerplate.


3. On the resulting page, you need to give your application a name. You can name your application anything you like provided it is unique in the Bluemix space. For consistency, this workshop will use the name **myWorkshop-xxx (Replace xx with your team number)**. Enter your application name in the application name field, and click **Create**. 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.

- You now need to create connections to your IoT and Db2 Warehouse on Cloud services. Go to the **Connections** section of your application page (listed on the menu to the left). There, as a part of the Node-RED Starter boilerplate, you will see that a Cloudant NoSQL database service has been created automatically. The Cloudant NoSQL database holds all of your application code. Here, is where you will add the connections to your two additional services.

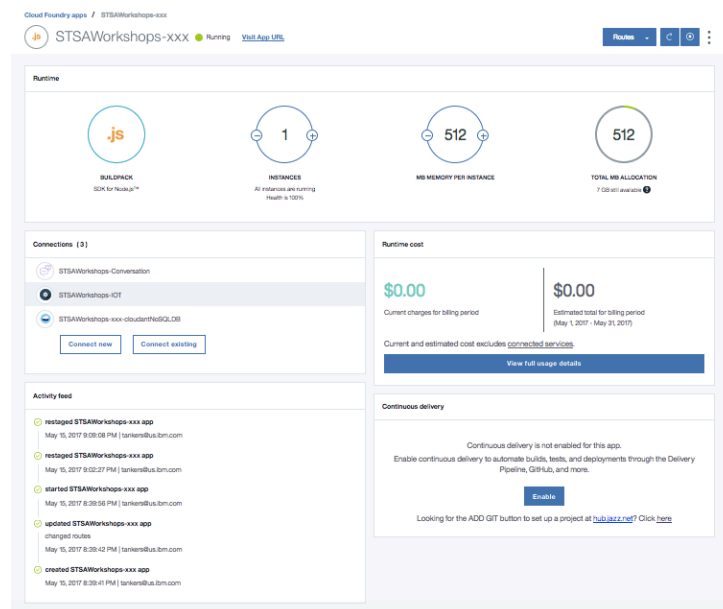
- Click on  , select your IoT Platform service  (**myWorkshop-IoT**) from the list

of existing services that you are presented, and then click .

- Select **Cancel** in response to the restaging request that pops up.

- Repeat steps 5 & 6 for your Db2 Warehouse on Cloud service  (**myWorkshop-Db2**) only this time, select **Restage** in response to the restaging request.

- You will be returned to the **Connections** section of your application. Click on **Overview** in order to see the Overview page which should look similar to this:



Leave this browser tab open. We will return to it in a few minutes.

PART VI: Create the 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 Platform 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 Platform 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.

Before you can create the Node-RED application on the Raspberry Pi, it must be powered on and the Node-RED service must be started.

1. Apply power to your Raspberry Pi by attaching a standard microUSB cable between the microUSB connector on the Raspberry Pi and a power source such as a laptop or USB power block. (booting takes less than a minute).
2. Connect to your Raspberry Pi using ssh. Your Pi can be reached at via it's IP address by using the following format: **ssh pi<ip address>**.

Note: Depending on your system configuration, you may be able to connect to your laptop by name rather than by ip address. To do this, you append ".local" to your Raspberry Pi hostname. For example: **ssh pi@raspberrypi.local** (pi is the default username on the Raspberry Pi).

- If you are using MacOS or Linux, open a terminal window to execute the ssh command.
 - If you are using windows, start the PuTTY application from the Windows start menu and provide the connection information there.
3. At this point, you will be prompted for the user password. The Raspberry Pi credentials are:
User ID: pi
Password: raspberry

Note: When you connect to the Raspberry Pi for the first time, you may see a message indicating that the authenticity of the host could not be established. Simply answer with "yes".

4. 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.
5. In order to ensure that Node-RED will restart automatically in the event of reboots and crashes, you should enable the Node-RED service with the command:
sudo systemctl enable nodered.service
6. Finally, start Node-RED on the Pi with the command:
node-red-start

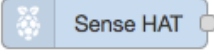
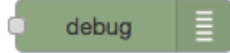




Note: When the Node-RED app starts, the last action it performs is to start a logging function. You can exit this logging, if needed, by pressing **Ctrl-C**. This will not stop Node-RED itself. If/When you want to stop Node-RED, you need to issue the command: **node-red-stop**.

Flow 1 – Sending Sensor Data

1. Node-RED programming is done via web browser. In order to get started, fire up another of your team's laptop browsers and in the address bar enter: **<ip address>:1880**. Here you replace <ip address> with the ip address of your Raspberry Pi.

