# Maxis NB-IoT Challenge Workshop

In the end of the workshop, you will learn how to:

- 1. Interface, control and acquire data from various kinds of sensors using Raspberry Pi with Grove Pi.
- 2. Establish connection between Raspberry Pi and Maxis NB-IoT network.
- 3. Send sensor telemetry from Raspberry Pi to Microsoft Azure Cloud via NB-IoT network.
- 4. Create simple server and web application to view and monitor the activity of the sensors connected to Raspberry Pi.

## Courses

**Microsoft Azure Basic Setup** 

**Rasbperry Pi + Azure Integration** 

Raspberry Pi + Grove Pi Setup

Raspberry Pi + NB-IoT Setup

**Build Sensors Monitoring Web Application hosted at Azure Cloud** 

## Downloads

- Related materials are at https://github.com/CytronTechnologies/maxisnbiot-hackathon.
- You can either git clone this repo or simply download it.

# Create lot Hub and Devices In Azure

# Steps

## Sign up for Free Microsoft Azure Account

To create and use Azure services, you will eventually need an account.

Remember, the account is free - you will not be charged for any services until you convert the account to a pay-as-you-go account.



#### Note

You will need a valid credit card to create the account. This is used for age and identity validation only. Your card will not be charged until you decide to upgrade the free/trial account to a full subscription.

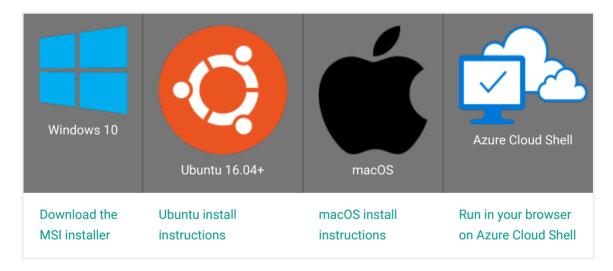
- 1. In a web browser, navigate to azure.microsoft.com, and click **Free Account**.
- 2. On the Create your Azure free account today page, click Start free > button.
- 3. Scroll down through the page it lists the products you can access for free, as well as the free products available for the first year.
- Click the Start free > button. You'll be prompted to sign in with your Microsoft account. Sign in with your Microsoft credentials or create a new free Microsoft account.
- 5. On the **About you** page, select your correct country or region, and then enter your first and last names, along with your email address and phone number. Depending on your country, you may see additional fields, such as a VAT number. Click **Next** to continue.
- 6. On the **Identity verification by phone** screen, select your country code, and type the number of a telephone to which you have immediate access.
- 7. You have the option of text or callback to obtain a verification code. Click the relevant button, type the code in the **Verification code** box, and click **Verify code**.

- 8. If the verification code is correct, you will now be asked to enter details of a valid credit card. Enter the card number, the expiration date, the CVV number, your name, and address, and click **Next**.
- Finally, check the box to accept the subscription agreement, privacy statement, and communications policy. The second checkbox is optional. Now click Sign up.



## Install Azure Command-Line Interface (CLI)

The Azure command-line interface (CLI) is Microsoft's cross-platform command-line experience for managing Azure resources. The Azure CLI is easy to learn and the perfect tool for building custom automation that works with Azure resources.



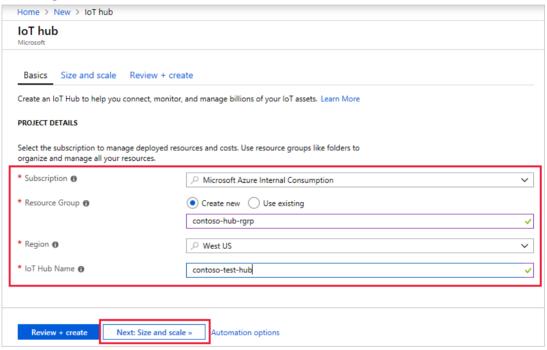
## See all of the supported installation platforms.



## Create IoT Hub

This section describes how to create an IoT hub using the Azure portal.

- 1. Log in to the Azure portal.
- 2. Choose +Create a resource, then Search the Marketplace for the IoT Hub.
- 3. Select **IoT Hub** and click the **Create** button. You see the first screen for creating an IoT hub.

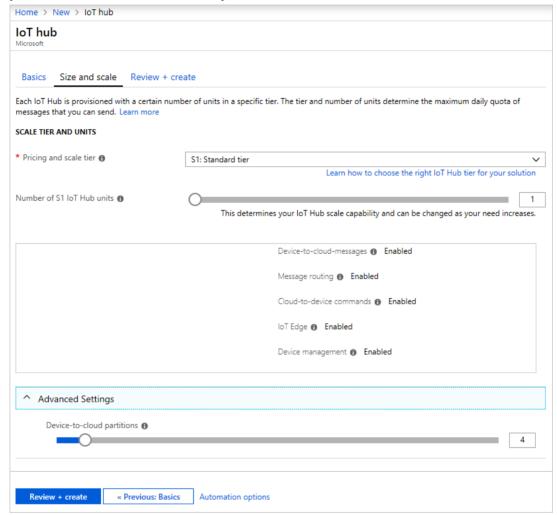


- 4. Fill in the fields.
  - **Subscription**: Select the subscription to use for your IoT hub.
  - Resource Group: You can create a new resource group or use an
    existing one. To create a new one, click Create new and fill in the name
    you want to use. To use an existing resource group, click Use existing
    and select the resource group from the dropdown list. For more
    information, see Manage Azure Resource Manager resource groups.
  - Region: This is the region in which you want your hub to be located.
     Select the location closest to you from the dropdown list.
  - IoT Hub Name: Put in the name for your IoT Hub. This name must be globally unique. If the name you enter is available, a green check mark appears.



Because the IoT hub will be publicly discoverable as a DNS endpoint, be sure to avoid entering any sensitive or personally identifiable information when you name it.

