

Practical No: 1

Aim: Loading Raspbian and Windows IoT Core on Raspberry Pi and executing applications on it using Python and node.js

Set up Raspberry Pi

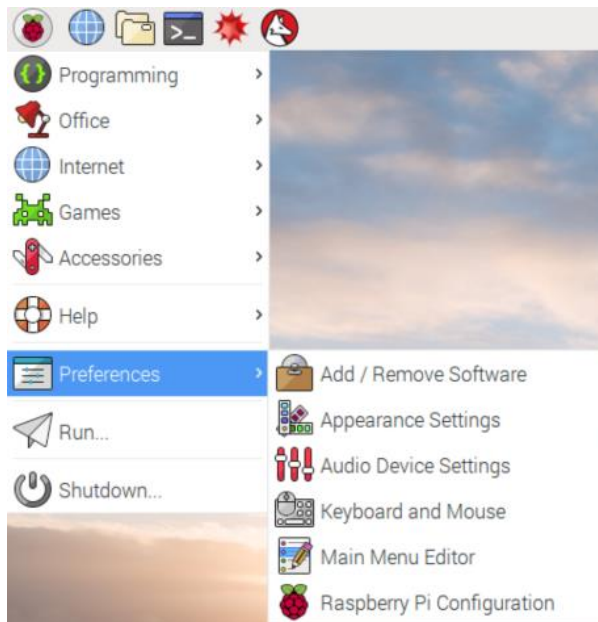
Install the Raspberry Pi OS

Prepare the microSD card for installation of the Raspberry Pi OS image.

1. Download Raspberry Pi OS with desktop.
 - a. Raspberry Pi OS with desktop (the .zip file).
 - b. Extract the Raspberry Pi OS with desktop image to a folder on your computer.
2. Install Raspberry Pi OS with desktop to the microSD card.
 - a. Download and install the Etcher SD card burner utility.
 - b. Run Etcher and select the Raspberry Pi OS with desktop image that you extracted in step 1.
 - c. Select the microSD card drive. Etcher may have already selected the correct drive.
 - d. Click Flash to install Raspberry Pi OS with desktop to the microSD card.
 - e. Remove the microSD card from your computer when installation is complete. It's safe to remove the microSD card directly because Etcher automatically ejects or unmounts the microSD card upon completion.
 - f. Insert the microSD card into Pi.

Enable SSH and I2C

1. Connect Pi to the monitor, keyboard, and mouse.
2. Start Pi and then sign into Raspberry Pi OS by using pi as the user name and raspberry as the password.
3. Click the Raspberry icon > **Preferences** > **Raspberry Pi Configuration**.



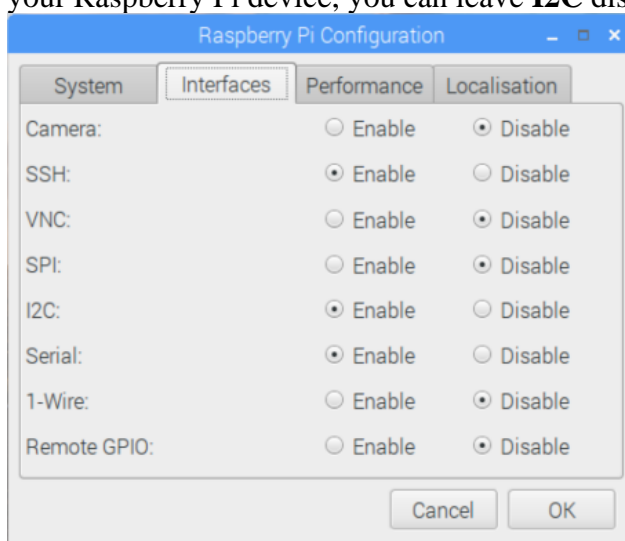
4. On the **Interfaces** tab, set **SSH** and **I2C** to **Enable**, and then click **OK**.

Interface Description

SSH Secure Shell (SSH) is used to remote into the Raspberry Pi with a remote command-line. This is the preferred method for issuing the commands to your Raspberry Pi remotely in this document.

I2C Inter-integrated Circuit (I2C) is a communications protocol used to interface with hardware such as sensors. This interface is required for interfacing with physical sensors in this topic.

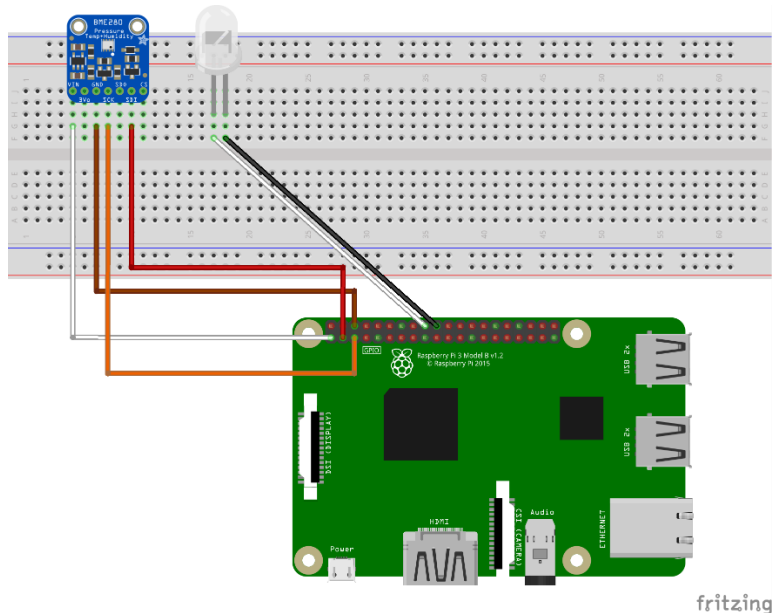
5. If you don't have physical sensors and want to use simulated sensor data from your Raspberry Pi device, you can leave **I2C** disabled.



- 6.

Connect the sensor to Pi

Use the breadboard and jumper wires to connect an LED and a BME280 to Pi as follows. If you don't have the sensor, [skip this section](#).



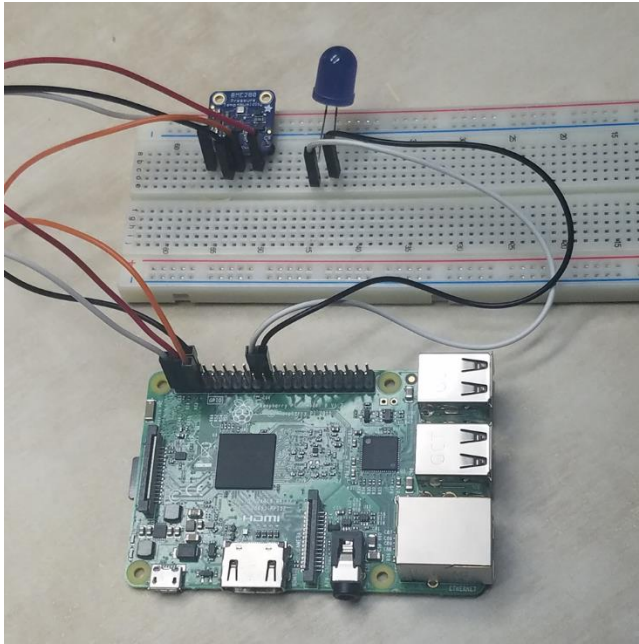
The BME280 sensor can collect temperature and humidity data. The LED blinks when the device sends a message to the cloud.

For sensor pins, use the following wiring:

Start (Sensor & LED)	End (Board)	Cable Color
VDD (Pin 5G)	3.3V PWR (Pin 1)	White cable
GND (Pin 7G)	GND (Pin 6)	Brown cable
SDI (Pin 10G)	I2C1 SDA (Pin 3)	Red cable
SCK (Pin 8G)	I2C1 SCL (Pin 5)	Orange cable
LED VDD (Pin 18F)	GPIO 24 (Pin 18)	White cable
LED GND (Pin 17F)	GND (Pin 20)	Black cable

Click to view [Raspberry Pi 2 & 3 pin mappings](#) for your reference.

After you've successfully connected BME280 to your Raspberry Pi, it should be like below image.



Connect Pi to the network

Turn on Pi by using the micro USB cable and the power supply. Use the Ethernet cable to connect Pi to your wired network or follow the [instructions from the Raspberry Pi Foundation](#) to connect Pi to your wireless network. After your Pi has been successfully connected to the network, you need to take a note of the IP address of your Pi.

Note

Make sure that Pi is connected to the same network as your computer. For example, if your computer is connected to a wireless network while Pi is connected to a wired network, you might not see the IP address in the devdisco output.

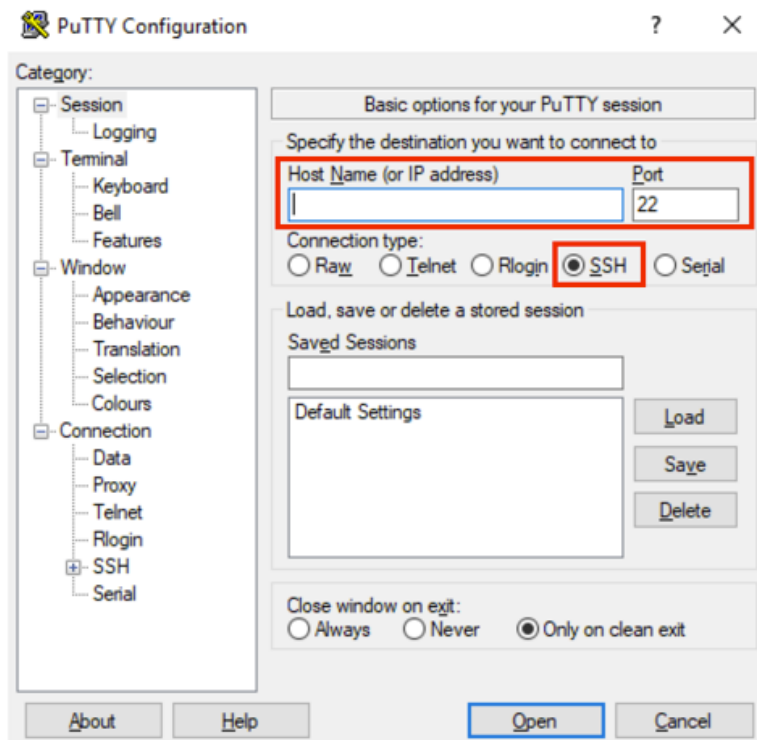
Run a sample application on Pi

Clone sample application and install the prerequisite packages

1. Connect to your Raspberry Pi with one of the following SSH clients from your host computer:

Windows Users

- a. Download and install [PuTTY](#) for Windows.
- b. Copy the IP address of your Pi into the Host name (or IP address) section and select SSH as the connection type.



Mac and Ubuntu Users

Use the built-in SSH client on Ubuntu or macOS. You might need to run `ssh pi@<ip address of pi>` to connect Pi via SSH.

Note

The default username is pi and the password is raspberry.

2. Install Node.js and npm to your Pi.

First check your Node.js version.

BashCopy

```
node -v
```

If the version is lower than 10.x, or if there is no Node.js on your Pi, install the latest version.

BashCopy

```
curl -sSL https://deb.nodesource.com/setup_16.x | sudo -E bash
sudo apt-get -y install nodejs
```

3. Clone the sample application.

BashCopy

```
git clone https://github.com/Azure-Samples/azure-iot-samples-node.git
```

4. Install all packages for the sample. The installation includes Azure IoT device SDK, BME280 Sensor library, and Wiring Pi library.

BashCopy

```
cd azure-iot-samples-node/iot-hub/Tutorials/RaspberryPiApp  
npm install
```

Note

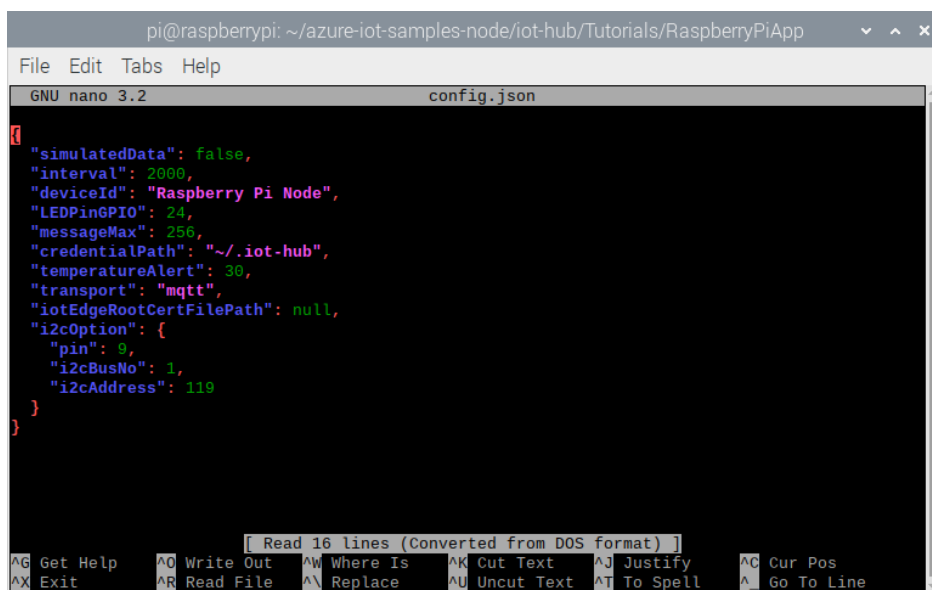
It might take several minutes to finish this installation process depending on your network connection.