Note: Depending on your system configuration, you may be able to connect to your laptop by name rather than by ip address. To do this, you append ".local" to your Raspberry Pi hostname. For example: **ssh pi@raspberrypi.local** (pi is the default username on the Raspberry Pi).

Also 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 Raspberry Pi.

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 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. The switch node has connectors on both the left and right side. The switch node is designed to take an incoming message from 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 connectors on the right.
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 msg.topic 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  **+ add** 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 appear in the switch node settings.

16. One thing that you may have noticed when viewing the debug output is sheer volume 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 Platform service, you would quickly use up the 200MB data limit that is imposed on a free IoT Platform service. This is also a consideration for your clients as the IoT Platform service does have data transmission charge. To deal with this, add two **delay** nodes. Connect one to the environment output and one to the motion output of the switch node.
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 Platform 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 IoT node will send a separate event type to the IoT Platform 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 specify 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 in the image.
- The value entered into Event type will define the actual event that this message comprises. Use one of the three values in each node.

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

24. Connect the three 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 now connected to the IoT Platform service

26. This concludes flow 1.

Flow 2 – Receive IoT Commands

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.

Note: You do not need to put this second flow on a separate tab. You can just move to some empty space below the first flow.

2. Like before, set the debug node to show the **complete msg object**.
3. The 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 device, 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 IoT node. In this case, you will still take different action based on the command type, but will determine the command type with javascript code in the next node. You could also do this by using two separate IoT nodes (one for each command) like we did for the outbound events, but this way you will see that there are many different ways to accomplish the same task with Node-RED.

Connect as	<div>Gateway</div>
Credentials	<div>PiGateway/STSAGateway</div>
Subscribe to	<div> <input type="radio"/> Gateway commands <input checked="" type="radio"/> Device commands </div>
Device Type	<div>SenseHat</div>
Device Id	<div>sensehat-xx (replace xx with team number)</div>
Command	<div>all commands</div>
QoS	<div>0</div>
Name	<div></div>

4. The function node is a powerful node that allows you to incorporate your own program logic into a Node-RED application. A 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 node and put it into the format that a Sense Hat expects. The msgs coming from the 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, the Sense Hat node expects a 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.payload format is a bit more detailed:

```
msg.color = "color"
(replacing color with a color choice like red, blue, green, etc)
msg.background = "color"
(replacing color with a color choice like red, blue, green, etc)
msg.payload = "message to display"
(If the message string is only a single character, it will not scroll)
```

With the knowledge of the contents of the incoming msg format 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

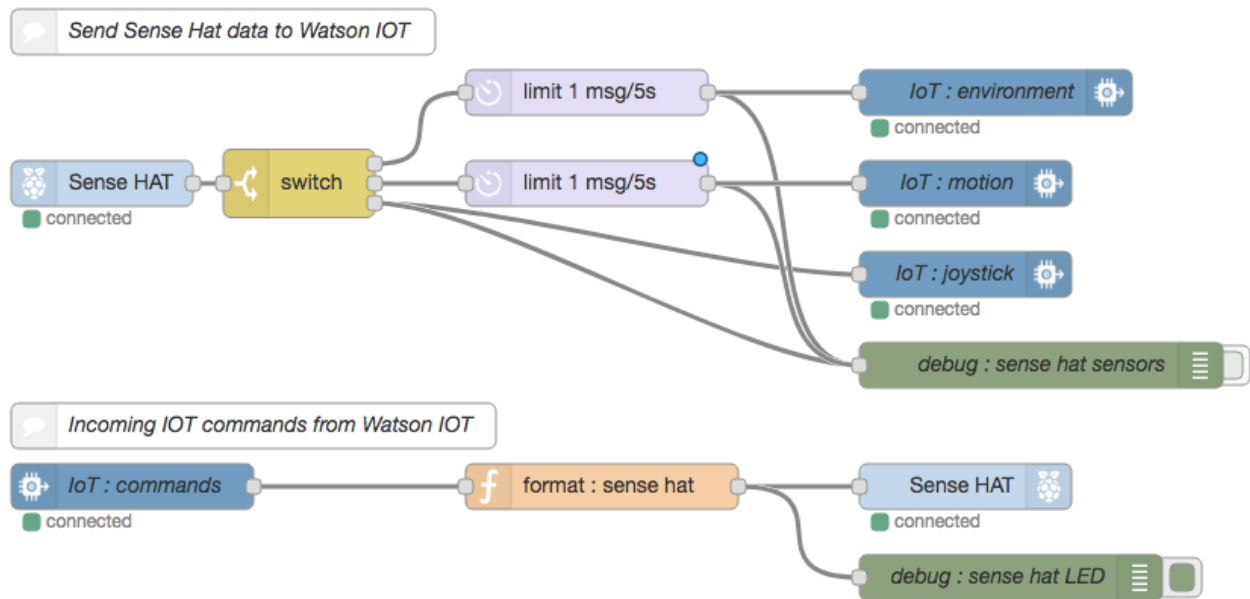
d = msg.payload.d;