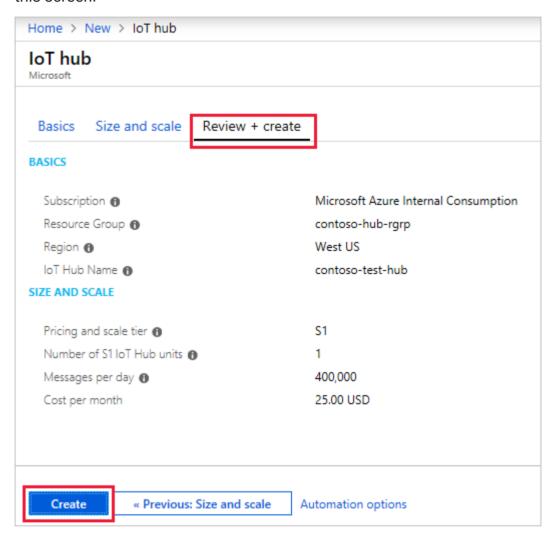
5. Click **Next: Size and scale** to continue creating your IoT hub. On this screen, you can take the defaults and just click **Review + create** at the bottom.



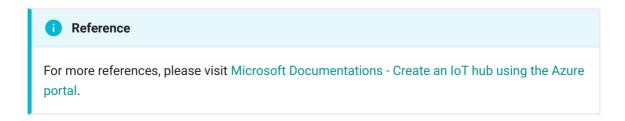
- Pricing and scale tier: You can choose from several tiers depending on how many features you want and how many messages you send through your solution per day. The free tier is intended for testing and evaluation. It allows 500 devices to be connected to the IoT hub and up to 8,000 messages per day. Each Azure subscription can create one IoT Hub in the free tier.
- IoT Hub units: The number of messages allowed per unit per day depends on your hub's pricing tier. For example, if you want the IoT hub to support ingress of 700,000 messages, you choose two S1 tier units.

For details about the other tier options, see Choosing the right IoT Hub tier.

- Advanced / Device-to-cloud partitions: This property relates the deviceto-cloud messages to the number of simultaneous readers of the messages. Most IoT hubs only need four partitions.
- 6. Click **Review + create** to review your choices. You see something similar to this screen.



7. Click **Create** to create your new IoT hub. Creating the hub takes a few minutes.



## Register lot Devices

A device must be registered with your IoT hub before it can connect. In this quickstart, you can use **local terminal** or **Azure Cloud Shell** to register a simulated device.

1. Login to your Azure Account with command below.

```
az login
```

2. Run the following command to create the device identity.

```
az iot hub device-identity create --hub-name <YourIoTHubName>
  --device-id <NameOfYourDevice>
```

**YourloTHubName**: Replace this placeholder below with the name you choose for your IoT hub.

**NameOfYourDevice**: The name of the device you're registering. You can choose any name for your device. In following workshop, you will need to use the same device name throughout this article. You will also need to update the device name in the sample applications before you run them.

3. Run the following commands in local terminal or Azure Cloud Shell to get the *device connection string* for the device you just registered:

```
az iot hub device-identity show-connection-string --hub-name
<YourIoTHubName> --device-id <NameOfYourDevice> --output table
```

**YourloTHubName**: Replace this placeholder below with the name you choose for your IoT hub.



Make a note of the device connection string, which looks like

HostName=<YourIoTHubName>.azure-

devices.net;DeviceId=<NameOfYourDevice>;SharedAccessKey={YourSharedAccessKey} .
You use this value later in the guickstart.

4. You also need a *service connection string* to enable the back-end application to connect to your IoT hub and retrieve the messages. The following command retrieves the service connection string for your IoT hub:

```
az iot hub show-connection-string --name YourIoTHubName --
policy-name service --output table
```

**YourloTHubName**: Replace this placeholder below with the name you choose for your IoT hub.



#### Note

Make a note of the service connection string, which looks like

HostName=<YourIoTHubName>.azure-

 $devices.net; Shared Access Key Name = service; Shared Access Key = \{Your Shared Access Key\} \; . \\$ 

You use this value later in the quickstart.



#### Important!!

The service connection string is different from the device connection string.

## Send simulated telemetry

The simulated device application connects to a device-specific endpoint on your IoT hub and sends simulated temperature and humidity telemetry.

In this workshop, we will be using **Python** as main language.

1. Check if Python is installed.

You can verify the current version of Python on your development machine using one of the following commands:

## Python2

```
python --version
```

### Python3

```
python3 --version
```

You can download Python from Python.org.

 Run the following command to add the Microsoft Azure IoT Extension for Azure CLI to your Cloud Shell instance. The IOT Extension adds IoT Hub, IoT Edge, and IoT Device Provisioning Service (DPS) specific commands to Azure CLI.

```
az extension add --name azure-cli-iot-ext
```

Download the sample Python project from https://github.com/Azure-Samples/azure-iot-samples-python/archive/master.zip and extract the ZIP archive.

3. Under Linux or MacOS system, make sure Boost Library is installed. Skip this step if Windows system is used.

#### Linux

```
apt-get install libboost-python-dev
```

#### **MacOS**

```
brew install boost
```

- 4. Navigate to the root folder of the downloaded sample Python project. Then navigate to the **iot-hub\Quickstarts\simulated-device** folder.
- 5. Open the **SimulatedDevice.py** file in a text editor of your choice.
  - Replace the value of the CONNECTION\_STRING variable with the device connection string you made a note of previously. Then save your changes to **SimulatedDevice.py** file.
- 6. In the local terminal window, run the following commands to install the required libraries for the simulated device application:

```
pip install azure-iothub-device-client
```

7. In the local terminal window, run the following commands to run the simulated device application:

```
python SimulatedDevice.py
```