Configure the sample application

1. Open the config file by running the following commands:

BashCopy

```
nano config.json
```



```
pi@raspberrypi: ~/azure-iot-samples-node/iot-hub/Tutorials/RaspberryPiApp  
File Edit Tabs Help  
GNU nano 3.2 config.json  
{  
  "simulatedData": false,  
  "interval": 2000,  
  "deviceId": "Raspberry Pi Node",  
  "LEDPinGPIO": 24,  
  "messageMax": 256,  
  "credentialPath": "~/.iot-hub",  
  "temperatureAlert": 30,  
  "transport": "mqtt",  
  "iotEdgeRootCertFilePath": null,  
  "i2cOption": {  
    "pin": 0,  
    "i2cBusNo": 1,  
    "i2cAddress": 119  
  }  
}  
[ Read 16 lines (Converted from DOS format) ]  
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos  
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line
```

There are two items in this file you can configure. The first one is interval, which defines the time interval (in milliseconds) between messages sent to the cloud. The second one is simulatedData, which is a Boolean value for whether to use simulated sensor data or not.

If you **don't have the sensor**, set the simulatedData value to true to make the sample application create and use simulated sensor data.

Note: The i2c address used in this tutorial is 0x77 by default. Depending on your configuration it might also be 0x76: if you encounter an i2c error, try to change the value to 118 and see if that works better. To see what address is used by your sensor, run `sudo i2cdetect -y 1` in a shell on the raspberry pi

2. Save and exit by typing Control-O > Enter > Control-X.

Run the sample application

Run the sample application by running the following command:

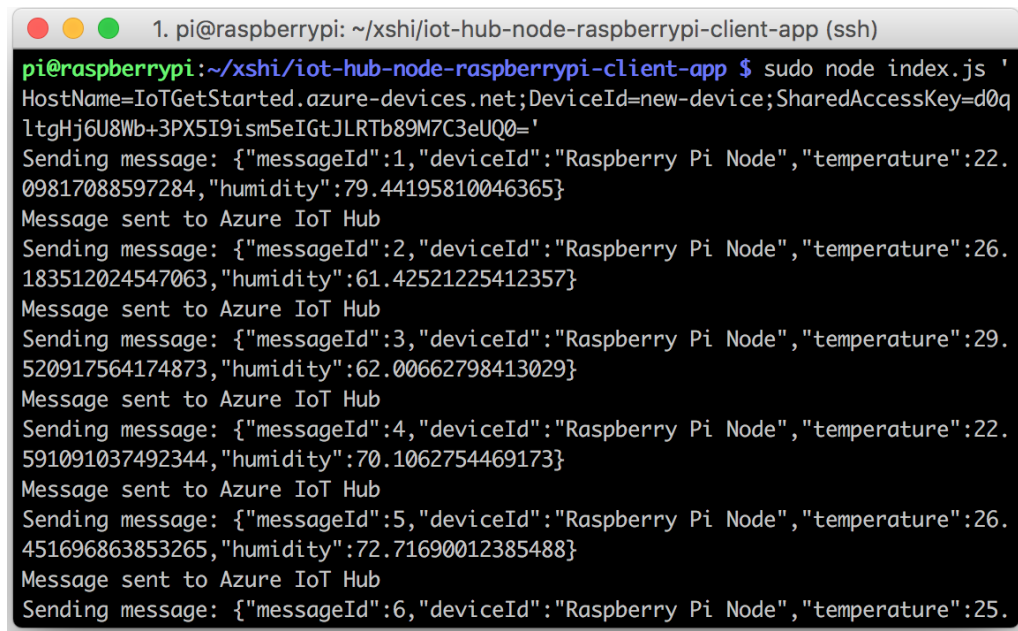
BashCopy

```
sudo node index.js '<YOUR AZURE IOT HUB DEVICE CONNECTION STRING>'
```

Note

Make sure you copy-paste the device connection string into the single quotes.

You should see the following output that shows the sensor data and the messages that are sent to your IoT hub.



```
1. pi@raspberrypi: ~/xshi/iot-hub-node-raspberrypi-client-app (ssh)
pi@raspberrypi:~/xshi/iot-hub-node-raspberrypi-client-app $ sudo node index.js '
HostName=IoTGetStarted.azure-devices.net;DeviceId=new-device;SharedAccessKey=d0q
ltgHj6U8Wb+3PX5I9ism5eIGtJLRTb89M7C3eUQ0='
Sending message: {"messageId":1,"deviceId":"Raspberry Pi Node","temperature":22.
09817088597284,"humidity":79.44195810046365}
Message sent to Azure IoT Hub
Sending message: {"messageId":2,"deviceId":"Raspberry Pi Node","temperature":26.
183512024547063,"humidity":61.42521225412357}
Message sent to Azure IoT Hub
Sending message: {"messageId":3,"deviceId":"Raspberry Pi Node","temperature":29.
520917564174873,"humidity":62.00662798413029}
Message sent to Azure IoT Hub
Sending message: {"messageId":4,"deviceId":"Raspberry Pi Node","temperature":22.
591091037492344,"humidity":70.1062754469173}
Message sent to Azure IoT Hub
Sending message: {"messageId":5,"deviceId":"Raspberry Pi Node","temperature":26.
451696863853265,"humidity":72.71690012385488}
Message sent to Azure IoT Hub
Sending message: {"messageId":6,"deviceId":"Raspberry Pi Node","temperature":25.
```

Read the messages received by your hub

One way to monitor messages received by your IoT hub from your device is to use the Azure IoT Hub extension for Visual Studio Code. To learn more, see [Use the Azure IoT Hub extension for Visual Studio Code to send and receive messages between your device and IoT Hub.](#)

For more ways to process data sent by your device, continue on to the next section.

Practical No: 2

Aim: Create a home automation system and control the devices remotely

Introduction

In this project, we will be making an **Arduino Home automation system using a TV remote**. This system will be able to control various appliances in your home, such as lights, fans, and TVs. You will be able to turn these appliances on or off using the TV remote. This project is perfect for those who want to have a simple and convenient way to control their home appliances. check out [GSM Based Home Automation System Using Arduino](#).

Material Used

Components

1. Arduino Nano ————— <https://amzn.to/3PoC5bg>
2. IR Sensor (TSOP1738) ————— <https://amzn.to/3sCiniz>
3. 4-Channel Relay module ————— <https://amzn.to/3yE1vMd>
4. IR Remote ————— <https://amzn.to/38xDiwp>
5. 12v 1A SMPS ————— <https://amzn.to/3yHj8e7>

What is a home automation system?

Home automation systems are designed to automate various tasks in the home. These systems can be used to control lighting, appliances, security systems, and other devices in the home. Home automation systems can be controlled via a mobile app, web browser, or voice control. You can use a home automation system to make your life easier and save energy.

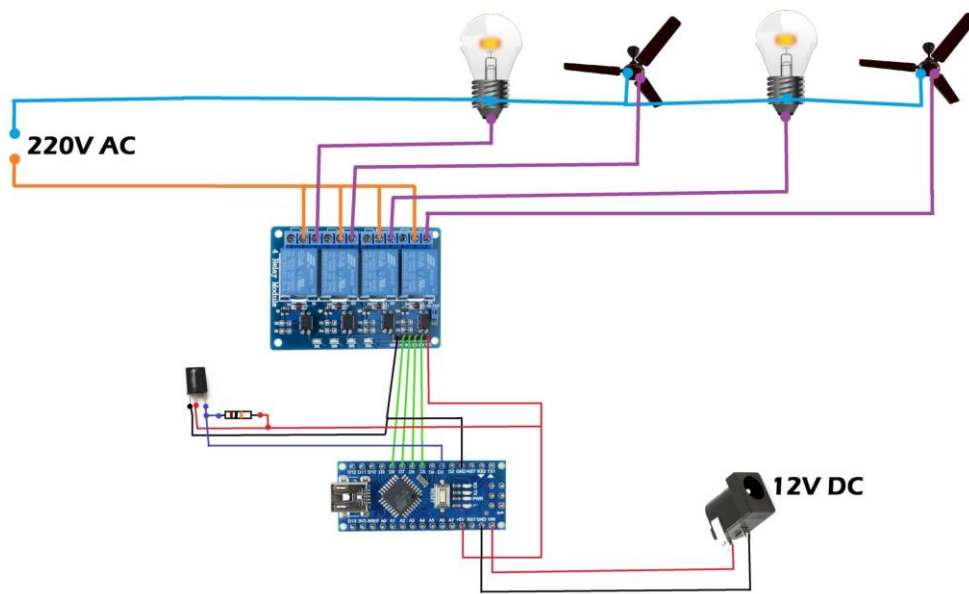
How can a home automation system using a TV remote be beneficial?

A home automation system can be beneficial because it can be used to control a variety of devices in the home, including the TV. be useful In this era of technology, a home automation system using a TV remote can be useful in various ways.

How to set up a home automation system using a TV remote.?

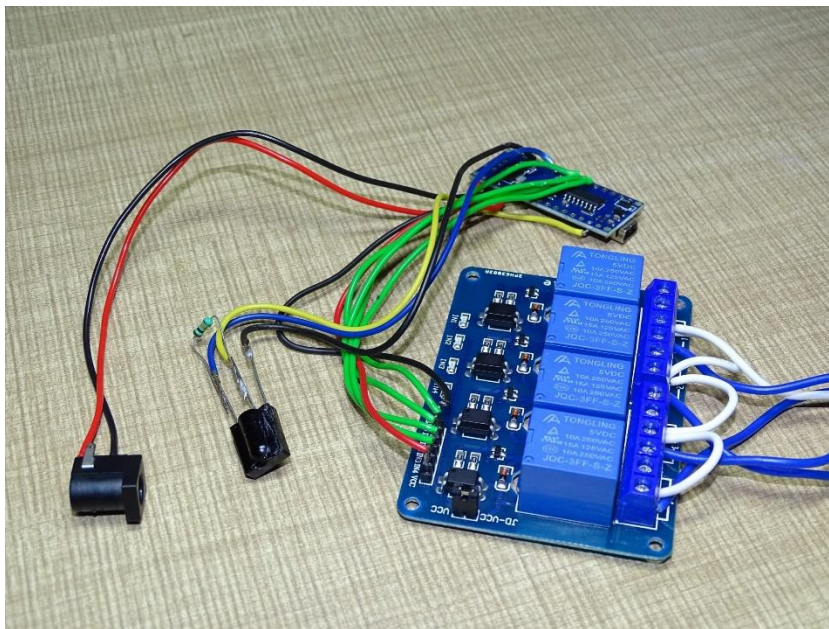
In this guide, we'll guide you on how to set up an Arduino home automation system using a TV remote. You'll need a TV remote and an Arduino controller. We'll assume you have a basic understanding of home automation.

Schematic Diagram



Prepare Circuit diagram

Make the circuit by following the schematic diagram shown in the above photo. after all, connections secure the whole circuit in upload code. now it is ready to use.



Source Code

```
//for 4channel IR Remote Controoler  
#include <IRremote.h>
```

```
int RECV_PIN = 3 ;  
int L1 = 5;
```

```
int L2 = 6;
int L3 = 7;
int L4 = 8;
```

```
int V1=0;
int V2=0;
int V3=0;
int V4=0;
int V5=0;
```

```
long Vc1= 2534850111; //put your button code here. Example "2534850111, 1033561079,
1635910171, 3855596927"
long Vc2= 1033561079; //put your button code here
long Vc3= 1635910171; //put your button code here
long Vc4= 2351064443; //put your button code here
long Vc5= 1217346747; //put your button code here
```

```
IRrecv irrecv(RECV_PIN);
```

```
decode_results results;
```

```
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver E318261B

  pinMode(L1,OUTPUT);
  pinMode(L2,OUTPUT);
  pinMode(L3,OUTPUT);
  pinMode(L4,OUTPUT);

  digitalWrite(L1,HIGH);
  digitalWrite(L2,HIGH);
  digitalWrite(L3,HIGH);
  digitalWrite(L4,HIGH);
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value);
    irrecv.resume(); // Receive the next value 3810010651

  }
  if((results.value== Vc1) && V1==0)
  {digitalWrite(L1,LOW);
    V1=1;
    results.value=0;
    delay(50);
  }
}
```

```

if((results.value== Vc1) && V1==1)
{digitalWrite(L1,HIGH);
  V1=0;
  results.value=0;
  delay(50);
}
if((results.value== Vc2) && V2==0)
{digitalWrite(L2,LOW);
  V2=1;
  results.value=0;
  delay(50);
}
if((results.value== Vc2) && V2==1)
{digitalWrite(L2,HIGH);
  V2=0;
  results.value=0;
  delay(50);
}
if((results.value== Vc3) && V3==0)
{digitalWrite(L3,LOW);
  V3=1;
  results.value=0;
  delay(50);
}
if((results.value== Vc3) && V3==1)
{digitalWrite(L3,HIGH);
  V3=0;
  results.value=0;
  delay(50);
}
if((results.value== Vc4) && V4==0)
{digitalWrite(L4,LOW);
  V4=1;
  results.value=0;
  delay(50);
}
if((results.value== Vc4) && V4==1)
{digitalWrite(L4,HIGH);
  V4=0;
  results.value=0;
  delay(50);
}

if((results.value== Vc5) && V5==0)
{
  digitalWrite(L1,LOW);
  digitalWrite(L2,LOW);
  digitalWrite(L3,LOW);
  digitalWrite(L4,LOW);
  V1=1;

```

```

V2=1;
V3=1;
V4=1;
V5=1;

results.value=0;
delay(50);
}
if((results.value== Vc5) && V5==1)
{ digitalWrite(L1,HIGH);
digitalWrite(L2,HIGH);
digitalWrite(L3,HIGH);
digitalWrite(L4,HIGH);
V1=0;
V2=0;
V3=0;
V4=0;
V5=0;
results.value=0;
delay(50);
}

delay(100);
}

```

Arduino home automation Conclusion

Control After extensive testing and research, it can be concluded that the Arduino home automation system using TV remote control is reliable, efficient, and easy to use. It is an affordable solution for those who want to automate their homes.