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
```

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



Part VII: Create the Bluemix Node-RED flows

In this portion of the workshop you will create a Node-RED application on Bluemix that will receive the sensor data sent to it from a corresponding application on a Raspberry Pi. You will then store some of that data in your Db2 Warehouse on Cloud 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. The motion and joystick events will just terminate in a debug node so that you can verify that they are being received. Additionally, this application will 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 – Receive IoT Events

- Node-RED programming is done via web browser. In order to get started, return to your Bluemix applications' Overview page which we left open at the end of Part V.
- From there, you get to the application's Node-RED programming space, by selecting

Cloud Foundry apps / STSAWorkshops-40



STSAWorkshops-40

Running

[Visit App URL](#)

Visit App URL

near the top of the page.

- 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

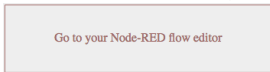
and Password that will help to secure your Node-RED space. You can choose to bypass this, but you really should add the Username and Password. Be sure to remember them, otherwise you will lose all of your work in the editor. While not normally good practice, make a note of them here:


Username: _____

Password: _____

Note: Do not forget these values or you will lose all of your work on this application as you will have no choice but to start over.

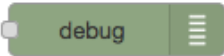
The next part of the wizard is an opportunity to browse some Node-RED nodes that might be useful. In the interest of time, just press next here to complete the wizard.


4. Once the first-time wizard is complete, you are taken to the Node-RED launch page and can start the editor by clicking: .

5. 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.
6. The ibmiot input node will receive the events that are sent from the Raspberry Pi application. You will need to configure the node by double-clicking on it to open its settings panel.


- Because the IoT Platform 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 needed by your application.
- This node should only accept events of type **environment**. You will create additional nodes to handle the joystick and motion events.


Authentication	<input checked="" type="checkbox"/> Authentication	Bluemix Service
Input Type	<input checked="" type="checkbox"/> 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	<input type="checkbox"/> QoS	0
Name	<input type="checkbox"/> Name	IoT : environment

7. Now, add two more ibmiot nodes below the current one that will handle the **motion** and **joystick** events respectively.
8. Next, find and add a  node (In the output section) to the right of the ibmiot nodes.
9. 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.

10. 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.

11. If the Raspberry Pi side of this project is complete, you will be able to verify that you are receiving events from the Raspberry Pi by clicking .
12. 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.

13. 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.

14. Now you will store the incoming environment event data into your Db2 Warehouse 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. A 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 node and put it into the format that the Db2 Warehouse 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 IoT msg into that which is expected by Db2 Warehouse.

```
// Format Sensor Data for Db2 Warehouse

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 application, 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 – Send Commands to Device

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 seven nodes. Four **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.

Note: You do not need to put this second flow on a separate tab. You can just move to some empty space below the first flow.

2. Like before, set the debug node to show the **complete msg object**.
3. The four inject nodes will be used to trigger messages that will 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:

Type	Payload	Topic	Name
string	off	alarm	Turn off LED
string	green	alarm	Green
string	red	alarm	Red
string	Enter any message you like here	message	Send Message

Note: You will be using the Red and Green buttons to provide answers to questions over the course of the event.

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.