The following screenshot shows the output as the simulated device application sends telemetry to your IoT hub:

```
C:\repos\iot-hub-quickstarts-python\simulated-device>python SimulatedDevice.py
IoT Hub Quickstart #1 - Simulated device
Press Ctrl-C to exit
IoT Hub device sending periodic messages, press Ctrl-c to exit
Sending message: {"temperature": 25.57, "humidity": 71.57}
Sending message: {"temperature": 21.78, "humidity": 63.41}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 28.86, "humidity": 71.56}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 30.92, "humidity": 74.79}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 20.50, "humidity": 73.00}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 24.16, "humidity": 64.02}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 31.89, "humidity": 73.26}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 34.25, "humidity": 62.60}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 26.87, "humidity": 72.21}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 23.57, "humidity": 72.21}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 20.59, "humidity": 70.30}
IoT Hub responded to message with status: OK
Sending message: {"temperature": 20.59, "humidity": 70.30}
IoT Hub responded to message with status: OK
```



- b. Check that you have the right version of C++ runtime Microsoft Visual C++ Redistributable for Visual Studio 2019. (We recommend the latest).
- c. Verify that you have installed the iothub client: pip install azure-iothub-device-client.

## Read telemetry from Cloud

**Troubleshoot** 

The IoT Hub CLI extension can connect to the service-side **Events** endpoint on your IoT Hub. The extension receives the device-to-cloud messages sent from

your simulated device. An IoT Hub back-end application typically runs in the cloud to receive and process device-to-cloud messages.

1. Run the following commands in local terminal or Azure Cloud Shell, replacing YourIoTHubName with the name of your IoT hub:

```
az iot hub monitor-events --hub-name YourIoTHubName --device-
id <NameOfYourDevice>
```

2. The following screenshot shows the output as the extension receives telemetry sent by the simulated device to the hub:

## References

For more info, please visit https://docs.microsoft.com/en-us/azure/iot-hub/

# Getting Started with Raspberry Pi

In this course, we will be using Raspberry Pi in headless mode (a.k.a no keyboard, mouse or monitor attached to Raspberry Pi). We will use our personal computer to remote access to Raspberry Pi instead. The Raspberry Pi in this workshop has prebuild image which has VNC and SSH capability.

# Getting IP address

First of all, we will need IP address of Raspberry Pi. Power up your Raspberry Pi with NB-IoT Hat attached. Then access to to get your Raspberry Pi Information by referring IMEI from NB-IoT HAT.

## **VNC**

- · Download VNC Viewer from here
- Access to Raspberry Pi using <yourip>:5901

## SSH

- Windows
  - · Download putty software from here.
- · Mac OS, Linux
  - Run command ssh pi@<yourip> in local terminal.

# Grove Pi with Raspberry Pi

This course explains how to use Grove Pi HAT with Rasbperry Pi to interface, control and get data from sensors.

The GrovePi can be programmed in **Python**, **C**, **C**#, **Go**, and **NodeJS** on the Raspberry Pi. We will be using **Python** as main language for the course.

# Install Grove Pi Python

```
$ sudo curl -kL dexterindustries.com/update_grovepi | bash
$ sudo pip install grovepi
$ sudo reboot
```

# Getting started with Example DHT Temperature & Humidity Sensor

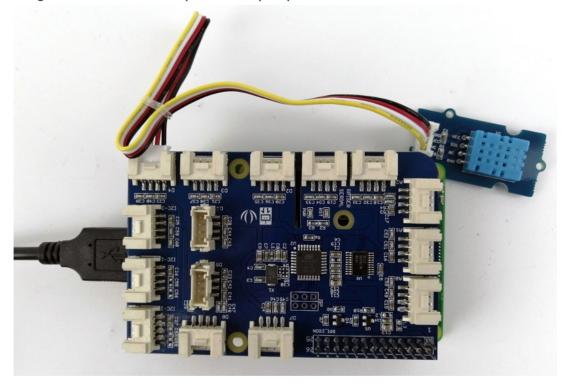
Sounds familiar? Temperature and humidity are the values that are being sent to Azure Cloud if you are following previous course. Right now we can send those values from actual hardware DHT11 Temperature & Humidity Sensor.

# Steps

1. Plug the Grove Pi HAT onto Raspberry Pi.



2. Plug the DHT11 Sensor (Blue color) to port D4 on GrovePi.



- 3. Create a python script called dht.py.
- 4. Copy the following content into dht.py.

#!/usr/bin/env python
import grovepi

```
import math
# Connect the Grove Temperature & Humidity Sensor Pro to
digital port D4
# This example uses the blue colored sensor.
# SIG, NC, VCC, GND
sensor = 4 # The Sensor goes on digital port 4.
# temp_humidity_sensor_type
# Grove Base Kit comes with the blue sensor.
blue = 0  # The Blue colored sensor.
white = 1  # The White colored sensor.
while True:
   try:
        # This example uses the blue colored sensor.
        # The first parameter is the port, the second
parameter is the type of sensor.
        [temp,humidity] = grovepi.dht(sensor,blue)
        if math.isnan(temp) == False and math.isnan(humidity)
== False:
            print("temp = %.02f C humidity =%.02f%%"%(temp,
humidity))
    except IOError:
        print ("Error")
```

- 5. Run command python dht.py to run the program.
- 6. If the setup is correct, we will see temperature and humidity values are showing up.

# References

- For more examples, you can visit https://github.com/DexterInd/GrovePi/ tree/master/Software/Python
- 2. For other programming language, you can refer to here https://github.com/ DexterInd/GrovePi/tree/master/Software

# Testing Raspberry Pi + SIM7000E NB-IoT Hat + Maxis NB-IoT

# NB-IoT AT command testing

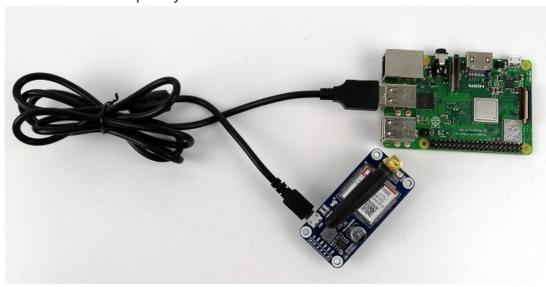
AT command test is basic step to check if NB-IoT is working in good condition.