Practical No: 3

Aim: Create the programs using the Microsoft Cognitive APIs for IoT.

Introduction

Azure Cognitive Services is a collection of APIs that make your applications smarter. Some of those APIs are listed below:

- Vision: image classification, face detection (including emotions), OCR
- Language: text analytics (e.g. key phrase or sentiment analysis), language detection and translation

To use one of the APIs you need to provision it in an Azure subscription. After provisioning, you will get an endpoint and API key. Every time you want to classify an image or detect sentiment in a piece of text, you will need to post an appropriate payload to the cloud endpoint and pass along the API key as well.

What if you want to use these services but you do not want to pass your payload to a cloud endpoint for compliance or latency reasons? In that case, the Cognitive Services containers can be used. In this post, we will take a look at the Text Analytics containers, specifically the one for Sentiment Analysis. Instead of deploying the container manually, we will deploy the container with IoT Edge.

IoT Edge Configuration

To get started, create an IoT Hub. The free tier will do just fine. When the IoT Hub is created, create an IoT Edge device. Next, configure your actual edge device to connect to IoT Hub with the connection string of the device you created in IoT Hub. Microsoft have a great tutorial to do all of the above, using a virtual machine in Azure as the edge device. The tutorial I linked to is the one for an edge device running Linux. When finished, the device should report its status to IoT Hub:

gebaioot - IoT Edge

Search (Ctrl+/)

Explorers

- Query explorer
- IoT devices

Automatic Device Management

- IoT Edge
- IoT device configuration

Messaging

- File upload
- Message routing

Resiliency

- Manual failover (preview)

Monitoring

IoT Edge devices

Deploy Azure services and solution-specific code to on-premises devices. Use IoT Edge devices to perform compute and analytics tasks on data before it's sent to the cloud.

IoT Edge devices | IoT Edge deployments

Field: Select or enter your own | Operator: = | Value:

+ Add new clause

Query devices

Switch to query editor

DEVICE ID	RUNTIME RESPONSE	IOT EDGE MODULE COUNT	CONNECTED CLIENT COUNT	DEPLOYMENT COUNT
ubuntu	OK	3	1	0

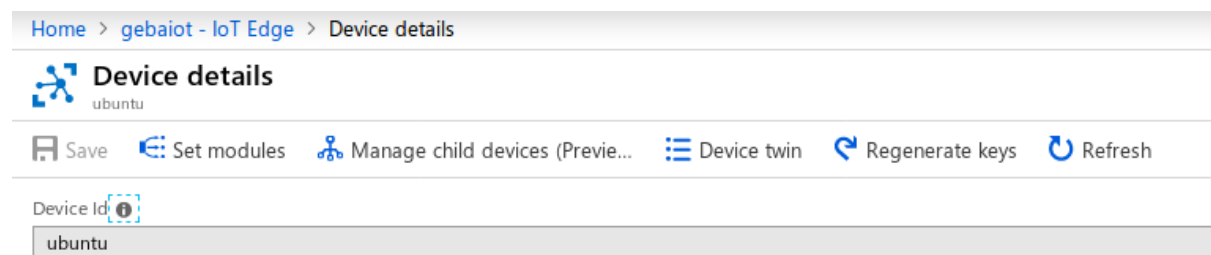
If you want to install IoT Edge on an existing device like a laptop, follow [the procedure for Linux x64](#).

Once you have your edge device up and running, you can use the following command to obtain the status of your edge device: **sudo systemctl status iotedge**. The result:

```
geert@geert-HP:~$ sudo systemctl status iotedge
[sudo] password for geert:
● iotedge.service - Azure IoT Edge daemon
   Loaded: loaded (/lib/systemd/system/iotedge.service; enabled; vendor preset:
   Active: active (running) since Mit 2018-12-26 18:36:29 CET; 22h ago
     Docs: man:iotedged(8)
  Main PID: 2530 (iotedged)
    Tasks: 11
   Memory: 20.2M
      CPU: 38.047s
   CGroup: /system.slice/iotedge.service
           └─2530 /usr/bin/iotedged -c /etc/iotedge/config.yaml
```

Deploy Sentiment Analysis container

With the IoT Edge daemon up and running, we can deploy the Sentiment Analysis container. In IoT Hub, select your IoT Edge device and select **Set modules**:



In **Set Modules** you have the ability to configure the modules for this specific device. Modules are always deployed as containers and they do not have to be specifically designed or developed for use with IoT Edge. In the three step wizard, add the Sentiment Analysis container in the first step. Click **Add** and then select **IoT Edge Module**. Provide the following settings:


Although the container can freely be pulled from the Image URI, the container needs to be configured with billing info and an API key. In the **Billing** environment variable, specify the endpoint URL for the API you configured in the cloud. In **ApiKey** set your API key. Note that the container always needs to be connected to the cloud to verify that you are allowed to use the service. Remember that although your payload is not sent to the cloud, your container usage is. The full container create options are listed below:

IoT Edge Custom Modules


TextAnalytics

* Name


TextAnalytics

* Image URI 


mcr.microsoft.com/azure-cognitive-services/sentiment

Container Create Options 

```
{
  "Env": [
    "Eula=accept",
    "Billing=https://westeurope.api.cogniti
    "ApiKey=aaaf2a5edee24706b0dc4bc06e84dca
  ],
  "HostConfig": {
```

Restart Policy 

always

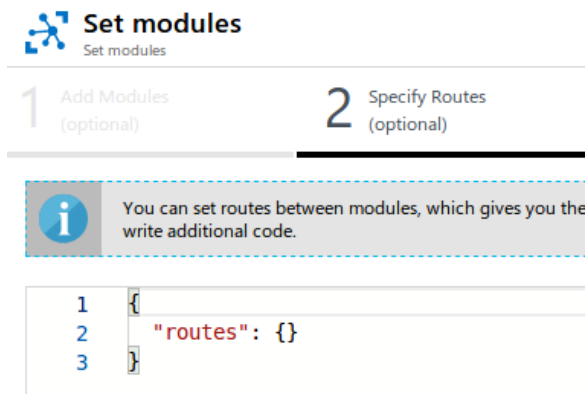
Desired Status 

running

```
{
  "Env": [
    "Eula=accept",
    "Billing=https://westeurope.api.cognitive.microsoft.com/text/analytics/v2.0",
    "ApiKey=<yourKEY>"
  ],
  "HostConfig": {
    "PortBindings": {
      "5000/tcp": [
        {
          "HostPort": "5000"
        }
      ]
    }
  }
}
```

In **HostConfig** we ask the container runtime (Docker) to map port 5000 of the container to port 5000 of the host. You can specify other create options as well. On the next page, you can configure routing between IoT Edge modules. Because we do not use actual IoT Edge modules, leave the configuration as shown below:

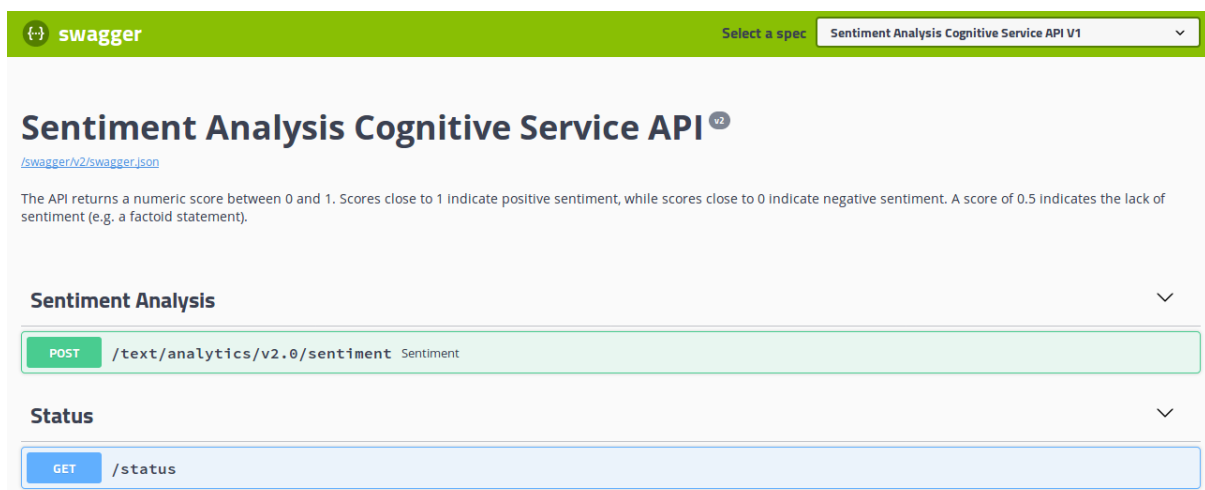
Now move to the last page in the **Set Modules** wizard to review the configuration and click **Submit**.



Give the deployment some time to finish. After a while, check your edge device with the following command: **sudo iotedge list**. Your TextAnalytics container should be listed. Alternatively, use **sudo docker ps** to list the Docker containers on your edge device.

Testing the Sentiment Analysis container

If everything went well, you should be able to go to <http://localhost:5000/swagger> to see the available endpoints. Open **Sentiment Analysis** to try out a sample:



You can use curl to test as well:

```
curl -X POST "http://localhost:5000/text/analytics/v2.0/sentiment" -H "accept: application/json" -H "Content-Type: application/json-patch+json" -d "{ \"documents\": [ { \"language\": \"en\", \"id\": \"1\", \"text\": \"I really really despise this product!! DO NOT BUY!!\" } ] }"
```

As you can see, the API expects a JSON payload with a documents array. Each document object has three fields: language, id and text. When you run the above command, the result is:


```
{"documents":[{"id":"1","score":0.0001703798770904541}],"errors":[]}
```

In this case, the text **I really really despise this product!! DO NOT BUY!!** clearly results in a very bad score. As you might have guessed, 0 is the absolute worst and 1 is the absolute best.

Just for fun, I created a small Go program to test the API:

```
geert@geert-HP:~/go/src/github.com/gbaeke/ta$ go run main.go --text "I really enjoyed this blog post!"
Getting sentiment with Cognitive Services
=====
Language set to: en
Text set to: I really enjoyed this blog post!
Sentiment score is 0.953618
```

The Go program can be found here: <https://github.com/gbaeke/sentiment>. You can download the executable for Linux with: **wget <https://github.com/gbaeke/sentiment/releases/download/v0.1/ta>**. Make ta executable and use **./ta --help** for help with the parameters.

Summary

IoT Edge is a great way to deploy containers to edge devices running Linux or Windows. Besides deploying actual IoT Edge modules, you can deploy any container you want. In this post, we deployed a Cognitive Services container that does Sentiment Analysis at the edge.

Practical No: 4

Aim: Create blockchain on Raspberry Pi and implement and test it. Authenticate IoT with blockchain

This series of tutorials will describe how to set up a private Ethereum blockchain that will be composed of a computer (miner) and one or several Raspberry PI 3 devices (nodes).

The objective is to build a development environment to learn about blockchain principles, to develop and test your own smart contracts before releasing them to the live chain (mainnet).

To make this project fun, we will integrate a RPi with a Smart Contract deployed on a private chain. The Smart Contract will be used to check if a user has enough tokens to use a service. The RPi will be used to visualize the status of the contract.

We will use the following naming conventions to identify the devices on which the commands are entered:

computer\$ A computer requires to perform initial configuration

pi\$ Shell of the node running on the Raspberry PI device

Disclaimer

This series of tutorials doesn't pretend to be comprehensive neither to give you a full introduction to Ethereum.

There are many ways to set up a private Ethereum blockchain from the most simplistic to the most complicated.

Here are some other examples:

- Microsoft Azure and RPi ([here](#))
- Docker image ([here](#))
- Pre-defined images such with EthRaspian ([here](#))

Part 1: Install an Ethereum node on a Raspberry PI (RPi)

In this first part, we will describe the steps to transform your Raspberry PI 3 (RPi) into an Ethereum node.

The RPi is intended to act as a node connected to our private Ethereum blockchain. This means that it's not required to install a full flesh operating system. A minimal image of Raspbian is good enough.

For information, there are pre-bundled RPi images that make it possible for you to transform your RPi into an Ethereum node.

One of these images is provided by the project [EthRaspbian](#).

Here, we chose to build our Ethereum node without these prepared images. The objective is to show you all the required steps to do it from scratch.

Due to hardware limitations (CPU, memory), your RPi will not be able to mine ethers or to build new blocks of transactions.

Step 1 – Check your configuration

We will use a Raspberry PI 3 with a microSD of 16 Gb and a computer to prepare the microSD card.