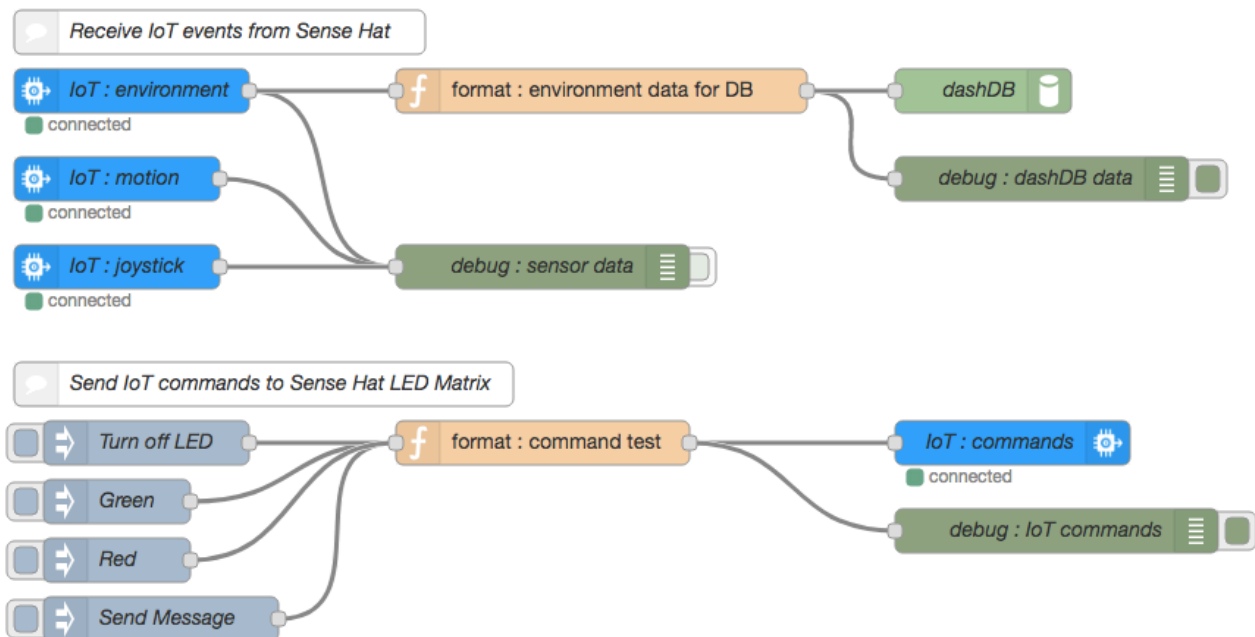
5. All that is left is to configure the ibmiot output node:

- 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 receiving an event from it.
- The command type and the data contents are being provided in the msg.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 (replace xx with team number)
Command Type	n/a
Format	json
Data	n/a
QoS	0
Name	lot-new-out

6. With that, the Bluemix side of this application is complete. Click on **Deploy** to start the application. And test the flows and ensure that everything is working as expected. Part VIII of this workshop has some verification tests you can use.

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

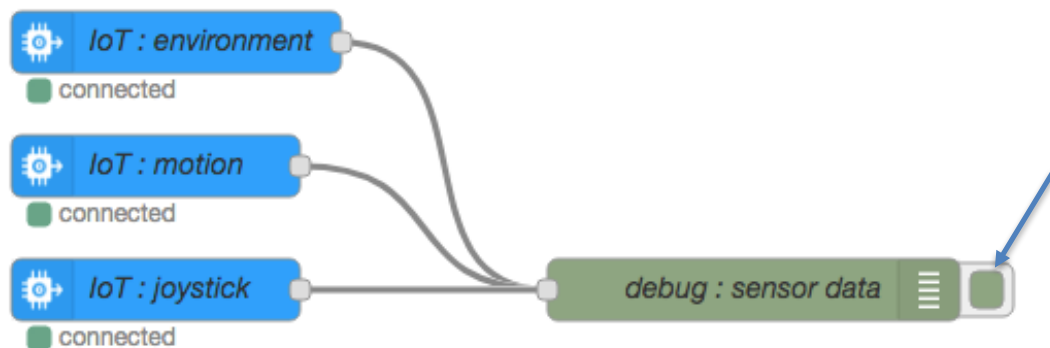


Part VIII: Deploy and validate success

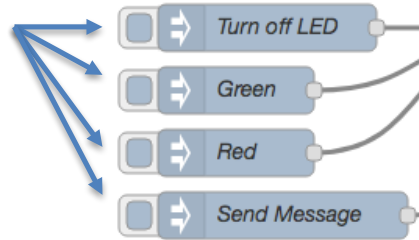
Once you have completed both the Bluemix and the Raspberry Pi Node-RED applications, it's time to do some testing to ensure that your application is performing as it should be. Future workshops will be reliant on the successful execution of this workshop so please check that the following items are as 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 four 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 Db2 Warehouse: 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: ^ v Apply 🔍 📊 📄 🔄

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 Db2 Warehouse. 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"

```

Useful Resources:

- Raspberry Pi: <https://www.raspberrypi.org/>
- Installing Raspberry Pi Operating Systems: <https://www.raspberrypi.org/downloads/>
- Node-RED on Raspberry Pi: <https://nodered.org/docs/hardware/raspberrypi.html>
- Raspberry Pi 3 Raspbian Jessie with WiFi, ssh, headless setup with no keyboard or ethernet: <https://caffinc.github.io/2016/12/raspberry-pi-3-headless/>
- Parts for IoT experimentation: <https://www.adafruit.com>
<https://www.sparkfun.com/>
<http://www.canakit.com/>