## Prerequisite

- · Raspberry Pi
- SIM7000E NB-IoT HAT
- · MicroUSB Type B Cable

## Steps

1. Insert and lock the NB-IoT SIM card onto NB-IoT HAT (Back). Connect the NB-IoT HAT to Raspberry Pi Board via USB cable.



2. Run the command below on Raspberry Pi to check availability of HAT. It will show /dev/ttyUSB2 if it exists.

ls /dev/ttyUSB2

3. Run the command below to install screen utility on Raspberry Pi, which allows us to communicate to HAT via UART.

```
sudo apt-get install screen
```

4. Run below command to start.

```
screen /dev/ttyUSB2 115200
```

- 5. Type and enter AT command repeatedly until you see OK in return.
- 6. Run AT commands below to test connection with Internet.

```
->: User input
# test using AT
-> AT
0K
# check firmware version, make sure it contains SIM7000E
-> AT+GSV
SIMCOM_Ltd
SIMCOM_SIM7000E
Revision: 1351B06SIM7000E
# check service, result might be different
-> AT+CPSI?
+CPSI: GSM, Online, 502-12, 0x1054, 62168, 512 DCS 1800, -71, 0, 32-32
-> AT+CPSI?
+CPSI: NO SERVICE, Online
# check signal quality
-> AT+CSQ
+CSQ: 23,99
# configure to use NB-IoT
-> AT+CMNB=2
0K
-> AT+CNMP=38
0K
-> AT+CBANDCFG="NB-IOT",8
0K
```

```
-> AT+CSTT="M2MXNBIOT"
0K
-> AT+CPSI?
+CPSI: LTE NB-IOT, Online, 502-12, 0x74CE, 28185722, 12, EUTRAN-
BAND8, 3702, 0, 0, -10, -82, -72, 15
# check establish connection
-> AT+CIPSTATUS
0K
STATE: IP START
# start establish connection
-> AT+CIICR
0K
-> AT+CIPSTATUS
0K
STATE: IP GPRSACT
# get IP address
-> AT+CIFSR
10.247.96.227
# ping www.google.com
-> AT+CIPPING="www.google.com"
+CIPPING: 1, "172.217.166.132", 111, 52
+CIPPING: 2, "172.217.166.132", 170, 52
+CIPPING: 3, "172.217.166.132", 110, 52
+CIPPING: 4, "172.217.166.132", 128, 52
```

- 7. Run Ctrl-a then Ctrl-d to minimise the screen. To Resume the screen run screen -r.
- 8. Run Ctrl-a then type: :quit to quit the screen.

# PPP Installation on Raspberry Pi

NB-IoT HAT offers PPP (Point-to-Point Protocol) for us to connect Raspberry Pi to Internet easily. The following steps show how to establish Internet connection with Raspberry Pi.

## Prerequisite

- Raspberry Pi
- SIM7000E NB-IoT HAT
- MicroUSB Type B Cable

## **Steps**

1. Plug the NB-IoT HAT into Raspberry Pi Board using USB. Ensure the Maxis SIM card is inserted and secured on the HAT.



2. Run the command below on Raspberry Pi to check availability of HAT. It will show /dev/ttyUSB2 if it exists.

```
ls /dev/ttyUSB2
```

3. Go to directory where you have downloaded the GIT respository. E.g. /home/pi/maxis-nbiot-hackathon

```
cd /home/pi/maxis-nbiot-hackathon
```

4. Run the commands below to install PPP on Raspberry Pi as root. You will need to enter password after command sudo su

```
$ sudo su
$ cd "Raspberry Pi/PPP Installer"
```

```
$ chmod +x install.sh
$ ./install.sh
```

5. Run command below to start the connection.

```
pon
```

6. Run command below to check if interface ppp0 exists.

```
ifconfig
```

- 7. If it is successful, you should see IP exists in interface ppp0 from ifconfig command.
- 8. Run command below to stop the connection.

```
poff
```

- 9. Automate the connection during Raspberry Pi boot-up.
  - Run command nano /etc/rc.local to start edit file.
  - Insert 2 lines before line exit 0 in file

```
sleep 5
sudo pon &
exit 0
```

10. Final script will be similiar as below:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser
runlevel.
# Make sure that the script will "exit 0" on success or any
other
# value on error.
#
# In order to enable or disable this script just change the
```

- 11. Make sure you have turned off WiFi or unplug Ethernet to make sure Raspberry Pi will use PPP in next bootup.
- 12. Reboot Rapsberry Pi for step 9 to take effect.

## References

For more references, please visit https://www.rhydolabz.com/wiki/?p=16325

# Send simulated telemetry using Raspberry Pi

We will be using Raspberry Pi to send simulated temperature and humidity telemetry for testing purpose. In this workshop, we will be using **Python** as main language.

# Steps

1. Check if Python is installed. You can verify the current version of Python on your development machine using one of the following commands:

## Python2:

```
python --version
```

## Python3:

```
python3 --version
```

2. You can download Python by running below commands.

### Python2:

```
sudo apt-get install python
```

## Python3:

```
sudo apt-get install python3
```

- 3. Download the sample Python project from https://github.com/Azure-Samples/azure-iot-samples-python/archive/master.zip and extract the ZIP archive.
- 4. Make sure Boost Library is installed.