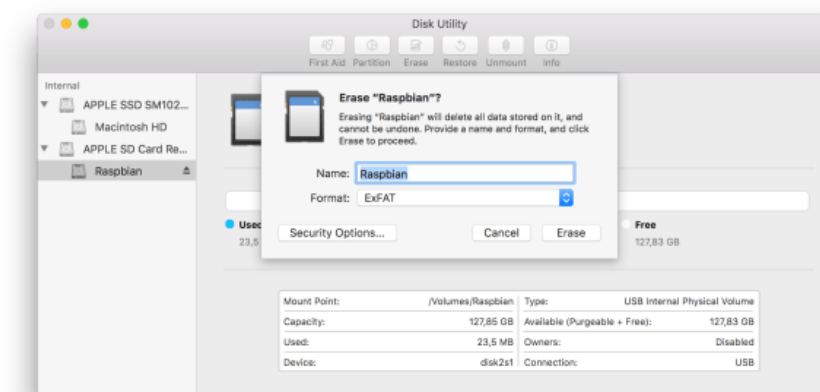
The setup of the RPi will require the following hardware:

- A Raspberry Pi
- An SD Card with at least 16Gb
- A LAN cable
- A keyboard
- A display

Step 2 – Format your SD Card

Insert your SD card on your computer and format it as FAT32.

On macOS, use the format **ExFat** in Disk Utility:



Step 3 – Write the Raspbian image

From your computer, download the Raspbian Jessie Lite image (minimal image) available [here](#).

Unzip the file.

Write the image on the SD card by following instructions [there](#). The process varies slightly depending on whether you are on a [MacOS](#), [Linux](#) or [Windows](#) machine.

Here is an example for macOS (be very careful with the commands, every character matters):

```
computer$ diskutil list
```

...

```
/dev/disk5 (external, physical):
```

...

```
computer$ diskutil unmountDisk /dev/disk5
```

```
computer$ sudo dd bs=1m if=2016-11-25-raspbian-jessie-lite.img of=/dev/rdisk5
```

```
computer$ diskutil eject /dev/disk5
```

Ensure that your RPi is powered off.

Insert the SD card on you RPi and power on the device. Your RPi will boot the image.

Step 4 – Configure your RPi

Log in the RPi with the default credential is:

- username: **pi**
- password: **raspberry**

The RPi needs to be configured:

- Change the **keyboard layout** (if required)
- Set the **timezone**: required to allow a blockchain synchronisation between nodes
- Enable **SSH**: securely access your RPi from your computer
- Change your **hostname**: easily identified your RPi within your network (example: node1, node2, etc.)

We use **raspi-config** to setup all these settings:

```
pi$ sudo raspi-config
```

1) **Timezone** can be changed from the option "**Internationalisation Options**" and "Change Timezone".

2) **Keyboard layout** can be changed from the option "**Internationalisation Options**" and "Change Keyboard Layout".

3) **SSH** can be enabled from the option "**Advanced Options**" then "SSH".

4) The **hostname** can be changed from the option "**Advanced Options**" and then "Hostname. In our tutorial, we use the hostname "**ethpinode1**"

Press Finish and reboot your RPi:

```
p$> sudo reboot
```

Finally, you should really change the default password of the pi account:

```
pi$ passwd
```

Step 5 – Setup Wi-Fi

To setup your Wi-Fi interface, proceed as described below:

```
pi$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

1. Change the country code to set yours
2. Add the following lines at the end of the file and replace the ssid with the name of your WiFi and psk with your WiFi's password:
network={

ssid="enter the SSID of your Wi-Fi network"

psk="enter the password of your Wi-Fi network"

}

To apply your changes, restart your RPi or your Wi-Fi interface:

```
pi$ sudo ifdown wlan0
```

```
pi$ sudo ifup wlan0
```

You can now unplug your LAN cable.

Step 6 – Connect to your RPi

To connect remotely to your RPi through SSH, you have to know the IP address of your RPi device.

You can find it by typing the following command from your RPi:

```
pi$ ifconfig
```

Or you can search it from your LAN by filtering IP addresses from a MAC prefix address

(*b8:27:eb* is the MAC prefix address of RPi devices):

```
computer$ arp -na | grep -i b8:27:eb
```

```
? (192.168.1.31) at b8:27:eb:1c:aa:94 on en0 ifscope [ethernet]
```

If you have several RPi, you can locate them using the “**host**” command:

```
computer$ host 192.168.1.31
```

```
31.1.168.192.in-addr.arpa domain name pointer ethpinode1.
```

At this stage, you can continue the setup of your RPi remotely from your computer using the “**ssh**” (“**pi**” is the username defined on your RPi):

```
computer$ ssh pi@192.168.1.31
```

It's time to install Ethereum.

Step 7 – Install Geth

In this step, we are going to install the Go implementation of Ethereum client called **Geth**.

Connect to your RPi and retrieve the type of your CPU:

```
pi$ cat /proc/cpuinfo
```

...

```
model name : ARMv7 Processor rev 4 (v7l)
```

...

Go to the download page of Go-Ethereum by clicking [here](#).

Select the tab **Linux** from the stable releases page.

Locate the row related to your CPU model and **copy the link address** . It's recommended to retrieve the latest version of Geth.

In our example, we use Geth 1.5.7.

Back to your RPi, retrieve the bundle and extract the archive:

```
pi$ wget https://gethstore.blob.core.windows.net/builds/geth-linux-arm7-1.5.7-da2a22c3.tar.gz
```



```
pi$ tar zxvf geth-linux-arm7-1.5.7-da2a22c3.tar.gz
```

Copy the Geth application to the **/usr/local/bin** folder:

```
pi$ cd geth-linux-arm7-1.5.7-da2a22c3
```

```
pi$ sudo cp geth /usr/local/bin
```

You can remove the archive and the uncompressed folder.

Check the version of Geth:

```
pi$ geth version
```

Geth

Version: **1.5.7-stable**

Git Commit: da2a22c384a9b621ec853fe4b1aa651d606cf42b

Protocol Versions: [63 62]

Network Id: 1

Go Version: go1.7.4

OS: linux

GOPATH=

GOROOT=/usr/local/go

The same process can be applied to update an existing version of Geth.

Now, Ethereum is installed.

Let's start it!

Step 8 – Run Geth

Start Geth using the following command:

```
pi$ geth
```

...

```
I1217 19:11:27.420830 p2p/server.go:342] Starting Server
```

...

```
I1217 19:11:39.816110 eth/downloader/downloader.go:326] Block synchronisation started
```

Press CTRL-C to stop the Ethereum server.

Summary:

At this stage, Ethereum is installed on your RPi and able to synchronise with the live chain (mainnet).

The blockchain data (**chaindata**) is located right here: **~/.ethereum/chaindata**

Your accounts will be stored in a **wallet** located in this folder: **~/.ethereum/keystore**

Congratulations! Your RPi is an Ethereum node.

You can repeat these steps to setup additional RPi.

Practical No: 5

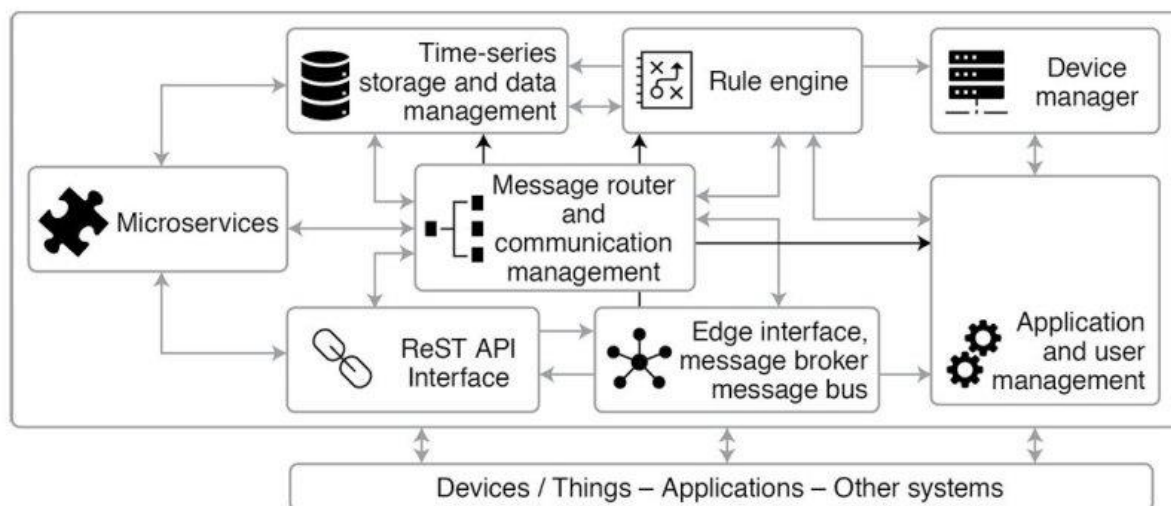
Aim:. Implement Microservices on IoT device.

Practical No: 6

Aim: Build your own IoT platform.

FOUR KEY COMPONENTS

So, what are the minimum requirements that an IoT platform should include? Well, just have a look at the **Figure 1**, which lists all the main required component blocks. We will now discuss the use of the main components in brief and how we can get the most out of our platform with these. All these are hosted on a virtual cloud machine or VPS (virtual private server). We can use any operating system (OS) but using a Linux-based OS is always preferred. We will be using Ubuntu Linux for this article because it provides better performance, stricter security controls and a better ease of access to our platform. Now, let's look at the four key component blocks.



CLOUD INSTANCE INITIALIZATION

This is the first step toward building our platform. We will require a space on the Internet where our platform will be hosted so that we can access it from anywhere. To do this, we will set up a cloud instance on a very popular site called DigitalOcean. It is an infrastructure-as-a-service (IAAS) that lets you host virtual servers and a lot more.

To get started, go to the website www.digitalocean.com and register. After that, you will need to add some credit card credentials for payment. There are a number of referral links that will provide you with free credits, so I recommend you do a search for them. After everything is done, you will see the webpage shown in **Figure 2**. Now, we will create our virtual server, which is called a Droplet.

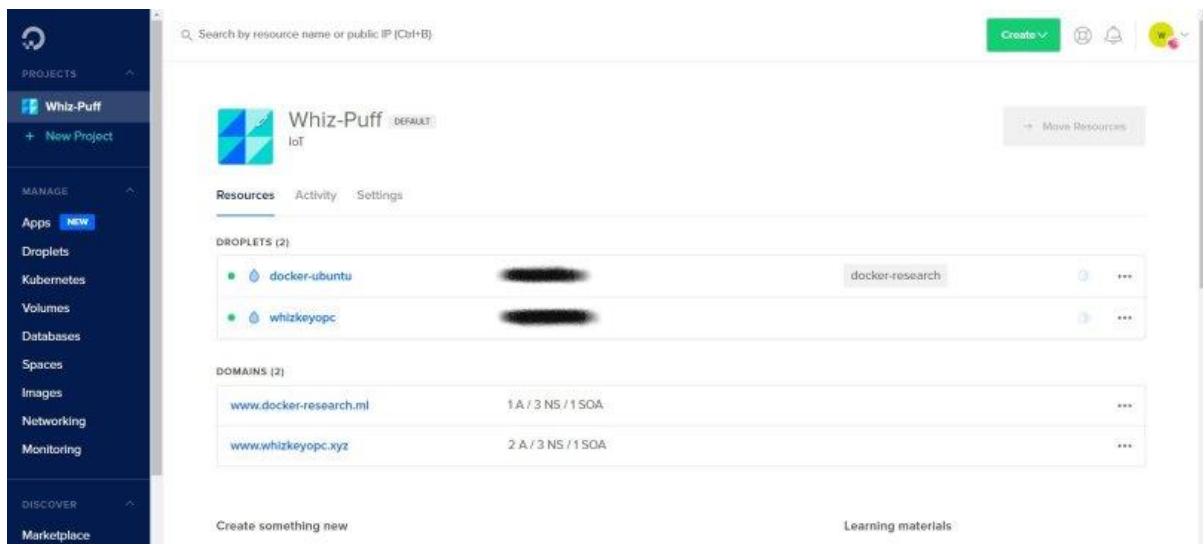


FIGURE 2 – DigitalOcean Dashboard. You will arrive on this page once you’ve created your account and given your payment details.

Click the blue button on the screen saying “*get started with a Droplet*” and it will redirect you to a new webpage where you need to specify some things with regard to your cloud instance. The first thing it asks is which OS to choose. DigitalOcean only supports Linux-based OSes out of the box but you can install custom images too. For now, we will stick with the popular Ubuntu OS because it has the biggest community. Next, you’ll need to select the plan you want to use. There are a number of plans available and they are divided into four main categories:

Standard plans: These are the cheapest and are for beginners. They provide a shared CPU space and start from as low as \$5 per month.

General purpose plans: These are for people who want to host a medium-scale website for their business. They have dedicated CPUs and start at \$40 per month.

CPU optimized plans: These plans are for those who do not require much RAM but they do want more CPU processing power.

Memory optimized plans: These plans are for those who need to perform memory intensive tasks like advanced data analytics.

For this article, we will go with either the \$5 or \$10 per month standard plan. Please note that if you are using the credit, you might want to choose a higher plan because its validity period is only two months. I am going with the \$10 plan, which provides me with 2GB of RAM, one shared CPU, 50GB SSD storage space (more can be easily added) and a 2TB data transfer limit. That transfer limit is truly great considering that the same data transfer limit on a platform like AWS (Amazon Web Services) will cost you around \$100.

Next, we will select the datacenter region for our platform. We can actually select any of the given choices, but if our platform is going to be used by everyone all over the world (which is the case we hope!). I would suggest you choose a US-based Datacenter region. In my personal experience they provide the best response times. Next, there are some additional options we need to fill in. And if you have a SSH key, which enables secure SSH access for you, you can just enter the public key you created and you are good to go! I would suggest using PuTTY and PuTTYgen for this. I have provided a link [1] in **RESOURCES** at the end of this article to a separate tutorial that will describe how to do the SSH key procedure. For this tutorial, we will directly access the platform through the public IP and login with our credentials.

After everything is done, just press *Create* after finalizing a host name for your Droplet. After that, wait for a few minutes as your server is created. Finally, if everything goes right, you will have a Droplet listed on your dashboard as shown in **Figure 3**. You can even see the Public IP Address assigned to your cloud instance on the dashboard. Now that it is up and running, we can connect to it via SSH! For Mac, you can use your terminal and type:

```
# ssh root@<Your Instance IP>
```

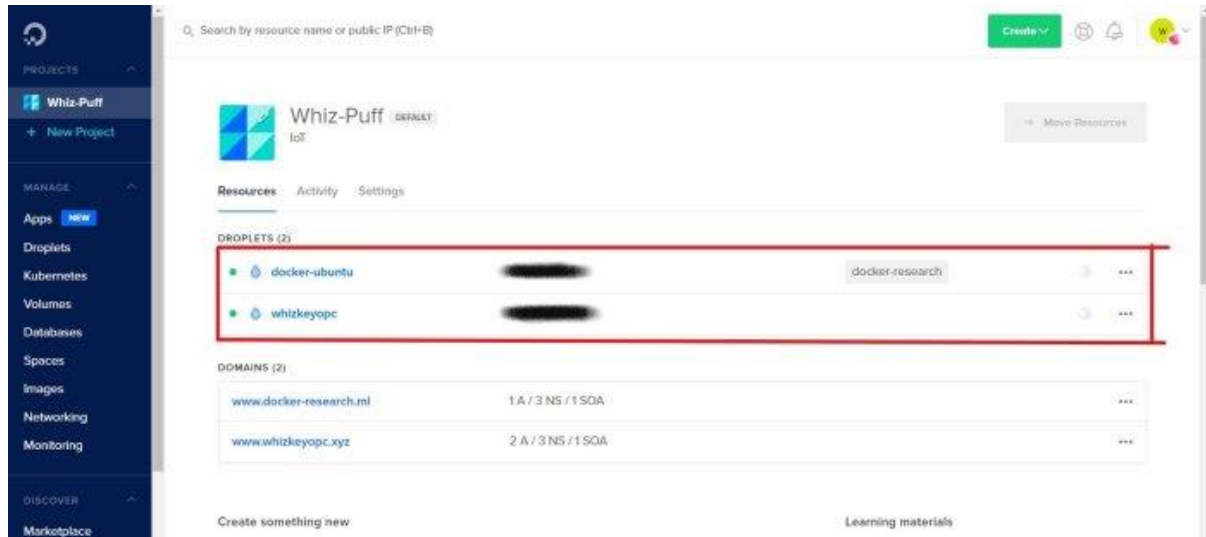


FIGURE 3 – This figure shows where your Droplets will be accessible once you create one using the required information.

For Windows, you can use a software like PuTTY, which is what I am doing. Just open the software and choose the connection type to SSH and enter your host IP. After that is done, just press the *Open* button. It will ask for login credentials. The default username is *root* and a temporary password is sent to you by email. Just enter that and if it's right, the system will prompt you to set up a new password of your choice. Do that and finally you have access to the command line of your Ubuntu Sever! We can also create floating IPs for our platform if required right from the DigitalOcean control panel.

We will skip that for now because it is not our primary requirement. Next, we will install the required software stacks to our servers. For our use case, we need the “LAMP” stack. LAMP is an acronym that stands for Linux-Apache-MySQL-PHP. As such, LAMP stacks typically consist of the Linux OS, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language. We will start with Apache Server.

APACHE SERVER

Apache is a free and open-source cross platform web server software. It is one of the most popular and widely used server software in the world. To install it on our virtual server, we only need to type the following two commands:

```
# apt update
# apt install apache2
```

We use the Ubuntu's default package manager to do this task. You will be prompted to allow the use of extra disk space. Keep pressing *Y* and *Enter* to continue until the installation is complete.

After it is done, you have to allow the web server to bypass the firewall if it is available. For that, just issue the following commands:

```
# ufw app list
# sudo ufw allow "Apache Full"
```

Now that this is done, you can navigate to your Virtual Server's IP address and you should see the Apache's default page as shown in **Figure 4**.

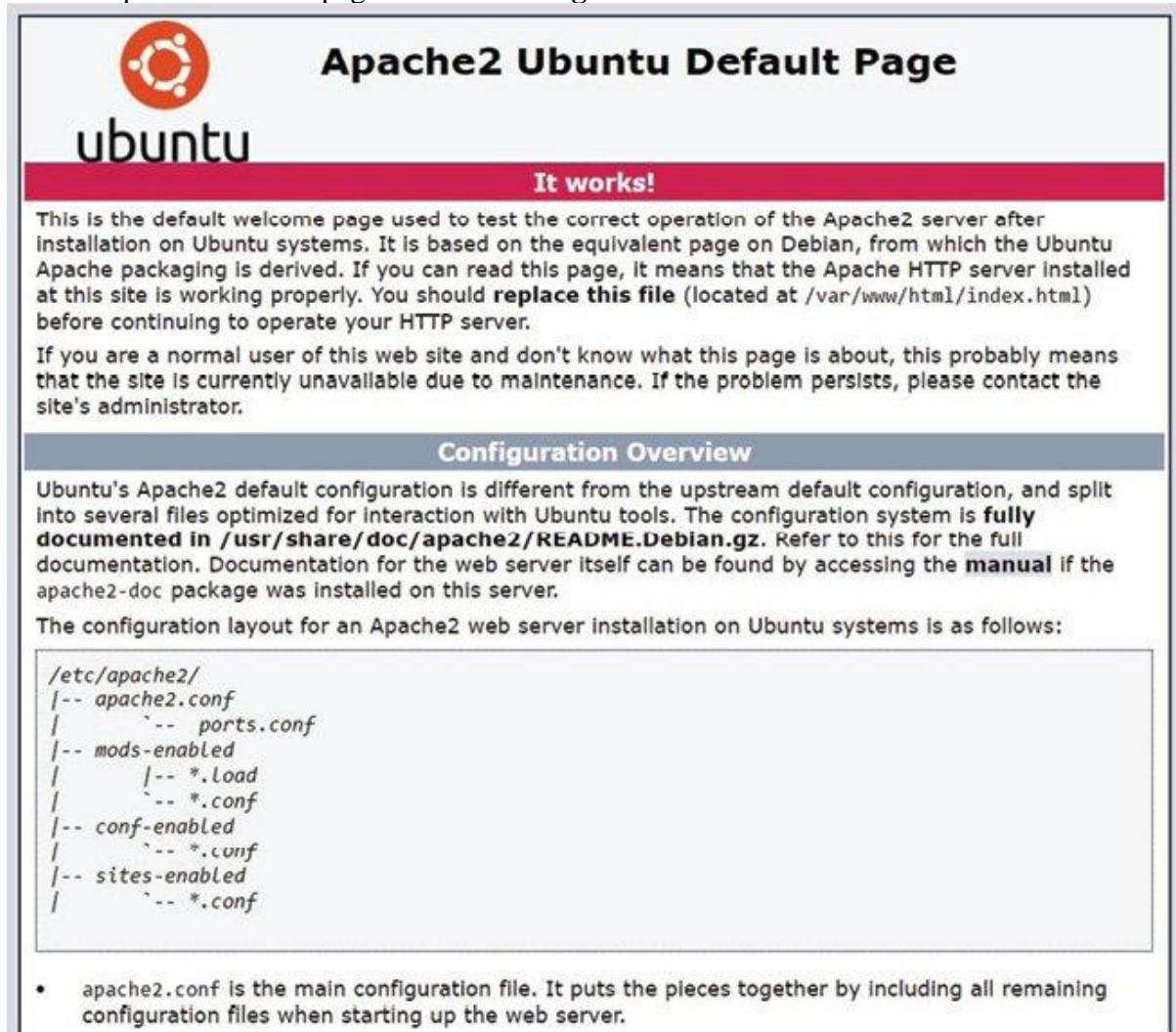


FIGURE 4 – This is default Apache Server page which will be accessible on your Droplet's IP address once you install Apache Server to it.

MYSQL DATABASE

We will use MySQL as the database management system for our IoT Platform. It is very easy to install MySQL to our system using a few commands:

```
# apt install mysql-server
```

This will also prompt for the extra disk space so just press *Y* and *Enter* and let the installation complete. Now, you need to go through a basic setup process to complete the installation. For that, issue the following command:

```
# mysql_secure_installation
```

This asks if we want to configure the validate password plugin. We will select *Y* for yes and continue providing additional passwords as prompted. When we provide a new password, the script shows the password strength for the root password we entered, and we have an opportunity to change it if we want to. We will skip this step and enter *N* for no at this stage.

For the rest of the questions, we keep pressing the *Y* and *Enter* keys at each prompt from the script. This essentially removes some default anonymous users and the test database. Now, you can just start your *mysql* shell by using the following command:

```
# mysql
```

INSTALLING PHP

PHP stands for Hypertext Pre-processor, which is an open source, server-side scripting language for the development of web applications and services. We will use Ubuntu's apt package manager to install PHP.

```
# apt install php libapache2-mod-php php-mysql
```

We will also install some additional packages, which are required to run all the software components together. After this installation is complete, we need to do some additional setup for our Apache Server.

By default, an Apache web server serves HTML files as a preference, and then looks for CGI and Perl files if the HTML file is not available. If the CGI or Perl file is not found, then it checks for a PHP file. However, since we wish to use PHP for our server-side programs in all cases, we need to change this behavior. We can change the Apache directory configuration with the following command:

```
# nano /etc/apache2/mods-enabled/dir.conf
```

This opens the configuration file in the default Ubuntu editor, called *nano*. This file has default file names listed in order, as shown:

```
<IfModule mod_dir.c>
```

```
DirectoryIndex index.html index.cgi index.pl index.php  
index.xhtml index.htm
```

```
</IfModule>
```

First, we change the order of the file names, starting with *index.php* followed by the *index.html*, and then the rest.

```
<IfModule mod_dir.c>
```

```
DirectoryIndex index.php index.html index.htm index.cgi  
index.pl index.xhtml
```

```
</IfModule>
```

Once the changes are done, the file can be saved by pressing *Ctrl+X* and then typing *Y*, followed by pressing *Enter*. This exits us from the editor. We restart the Apache web server to make these changes effective by using the following command:

```
# systemctl restart apache2
```

This silently restarts the Apache web server and reloads the configurations with the new changes. However, we still need to validate that these changes are effective. To do so, we create a test PHP program file and verify it within the browser by navigating to the IP address of our cloud instance. To create a new test program, we can open a new file with this command and add a few basic lines in the file as follows:


```
# nano /var/www/html/test.php
Add these contents to the file
<?php
echo ("Hi...Welcome to my IoT Platform!");
?>
```

Once finished, we save and close with the *Ctrl+X* combination followed by typing *Y* and then pressing *Enter*. Now when we navigate to http://<INSTANCE_IP>/test.php, we should see the message as shown in the **Figure 5**. At this stage, our LAMP stack is fully functional, but we need to enable an easy and efficient access to MySQL for which we will be using the popular phpMyAdmin program.



FIGURE 5 – Shown here is webpage that will be visible once you run the PHP test script after setting the Apache Server to serve PHP files instead of HTML (by default).

PhpMYADMIN

phpMyAdmin is a free and open-source administration tool for MySQL. As a portable web application written primarily in PHP, it has become one of the most popular MySQL administration tools. To begin the installation, we will first update the package index of our cloud instance. This is followed by the installation of base files for phpMyAdmin with the following two commands:

```
# apt update
# apt install phpmyadmin php-mbstring
```

At this stage of the installation process, the system asks a few questions. On the first screen, we will select *apache2* as the server. We use the spacebar to mark our selection while we move our choices using arrow keys. Once selected, the installation continues and then asks the next question with another prompt —“*configure database for phpmyadmin with dbconfig-common*“. Select *Yes*.

Finally, it asks you to choose and confirm your MySQL application password for phpMyAdmin. After you input that, the installation is complete. At this stage, the installation has added the phpMyAdmin Apache configuration file to the */etc/apache2/conf-enabled/* directory, where it is read automatically. The only thing we now need to do is explicitly enable the *mbstring* PHP extension, which we can do by entering the following:

```
# phpenmod mbstring
# systemctl restart apache2
```

Now we can access the MySQL database with phpMyAdmin by navigating to http://<INSTANCE_IP>/phpmyadmin. It asks for credentials, which we just created. Upon

providing the correct credentials, we are able to access our MySQL database in the browser, as shown in **Figure 6**.



FIGURE 6 – This is the phpMyAdmin login page that will be visible after you install and set up in on your Droplet.

After this, we have a fully functional platform with a running system, a phpMyAdmin based database management system and also PHP programming capabilities. But I wanted to make it even easier, enabling even the absolute beginners to be able to leverage the most out of this. For that reason, we will be using Node-RED to do all the tasks of our platform for now. Note that when you want to take this into production, you will need to move to a more sophisticated language like Python or Node.js for APIs and database management. But for now, we will install Node-RED to our virtual server.

NODE.JS AND NODE-RED

Node.js is a JavaScript platform for general-purpose programming that allows users to build network applications quickly. By leveraging JavaScript on both the frontend and backend, Node.js makes development more consistent and integrated. We will use Ubuntu's apt package manager to update and install Node.js on our cloud instance with the following commands:

```
# apt update
# apt install nodejs
# apt install npm
```

In those three commands, the first command refreshes the local package index, and the next command installs Node.js on our instance. We are also installing the node package manager (NPM), which helps us update and add node packages to our instance as needed. For each of the installations, we are prompted with the amount of disk space being used. We are selecting *Y* for yes in both cases. Upon successful installation, we can check the installed version of each application with the following commands:

```
# nodejs -v
# npm -v
```

If these run without any error, that means node.js has been successfully installed on your server. Now, we will get Node-RED using the NPM or node packet manager using the following command:

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

After the installation is done, you start Node-RED using the command:

```
# node-red
```

The Node-RED software runs on the port 1880 of our IP Address and you can access it easily through your browser. But if you have your firewall enabled, you need to allow TCP connection on the port 1880 to access your Node-RED Dashboard. This can be done using the following command:

```
# ufw allow 1880/tcp
```

Now, go to your browser and navigate to the public IP

address `http://<INSTANCE_IP>:1880` and you should see the Node-RED Dashboard as shown in **Figure 7**. This enables us to create simple REST APIs and set up MQTT connections using a simple drag and drop programming mechanism. We will cover these in Part 2 of this article series. For now, we will just install the MQTT server to our platform as the last task of this article.

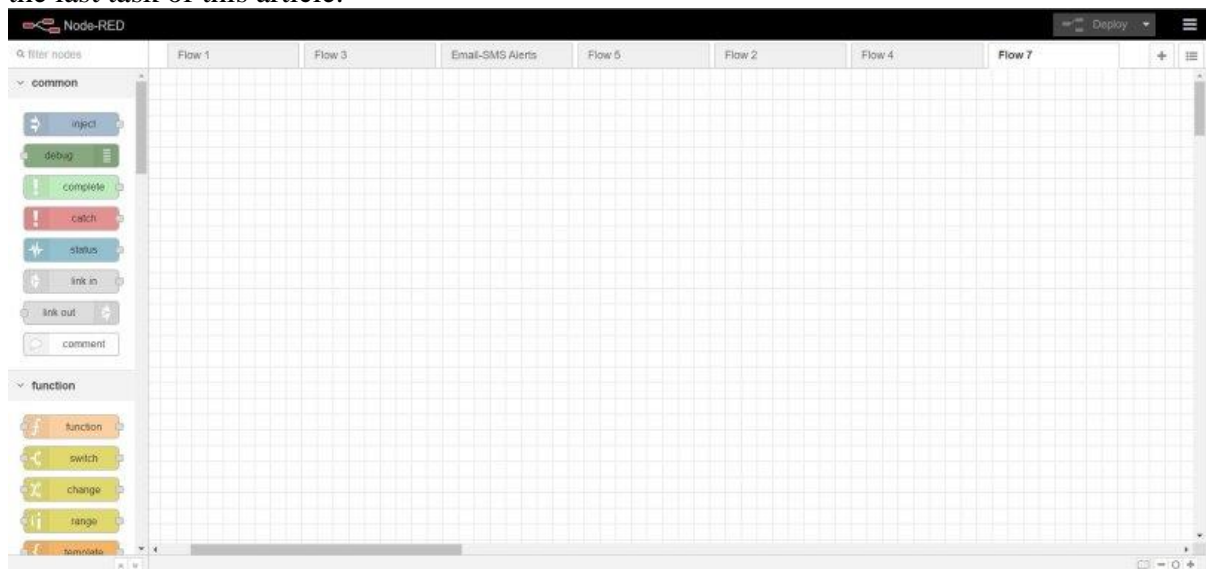


FIGURE 7 – This is the Node-RED home page you'll see when opening the following web address: {your Droplet ip here}:1880.

MQTT: MESSAGE BROKER

As we know, we will be using HTTP and MQTT protocols for our IoT platform. HTTP is pretty straightforward and doesn't require any extra software but when it comes to MQTT, you need a MQTT Broker running on your platform to handle MQTT connections and data. MQTT is a broker-based communication protocol that follows a publish/subscribe mechanism. This paradigm is explained in **Figure 8**.

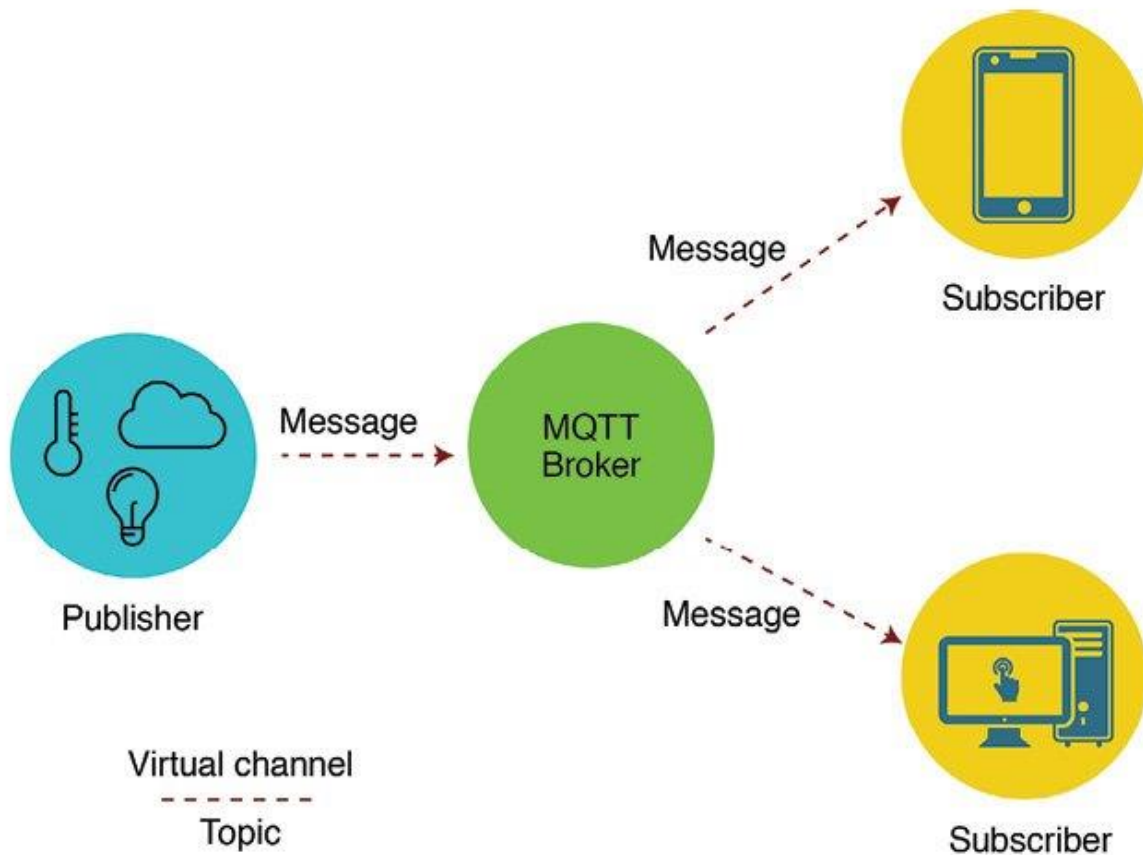


FIGURE 8 – This image describes how the MQTT communication protocol works. It is a publish/subscribe protocol with clients and a broker.

Now that we have discussed the fundamentals of a publish/subscribe mechanism and the way that a broker functions, let's install and configure the MQTT broker on our cloud instance. We will install the Mosquitto broker on our system. The Ubuntu repository has the latest version of Mosquitto available to install through apt package manager.

```
# apt update
```

```
# apt install mosquitto mosquitto-clients
```

After the installation is completed, Mosquitto starts immediately on port 1883, which can be checked using the following command:

```
# lsof -i :1883
```

By default, Mosquitto listens on a standard MQTT port 1883, and we can see in the output that the program is running and listening on that port. At this stage, we can run a quick check to see if everything is working as expected. To do that, we open another terminal window and log in for a second time. Let's keep the two windows open side by side to see what is going on. On one terminal, we subscribe to *test_Topic* with the following command:

```
# mosquitto_sub -h localhost -t test_Topic
```

Here, *-h* specifies the hostname and *-t* specifies the subscription topic. After sending the command, we do not see anything output because the client that we just created is listening for new messages and nothing has been sent yet. On the other terminal, we publish a test message on *test_Topic* as follows:

```
# mosquitto_pub -h localhost -t test_Topic -m "Hello world!"
```

The additional *-m* option is for specifying a message for the given topic. Once the command is sent, you should see this message pop up on another terminal that has subscribed to this *test_Topic*.

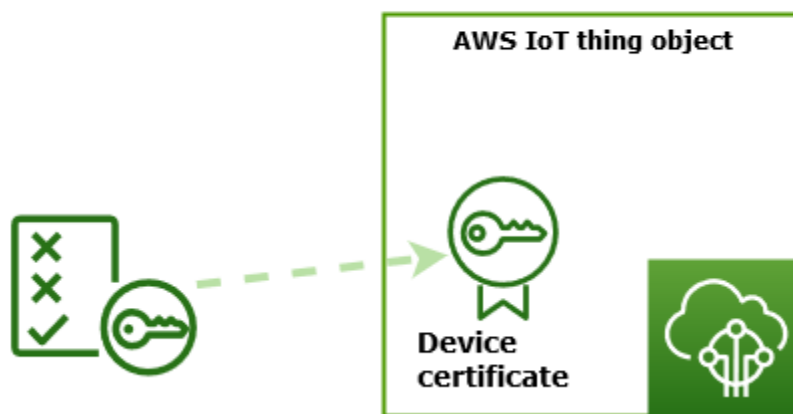
Practical No: 7

Aim: Use IoT device with AWS.

In this tutorial, you'll create the AWS IoT resources that a device requires to connect to AWS IoT Core and exchange messages.

Create an AWS IoT Core policy

Create a thing and its certificate



1. Create an AWS IoT policy document, which will authorize your device to interact with AWS IoT services.
2. Create a thing object in AWS IoT and its X.509 device certificate, and then attach the policy document. The thing object is the virtual representation of your device in the AWS IoT registry. The certificate authenticates your device to AWS IoT Core, and the policy document authorizes your device to interact with AWS IoT.

Create an AWS IoT policy

Devices use an X.509 certificate to authenticate with AWS IoT Core. The certificate has AWS IoT policies attached to it. These policies determine which AWS IoT operations, such as subscribing or publishing to MQTT topics, the device is permitted to perform. Your device presents its certificate when it connects and sends messages to AWS IoT Core.

Follow the steps to create a policy that allows your device to perform the AWS IoT operations necessary to run the example program. You must create the AWS IoT policy before you can attach it to the device certificate, which you'll create later.

To create an AWS IoT policy

1. On the left menu, choose **Security**, and then choose **Policies**. On the **You don't have a policy yet** page, choose **Create policy**.

If your account has existing policies, choose **Create**.

2. On the **Create policy** page:
 - a. In the **Policy properties** section, in the **Policy name** field, enter a name for the policy (for example, **My_Iot_Policy**). Don't use personally identifiable information in your policy names.
 - b. In the **Policy document** section, create the policy statements that grant or deny resources access to AWS IoT Core operations. To create a policy statement that grants all clients to perform **iot:Connect**, follow these steps:
 - In the **Policy effect** field, choose **Allow**. This allows all clients that have this policy attached to their certificate to perform the action listed in the **Policy action** field.
 - In the **Policy action** field, choose a policy action such as **iot:Connect**. Policy actions are the actions that your device needs permission to perform when it runs the example program from the Device SDK.
 - In the **Policy resource** field, enter a resource Amazon Resource Name (ARN) or *. A * to select any client (device).

To create the policy statements for **iot:Receive**, **iot:Publish**, and **iot:Subscribe**, choose **Add new statement** and repeat the steps.

Policy effect	Policy action	Policy resource	
Allow ▼	iot:Connect ▼	*	Remove
Allow ▼	iot:Receive ▼	*	Remove
Allow ▼	iot:Publish ▼	*	Remove
Allow ▼	iot:Subscribe ▼	*	Remove

3. After you've entered the information for your policy, choose **Create**.

For more information, see [How AWS IoT works with IAM](#).

Create a thing object

Devices connected to AWS IoT Core are represented by *thing objects* in the AWS IoT registry. A *thing object* represents a specific device or logical entity. It can be a physical device or sensor (for example, a light bulb or a light switch on the wall). It can also be a

logical entity, like an instance of an application or physical entity that doesn't connect to AWS IoT, but is related to other devices that do (for example, a car that has engine sensors or a control panel).

To create a thing in the AWS IoT console

1. In the [AWS IoT console](#), in the left menu, choose **Manage**, then choose **Things**.
2. On the **Things** page, choose **Create things**.
3. On the **Create things** page, choose **Create a single thing**, then choose **Next**.
4. On the **Specify thing properties** page, for **Thing name**, enter a name for your thing, such as **MyIotThing**.

Choose thing names carefully, because you can't change a thing name later.

To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

5. Keep the rest of the fields on this page empty. Choose **Next**.
6. On the **Configure device certificate - optional** page, choose **Auto-generate a new certificate (recommended)**. Choose **Next**.
7. On the **Attach policies to certificate - optional** page, select the policy you created in the previous section. In that section, the policy was named, **My_Iot_Policy**.

Choose **Create thing**.

8. On the **Download certificates and keys** page:
 - a. Download each of the certificate and key files and save them for later. You'll need to install these files on your device.

When you save your certificate files, give them the names in the following table. These are the file names used in later examples.

Certificate file names	
File	File path
Private key	private.pem.key
Public key	<i>(not used in these examples)</i>
Device certificate	device.pem.crt
Root CA certificate	Amazon-root-CA-1.pem

- b. To download the root CA file for these files, choose the **Download** link of the root CA certificate file that corresponds to the type of data endpoint and cipher suite you're using. In this tutorial, choose **Download** to the right of **RSA 2048 bit key: Amazon Root CA 1** and download the **RSA 2048 bit key: Amazon Root CA 1** certificate file.
- c. Choose **Done**.

After you complete this procedure, you should see the new thing object in your list of things.

Practical No: 8

Aim: Send telemetry from a device to an IoT hub and read it with a service application.

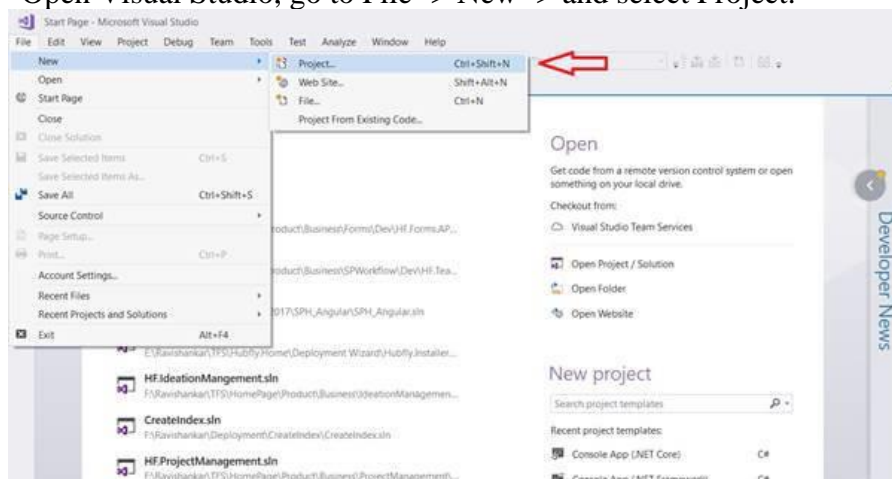
Introduction

In this article, we are going to see how to send telemetry from an IoT device to the Azure IoT Hub using C#. IoT Hub is a cloud platform to securely connect billions of IoT devices to create IoT applications. Please read the previous parts of the article before continuing with this one.

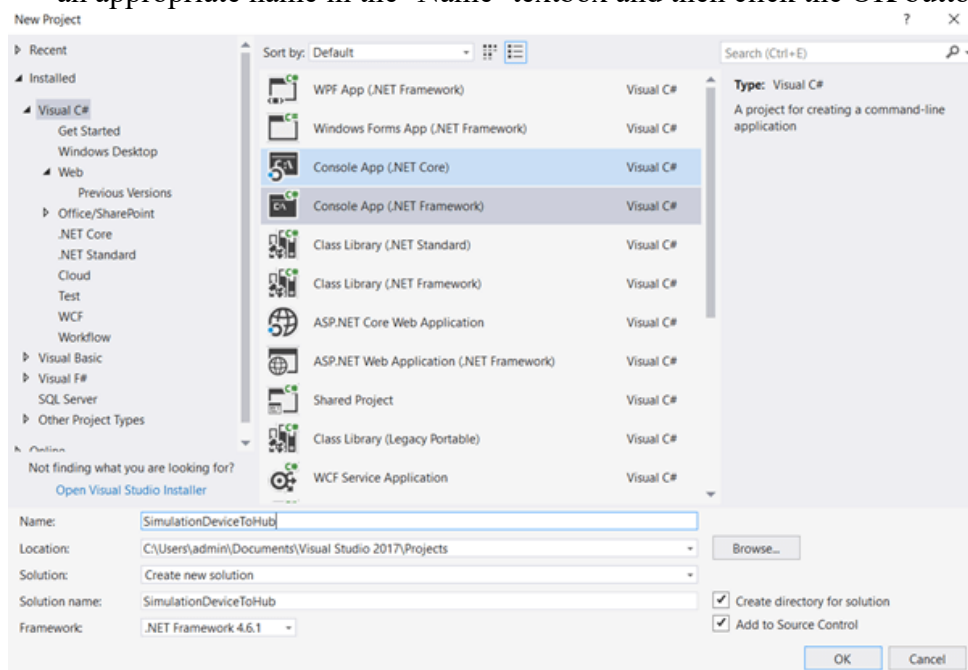
1. How To Create Azure IoT Hub Using PowerShell
2. How to Register IoT Device in Azure IoT Hub Using PowerShell

Creating a C# Console Application

- Open Visual Studio, go to File -> New -> and select Project.

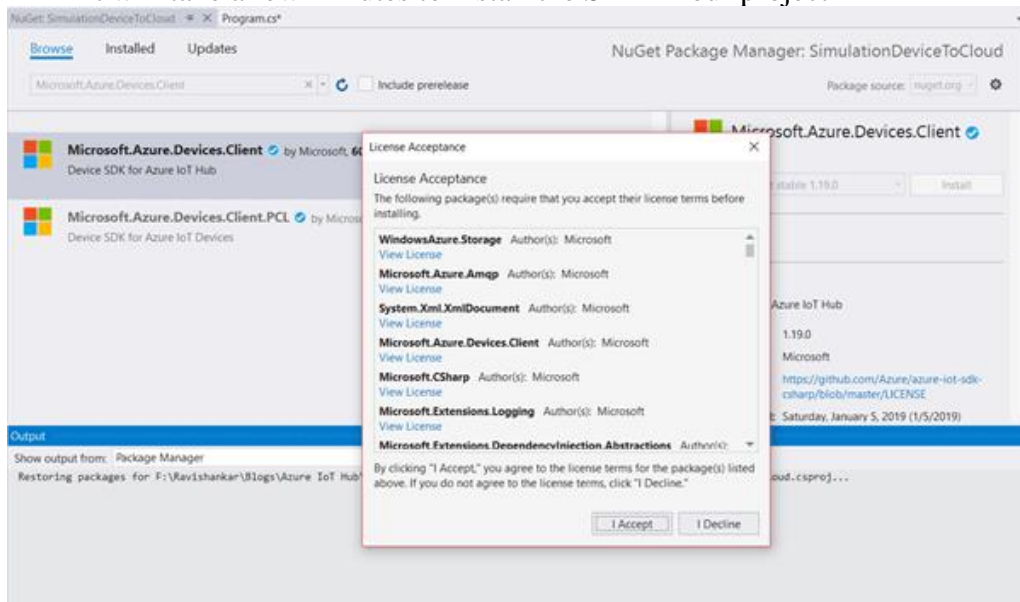


- In Templates, select Visual C#, select Console App (.NET Framework) and give an appropriate name in the 'Name' textbox and then click the OK button.



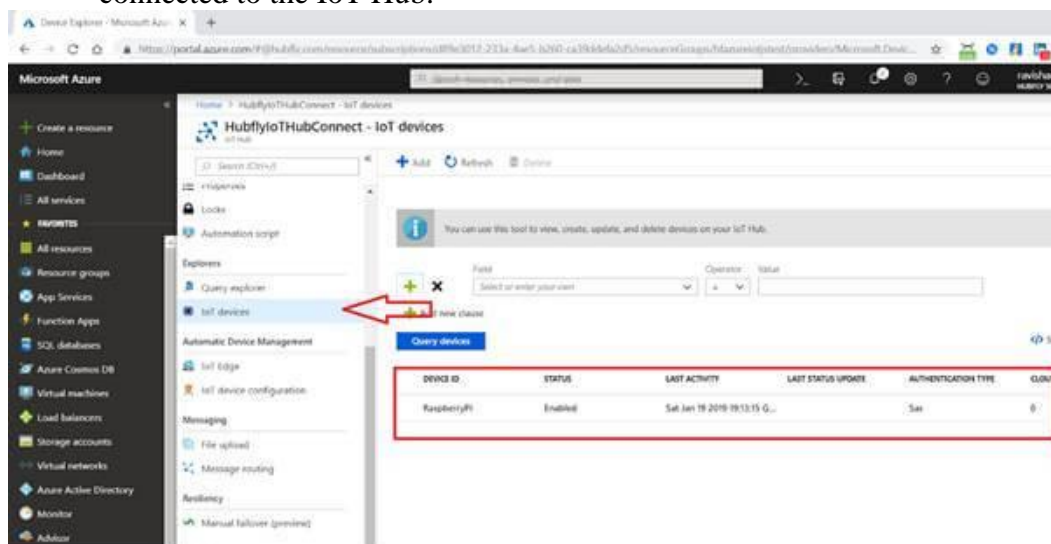
Installing Microsoft Azure IoT Device Client SDK

- Go to Project -> Manage NuGet Packages.
- Click Browse tab and search for Microsoft.Azure.Devices.Client. You will see the Microsoft.Azure.Devices.Client device SDK will have listed in the search result and click Install button
- Now, click the I Accept button to accept the license agreement.
- It will take a few minutes to install the SDK in our project

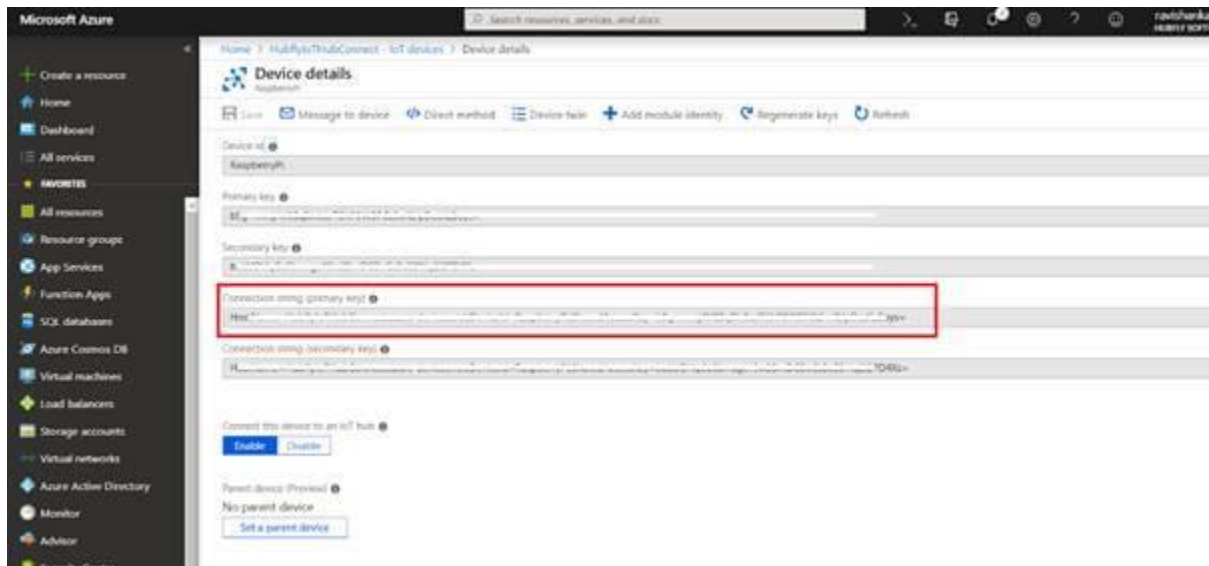


Get IoT device connect string from Azure IoT Hub

- Open your Azure portal and choose your IoT Hub.
- Click IoT devices In Explorers. You can see the list of devices that are connected to the IoT Hub.



- Double click the device, you can see the device detailed information like device id, Primary Key, Secondary Key, Connection String(primary key) and Connection String (secondary key).



- Using Microsoft.Azure.Devices.Client library we can create device client. The device client has CreateFromConnectionString method which requires device connection string as parameter. Create a read only static string s_connectionString01 and assign the connection string that we copy from Azure Portal.
 - Here you can create a random temperature and humidity values using Random() method.
 - Now open the program.cs file and type the below code
1. **using** System;
 2. **using** Microsoft.Azure.Devices.Client;
 3. **using** System.Text;
 4. **using** Newtonsoft.Json;
 5. **using** System.Threading.Tasks;
 - 6.
 7. **namespace** SimulationDeviceToCloud
 8. {
 9. **class** Program
 10. {
 11. **private static** DeviceClient s_deviceClient;
 12. **private readonly static** string s_connectionString01 = "HostName=HubflyIoT
oTHubConnect.azure-
devices.net;DeviceId=RaspberryPi;SharedAccessKey=b9g+mmjAV8SqBlv8o/T
ChP0WBFCL5wi8/pDccXzBoys=";
 13. **static void** Main(string[] args)
 14. {
 15. s_deviceClient = DeviceClient.CreateFromConnectionString(s_connecti
onString01, TransportType.Mqtt);
 16. SendDeviceToCloudMessagesAsync(s_deviceClient);
 17. Console.ReadLine();
 18. }
 19. }
 - 20.
 21. **private static async void** SendDeviceToCloudMessagesAsync(DeviceClie
nt s_deviceClient)

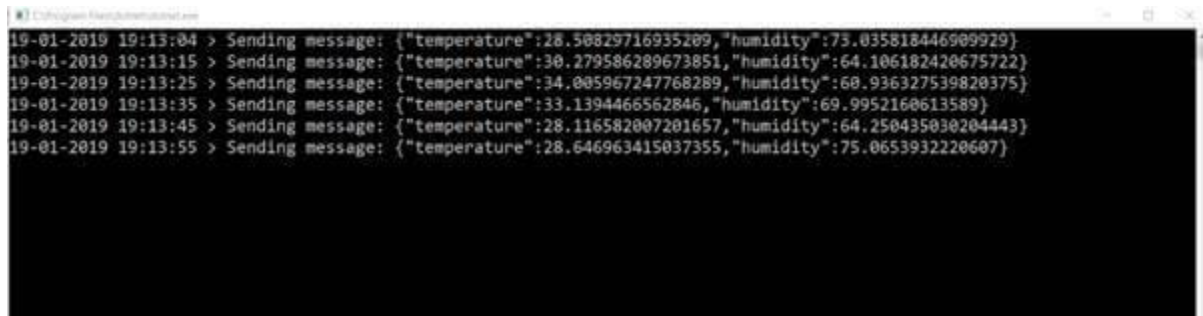
```