```
sudo apt-get install libboost-python-dev
```

- 5. Navigate to the root folder of the downloaded sample Python project. Then navigate to the **iot-hub\Quickstarts\simulated-device** folder.
- Open the SimulatedDevice.py file in a text editor of your choice.
   Replace the value of the CONNECTION\_STRING variable with the device

connection string you made a note of previously. Then save your changes to **SimulatedDevice.py** file.

7. In the local terminal window, run the following commands to install the required libraries for the simulated device application:

```
sudo pip install azure-iothub-device-client
```

8. In the local terminal window, run the following commands to run the simulated device application:

sudo python SimulatedDevice.py



#### **Troubleshoot**

**To avoid the import iothub\_client error** The current version of the Azure IoT SDK for Python is a wrapper over our C SDK. It is generated using the Boost library. Because of that, it comes with several significant limitations. See more details here

- a. Check that you have the right version of Python. Be aware that only certain versions works fine for this sample.
- b. Check that you have the right version of C++ runtime Microsoft Visual C++ Redistributable for Visual Studio 2019. (We recommend the latest).
- c. Verify that you have installed the iothub client:

sudo pip install azure-iothub-device-client

# IoT Web Application using Node Red

# Running Node Red on your Local Machine

- Node Red requires NodeJS installation. Refer here to install latest NodeJS package.
- 2. In local terminal window, install Node Red.

```
npm install node-red --unsafe-perm
```

- 3. Run command node-red to start Node Red.
- 4. If the output shows **Server now running at http://127.0.0.1:1880/**, navigate to 127.0.0.1:1880 in your web browser. You should be able to see Node Red webview.
- 5. Press Ctr1+C in terminal window to close Node Red.
- 6. (Optional) By default, the Node-RED editor is not secured anyone who can access its IP address can access the editor and deploy changes. This is only suitable if you are running on a trusted network. See this guide to secure your Node-RED application.

# Design your Node Red Application

In this application, we are going to learn to:

- Run an application which reads telemetry of the device using Azure Node Red Library.
- Create a web UI view to display telemetry of the device.
- Send an email when alert is triggered.

## Read Telemetry from Azure IoT Hub

1. Start Node Red, and browse to http://127.0.0.1:1880

- 2. Install node-red-contrib-azure-iot-hub via **Manage Palette**. Manage Palette can be found under dropdown menu from menu button at top right corner of browser window.
- From left pane of the browser window, scroll and search for Azure IoT Hub
   Receiver node under Cloud category, then drag it into the center of
   dashboard.



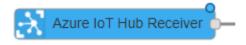
4. From the same left pane under **output** category, drag **debug** node to the dashboard. (It will be renamed as msg.payload once it is placed at dashboard)

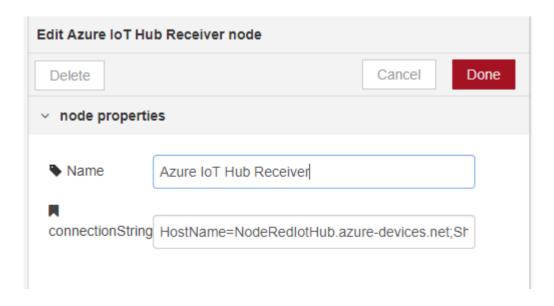


5. Link it to Azure IoT Hub Receiver node.



6. Double-click on the Azure IoT Hub Receiver node and enter your Iot Hub connectionString for your Azure IoT Hub and click Done.



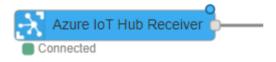


7. Click **Deploy**.

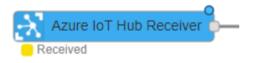


8. You should see the below messages on your command line from where you are running NodeRED. The Azure IoT Hub Receiver node should now say 'Connected'.

```
13 Mar 13:27:06 - [info] [azureiothubreceiver:Azure IoT Hub Receiver] Created Event Hub partition receiver: 0
13 Mar 13:27:06 - [info] [azureiothubreceiver:Azure IoT Hub Receiver] Created Event Hub partition receiver: 3
13 Mar 13:27:07 - [info] [azureiothubreceiver:Azure IoT Hub Receiver] Created Event Hub partition receiver: 2
13 Mar 13:27:09 - [info] [azureiothubreceiver:Azure IoT Hub Receiver] Created Event Hub partition receiver: 1
13 Mar 13:27:09 - [info] [azureiothubreceiver:Azure IoT Hub Receiver] Connected to each partition receiver - ready to receive data!
```



9. Once you have messages coming into your Azure IoT Hub, you should see them in the debug pane. The Azure IoT Hub Receiver node should now say 'Received'.

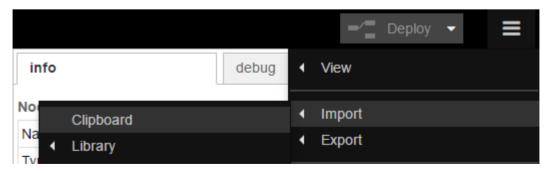


```
debug
info
                             T all nodes
                                              ŵ
3/13/2018, 2:35:48 PM node: Log
msg : Object
▶ { deviceId: "nodeRedTestDevice",
payload: object, _msgid:
"a4f8737.207729" }
3/13/2018, 2:35:49 PM node: Log
msg : Object
▶ { deviceId: "nodeRedTestDevice",
payload: object, _msgid:
"d05d8ac7.555208" }
3/13/2018, 2:35:50 PM node: Log
msg: Object
▶ { deviceId: "nodeRedTestDevice",
payload: object, _msgid:
"29fe1828.3be728" }
3/13/2018, 2:35:51 PM node: Log
msg: Object
▶ { deviceId: "nodeRedTestDevice",
payload: object, _msgid:
"b4b5b4bb.819948" }
```

## Create a Web UI to display device telemetry

We are going to create 2 widgets which show temperature and humidity of the sensor.

- 1. Install node-red-dashboard using Manage Palette.
- 2. Go to Hamburger Menu -> Import -> Clipboard.



3. Paste the following code into the "Import nodes" dialog. Then click Import.

```
[{"id":"4748d9e2.be7648","type":"tab","label":"Flow
1", "disabled":false, "info":""},
{"id":"706c73f9.d1df8c","type":"azureiothubreceiver","z":"4748d9e
Receiver", "x":140, "y":60, "wires":
[["3ba06501.57624a", "4a231742.945ba8", "6851aa80.949924"]]},
{"id":"3ba06501.57624a", "type":"debug", "z":"4748d9e2.be7648", "nam
380, "y":60, "wires":[]},
{"id":"4a231742.945ba8","type":"ui_gauge","z":"4748d9e2.be7648","
0, "width":"6", "height":"6", "gtype":"gage", "title":"Temperature", "
, "format": "{{payload.temperature}}", "min":
0, "max":"100", "colors":
["#00b500","#e6e600","#ca3838"],"seq1":"","seq2":"","x":
429, "y":147, "wires":[]},
{"id":"6851aa80.949924","type":"ui_gauge","z":"4748d9e2.be7648","
1, "width": "6", "height": "6", "gtype": "gage", "title": "Humidity", "lab
0, "max":"100", "colors":
["#00b500", "#e6e600", "#ca3838"], "seg1": "", "seg2": "", "x":
358, "y":208, "wires":[]},
{"id":"d4a658b7.4c8a28", "type":"ui_group", "z":"", "name":"Current
Reading", "tab": "b8f8f747.dab748", "order":
1, "disp":true, "width": "12", "collapse":false},
{"id":"b8f8f747.dab748","type":"ui_tab","z":"","name":"Overview",
1, "disabled":false, "hidden":false}]
Import nodes
   [{"id":"6a913d9a.fa0844","type":"azureiothubregistry","z":"e621cbfa.
   f8a5f8", "name": "Azure IoT Hub Registry", "x": 630, "y": 240, "wires":
   [["14a58301.e60a9d"]]},
   {"id":"2fd0a3f.969ce5c","type":"inject","z":"e621cbfa.f8a5f8","name":
   "Register Payload", "topic": "", "payload": "
```

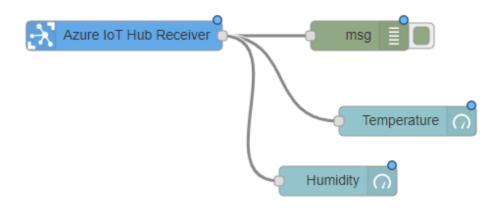
Import to

current flow

new flow

Cancel

Import



#### A

#### Important !!

Import flow including Azure IoT Hub Receiver node does not include connectionString by default. You will have to provide connectionString again. Double-click on the Azure IoT Hub Receiver node and enter your Iot Hub connectionString for your Azure IoT Hub and click Done.

4. Click Deploy.



5. Browse to http://127.0.0.1:1880/ui. You should be able to see the following screen.



To know more about how to create customized Node Red Dashboard, you can visit https://randomnerdtutorials.com/getting-started-with-node-red-dashboard/ as a starting point. There are also many websites online for you to Google!

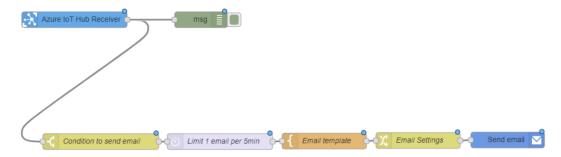
## Send an email when alert is triggered

Here we will be using **SendDrid** service to send email for alert notification. You can use other option like **Gmail** service.

- 1. Install SendGrid node node-red-contrib-sendgrid using **Manage Palette** in Node-RED.
- 2. Create the flow by importing JSON below.

```
[{"id":"4748d9e2.be7648","type":"tab","label":"Flow
1", "disabled":false, "info":""},
{"id":"706c73f9.d1df8c","type":"azureiothubreceiver","z":"4748d9e
IoT Hub Receiver", "x":140, "y":60, "wires":
[["77746ec7.63db9", "3ba06501.57624a"]]},
{"id":"3ba06501.57624a", "type":"debug", "z":"4748d9e2.be7648", "nam
380, "y":60, "wires":[]}, {"id":"4f0d6c8e.
4291e4", "type": "sendgrid", "z": "4748d9e2.be7648", "from": "no-
reply@example.com", "to": "popfibo@gmail.com", "name": "Send
email", "content": "text", "x":962, "y":288, "wires":[]},
{"id":"77746ec7.63db9","type":"switch","z":"4748d9e2.be7648","nam
to send
email", "property": "payload.temperature", "propertyType": "msg", "rul
[{"t":"gte", "v":"30", "vt":"str"}], "checkall":"true", "repair":fals
1, "x":190, "y":291, "wires":[["167c7aa.94e2385"]]},
{"id":"27bb65ef.
759cba", "type": "template", "z": "4748d9e2.be7648", "name": "Email
template", "field": "payload", "fieldType": "msg", "format": "handlebar
The temperature from {{ deviceId }} has exceeded 30°C! Current
temperature is {{ payload.temperature }}.","output":"str","x":
614, "y":290, "wires":[["38930ee.12709f2"]]}, {"id":"167c7aa.
94e2385", "type": "delay", "z": "4748d9e2.be7648", "name": "Limit 1
email per
5min", "pauseType": "rate", "timeout": "5", "timeoutUnits": "seconds", "
416, "y":291, "wires":[["27bb65ef.759cba"]]}, {"id":"38930ee.
12709f2", "type": "change", "z": "4748d9e2.be7648", "name": "Email
Settings", "rules":
[{"t":"set", "p":"payload", "pt":"msg", "to":"payload", "tot":"msg"},
{"t":"set", "p":"topic", "pt":"msg", "to":"Temperature
Alert", "tot": "str" }], "action": "", "property": "", "from": "", "to": "",
792, "y":289, "wires":[["4f0d6c8e.4291e4"]]}]
```

You will see the screen as below.



- 3. Get API key from SendGrid website.
- 4. Double click on **Send email** node (which is a sendgrid node), fill in necessary info and paste the API key.
- 5. Click Deploy.



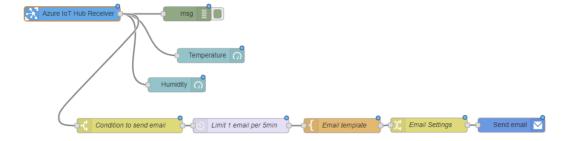
### **Explanation:**

- Condition to send email is handled by Condition to send email node. When the temperature exceeds 30°C, the email will be sent.
- Email has a cooldown period of 5 minutes, during this time the email will not be sent although the temperature exceeds 30°C. This is handled by

  Limit 1 email per 5 min node.
- Email template node allows you to decide what content to be sent in email.
- Email settings node allows you to add extra parameters like topic, cc, bcc.

## Final result

· Backend Application



#### UI Dashboard



## Full working JSON

```
[{"id":"bc488bc5.7b36c8","type":"tab","label":"Flow
1", "disabled":false, "info":""},
{"id":"72280523.04ccec","type":"ui_group","z":"","name":"Current
Reading", "tab": "619a0ac8.53b8a4", "order":
1, "disp":true, "width": "12", "collapse":false},
{"id":"619a0ac8.53b8a4","type":"ui_tab","z":"","name":"0verview",
1}, {"id": "74561fa2.60827", "type": "ui_base", "theme":
{"name":"theme-light","lightTheme":
{"default":"#0094CE", "baseColor":"#0094CE", "baseFont":"-apple-
system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen-
Sans, Ubuntu, Cantarell, Helvetica Neue, sans-
serif", "edited":true, "reset":false}, "darkTheme":
{"default":"#097479", "baseColor":"#097479", "baseFont":"-apple-
system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen-
Sans, Ubuntu, Cantarell, Helvetica Neue, sans-
serif", "edited":false}, "customTheme":{"name":"Untitled Theme
1", "default": "#4B7930", "baseColor": "#4B7930", "baseFont": "-
apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen-
Sans,Ubuntu,Cantarell,Helvetica Neue,sans-serif"},"themeState":
{"base-color":
{"default":"#0094CE", "value":"#0094CE", "edited":false}, "page-
titlebar-backgroundColor":
{"value":"#0094CE", "edited":false}, "page-backgroundColor":
{"value": "#fafafa", "edited": false}, "page-sidebar-
```

```
backgroundColor":{"value":"#ffffff","edited":false},"group-
textColor":{"value":"#1bbfff", "edited":false}, "group-
borderColor":{"value":"#ffffff","edited":false},"group-
backgroundColor":{"value":"#ffffff","edited":false},"widget-
textColor":{"value":"#111111", "edited":false}, "widget-
backgroundColor":{"value":"#0094ce", "edited":false}, "widget-
borderColor":{"value":"#ffffff","edited":false},"base-font":
 {"value": "-apple-system, BlinkMacSystemFont, Segoe
UI, Roboto, Oxygen-Sans, Ubuntu, Cantarell, Helvetica Neue, sans-
 serif"}}, "angularTheme":
 {"primary":"indigo", "accents":"blue", "warn":"red", "background":"g
 {"name":"Node-RED
Dashboard", "hideToolbar": "false", "allowSwipe": "false", "dateFormat
MM/YYYY", "sizes":{"sx":48, "sy":48, "gx":6, "gy":6, "cx":6, "cy":
6, "px":0, "py":0}}},
 {"id":"78f670e6.6d9bc","type":"ui_group","z":"","name":"Current
Reading", "tab": "78eac48a.20fd8c", "order":
1, "disp":true, "width": "12", "collapse":false}, { "id": "78eac48a.
20fd8c","type":"ui_tab","z":"","name":"Overview","icon":"dashboar
1, "disabled":false, "hidden":false},
 {"id":"b828193e.d6c448","type":"ui_group","z":"","name":"Current
Reading", "tab": "91d14d66.2b94", "order":
1, "disp":true, "width": "12", "collapse":false},
 {"id":"91d14d66.2b94","type":"ui_tab","z":"","name":"Overview","i
1, "disabled":false, "hidden":false}, {"id":"8ea2cd4f.
2878c", "type": "azureiothubreceiver", "z": "bc488bc5.7b36c8", "name":
IoT Hub Receiver", "x":140, "y":60, "wires":
 [["8f36c520.fb5e58","40b6b683.04c548","89f0aa8d.
 578d48", "96844771.27e478"]]},
 {"id":"8f36c520.fb5e58","type":"debug","z":"bc488bc5.7b36c8","nam
380, "y":60, "wires":[]},
 {"id":"40b6b683.04c548","type":"ui_gauge","z":"bc488bc5.7b36c8","
0, "width": "6", "height": "6", "gtype": "gage", "title": "Temperature", "
0,"max":"100","colors":
 ["#00b500","#e6e600","#ca3838"],"seg1":"","seg2":"","x":
429, "y":147, "wires":[]}, {"id":"89f0aa8d.
578d48", "type": "ui_gauge", "z": "bc488bc5.7b36c8", "name": "Humidity"
1, "width": "6", "height": "6", "gtype": "gage", "title": "Humidity", "lab
0, "max":"100", "colors":
 ["#00b500", "#e6e600", "#ca3838"], "seg1": "", "seg2": "", "x":
358, "y":208, "wires":[]}, {"id":"a72bcedd.
3440d", "type": "sendgrid", "z": "bc488bc5.7b36c8", "from": "", "to": "",
 email", "content": "text", "x":1056, "y":292, "wires":[]},
 {"id":"96844771.27e478","type":"switch","z":"bc488bc5.7b36c8","na
to send
email", "property": "payload.temperature", "propertyType": "msg", "rul
 [{"t":"gte", "v":"30", "vt":"str"}], "checkall":"true", "repair":fals
1, "x":260, "y":293, "wires":[["50792deb.086f44"]]},
{"id":"33d14c05.c9e964","type":"template","z":"bc488bc5.7b36c8","
```

```
template", "field":"payload", "fieldType":"msg", "format":"handlebar
The temperature from {{ deviceId }} has exceeded 30°C! Current
temperature is {{ payload.temperature }}
°C.", "output": "str", "x":703, "y":293, "wires":
[["c97cd755.df9218"]]}, {"id":"50792deb.
086f44", "type":"delay", "z":"bc488bc5.7b36c8", "name":"Limit 1
email per
5min", "pauseType": "rate", "timeout": "5", "timeoutUnits": "seconds", "
492, "y":293, "wires": [["33d14c05.c9e964"]]},
{"id":"c97cd755.df9218", "type": "change", "z": "bc488bc5.7b36c8", "na
Settings", "rules":
[{"t":"set", "p":"payload", "pt": "msg", "to": "payload", "tot": "msg"},
{"t":"set", "p": "topic", "pt": "msg", "to": "Temperature
Alert", "tot": "str"}], "action": "", "property": "", "from": "", "to": "",
884, "y":292, "wires": [["a72bcedd.3440d"]]}]
```

## Challenges

You can implement other features in this application.

- 1. Store telemetry of the device to database.
- 2. Create a web UI which shows the analytics.
- 3. .. and many more.

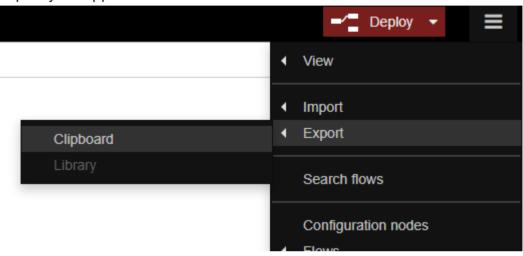
You can look into https://github.com/Azure/node-red-contrib-azure for more libraries and modules for Azure. It is up to your creativity. The other libraries you may find useful for your application:

- Azure Blob Storage
- Azure CosmosDB (formerly DocumentDB)
- Azure Event Hub
- Azure SQL
- Azure Table Storage

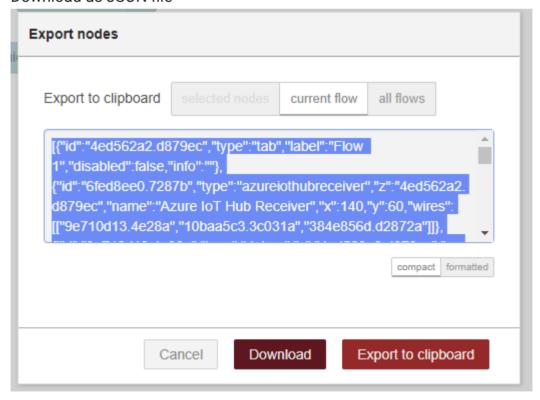
## Save your work!!!

One thing Node Red is good is that you can export your work to a JSON file. So if next time you are building a new machine and set up fresh Node Red, you can import the same file, do some configuration on nodes and you are ready to go!

· Export your application



· Download as JSON file



# Deploy on Microsoft Azure

Once we have done developing the application, it is time for us to deploy it to the cloud so we can view it anywhere as long as we have Internet.

## Create base image

- 1. Log in to the Azure console
- 2. Select Virtual Machines option under Favourite List.
- 3. In the list of Virtual Machines, select Ubuntu Server, then click 'Create'
- 4. Give your machine a name, the username you want to use and the authentication details you want to use to access the instance
- 5. Choose the Size of your instance. Remember that node.js is single-threaded so there's no benefit to picking a size with multiple cores for a simple nodered instance. A1 Basic is a good starting point
- 6. On the 'Settings' step, click on the 'Network security group' option. Add a new 'Inbound rule' with the options set as:

```
- Name: node-red-editor
- Priority: 1010
- Protocol: TCP
- Destination port range: 1880
```

7. Click 'Ok' on the Settings page, check the Summary then click 'Ok' to deploy the new instance

After a couple of minutes your instance will be running. In the console you can find your instance's IP address.

## Setup Node Red

- 1. The next task is to log into the instance then install node.js and Node-RED.
- 2. Log into your instance using the authentication details you specified in the previous stage.
- 3. Once logged in you need to install node.js and Node-RED.

```
$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E
bash -
$ sudo apt-get install -y nodejs build-essential
$ sudo npm install -g node-red
```

4. At this point you can test your instance by running command below.

```
node-red
```



#### Note

You may get some errors regarding the Serial node - that's to be expected and can be ignored.

- 5. Once started, you can access the editor at http://<your-instance-ip>: 1880/.
- 6. To get Node-RED to start automatically whenever your instance is restarted, you can use pm2:

```
sudo npm install -g pm2
pm2 start `which node-red` -- -v
pm2 save
pm2 startup
```

## Import your Application

If you have your application exported to JSON file from this step, deployment will be super easy!

- 1. Go to Hamburger Menu -> Import -> Clipboard.
- 2. Select select a file to import. Choose your desired JSON file to be import.

```
♣ select a file to import
```

- 3. Click **Import** to finish your import.
- 4. Make sure your configuration is complete (e.g. Fill in Azure IoT Hub ConnectionString, SendGrid info etc.)

# 5. Click **Deploy**.



# References

For more information, please visit https://nodered.org/docs/getting-started/azure

# Example Web Application with ExpressJS + Ionic

# Prerequisite

- 1. Install nodejs. Can be installed from official nodejs website
- 2. Install Ionic. Run npm install -g ionic cordova

# Steps

## Setup Backend

- 1. Go the Github repo that you have downloaded, e.g. <your/path/to/maxisnbiot-hackathon>.
- 2. The example project can be found on folder WebApplication/NodeJS.
- 3. Go to the root folder of example project.
- 4. Run command npm install to install dependencies.
- 5. Create a file call .env at root folder of this project and write following content to the file.

```
AZURE_CONNECTION_STRING=<your azure iot hub connection string>
```

6. Finally, run command npm start to start the server.

## Setup Frontend

- 1. At the same folder WebApplication/NodeJS, go to folder named frontend.
- 2. Run command npm install to install dependencies.
- 3. Run ionic serve. Open browser at http://localhost:8010 to see the result.

4. Run ionic build --prod to build production files, which can be served by express server itself. If server is running, you can browse http://localhost:3000