22.     {
23.         try
24.         {
25.             double minTemperature = 20;
26.             double minHumidity = 60;
27.             Random rand = new Random();
28.
29.             while (true)
30.             {
31.                 double currentTemperature = minTemperature + rand.NextDouble()
32.                 * 15;
33.                 double currentHumidity = minHumidity + rand.NextDouble() * 20;
34.
35.                 // Create JSON message
36.
37.                 var telemetryDataPoint = new
38.                 {
39.                     temperature = currentTemperature,
40.                     humidity = currentHumidity
41.                 };
42.
43.                 string messageString = "";
44.
45.
46.
47.                 messageString = JsonConvert.SerializeObject(telemetryDataPoint);
48.
49.                 var message = new Message(Encoding.ASCII.GetBytes(messageStr
50.                 ing));
51.
52.                 // Add a custom application property to the message.
53.                 // An IoT hub can filter on these properties without access to the me
54.                 ssage body.
55.                 //message.Properties.Add("temperatureAlert", (currentTemperature
56.                 > 30) ? "true" : "false");
57.
58.                 // Send the telemetry message
59.                 await s_deviceClient.SendEventAsync(message);
60.                 Console.WriteLine("{0} > Sending message: {1}", DateTime.Now,
61.                 messageString);
62.                 await Task.Delay(1000 * 10);
63.             }
64.         }
65.     } catch (Exception ex)
66.     {

```

```
65.         throw ex;
66.     }
67. }
68. }
69. }
70.
```

That's it. Now, run the web application, go to Debug menu, and click on "Start without Debugging" or press F5. This will display the below result



```
19-01-2019 19:13:04 > Sending message: {"temperature":28.50829716935209,"humidity":73.035818446909929}
19-01-2019 19:13:15 > Sending message: {"temperature":30.279586289673851,"humidity":64.106182420675722}
19-01-2019 19:13:25 > Sending message: {"temperature":34.005967247768289,"humidity":60.936327539820375}
19-01-2019 19:13:35 > Sending message: {"temperature":33.1394466562846,"humidity":69.9952160613589}
19-01-2019 19:13:45 > Sending message: {"temperature":28.116582007201657,"humidity":64.250435030204443}
19-01-2019 19:13:55 > Sending message: {"temperature":28.646963415037355,"humidity":75.0653932220607}
```

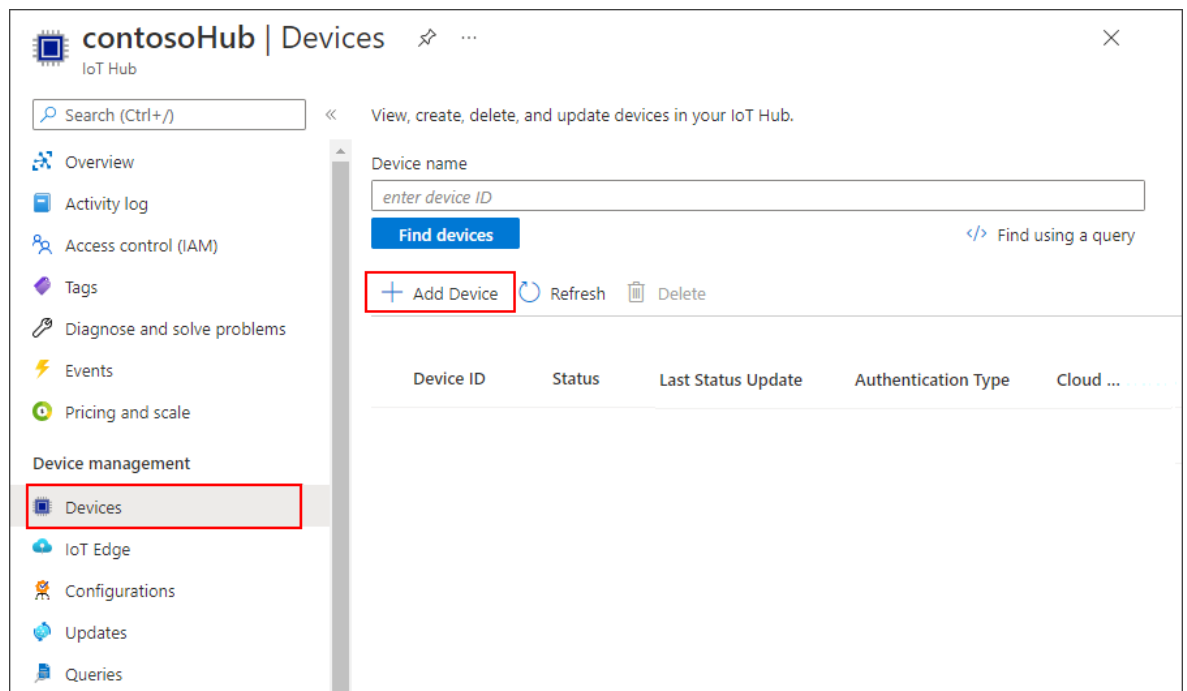
I hope you have learned how to send telemetry from an IoT device to an Azure IoT Hub using C#. Feel free to fill up the comment box below if you need any further assistance from us.

Practical No: 9

Aim: Use the Azure CLI and Azure portal to configure IoT Hub message routing.

Register a new device in your IoT hub.

- **Azure portal**
 1. Sign in to the Azure portal and navigate to your IoT hub.
 2. Select **Devices** from the **Device management** section of the menu.
 3. Select **Add device**.



4. Provide a device ID and select **Save**.
5. The new device should be in the list of devices now. If it's not, refresh the page. Select the device ID to open the device details page.
6. Copy one of the device keys and save it. You'll use this value to configure the sample code that generates simulated device telemetry messages.

Azure CLI

1. Define variables for your IoT hub and device.

IOTHUB_NAME: Replace this placeholder with the name of your IoT hub.

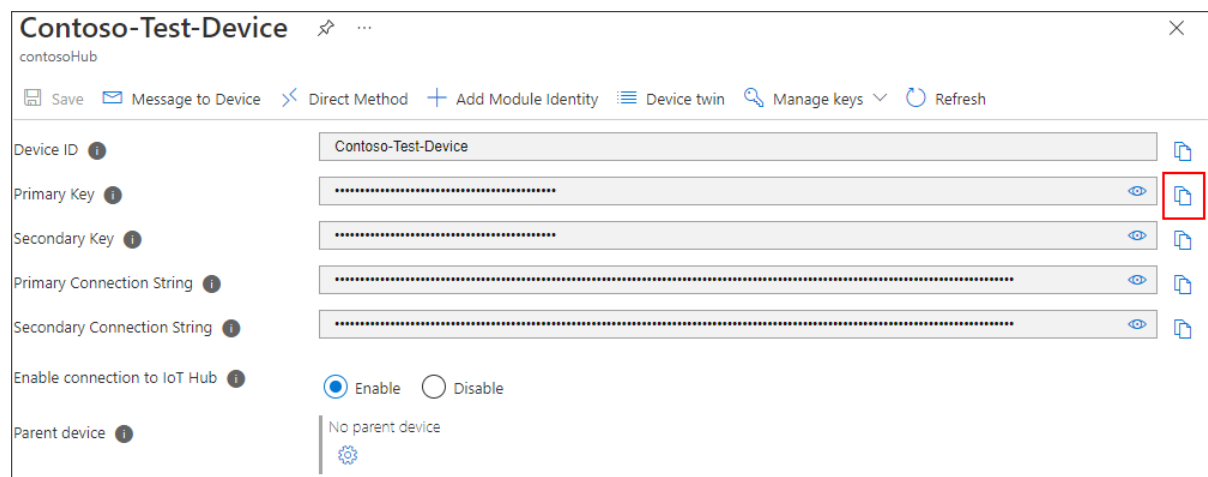
DEVICE_NAME: Replace this placeholder with any name you want to use for the device in this tutorial.

```
Azure CLICopy
Open Cloudshell
hubName=IOTHUB_NAME
deviceName=DEVICE_NAME
```

2. Run the `az iot hub device-identity create` command in your CLI shell. This command creates the device identity.

```
Azure CLICopy
Open Cloudshell
az iot hub device-identity create --device-id $deviceName --hub-name
$hubName
```

3. From the device-identity output, copy the **primaryKey** value without the surrounding quotation marks and save it. You'll use this value to configure the sample code that generates simulated device telemetry messages.



Now that you have a device ID and key, use the sample code to start sending device telemetry messages to IoT Hub.

Tip

If you're following the Azure CLI steps for this tutorial, run the sample code in a separate session. That way, you can allow the sample code to continue running while you follow the rest of the CLI steps.

1. If you didn't as part of the prerequisites, download or clone the [Azure IoT SDK for C# repo](#) from GitHub now.
2. From the folder where you downloaded or cloned the SDK, navigate to the `azure-iot-sdk-csharp\iothub\device\samples\how to guides\HubRoutingSample` folder.

3. Install the Azure IoT C# SDK and necessary dependencies as specified in the HubRoutingSample.csproj file:

ConsoleCopy

```
dotnet restore
```

4. In an editor of your choice, open the Parameters.cs file. This file shows the parameters that are supported by the sample. Only the PrimaryConnectionString parameter will be used in this article when running the sample. Review the code in this file. No changes are needed.
5. Build and run the sample code using the following command:

Replace <myDevicePrimaryConnectionString> with your primary connection string from your device in your IoT hub.

Windows Command PromptCopy

```
dotnet run --PrimaryConnectionString <myDevicePrimaryConnectionString>
```

6. You should start to see messages printed to output as they are sent to IoT Hub. Leave this program running during the tutorial.

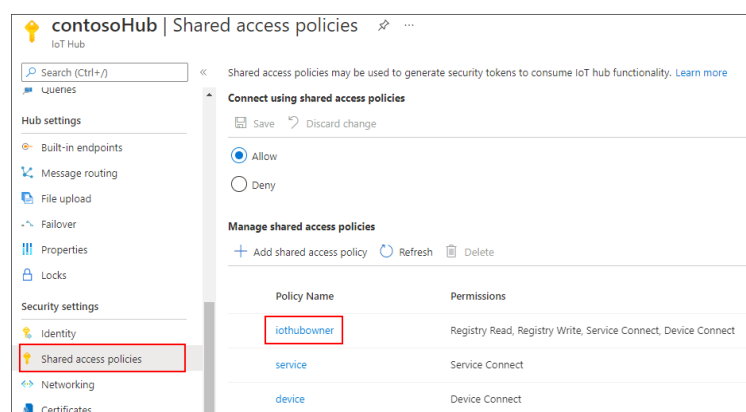
Configure IoT Explorer to view messages

Configure IoT Explorer to connect to your IoT hub and read messages as they arrive at the built-in endpoint.

First, retrieve the connection string for your IoT hub.

- **Azure portal**

1. In the Azure portal, navigate to your IoT hub.
2. Select **Shared access policies** from the **Security settings** section of the menu.
3. Select the **iothubowner** policy.



4. Copy the **Primary connection string**.

Azure CLI

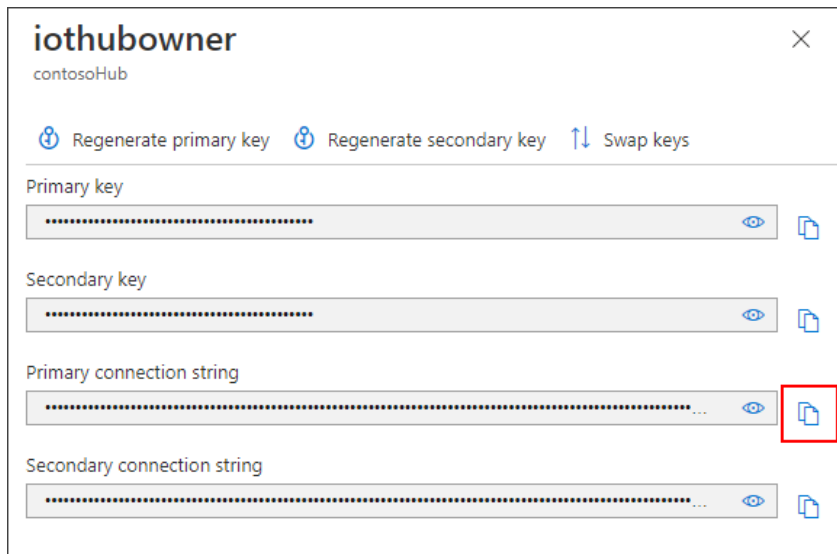
1. Run the `az iot hub connection-string show` command:

Azure CLICopy

Open Cloudshell

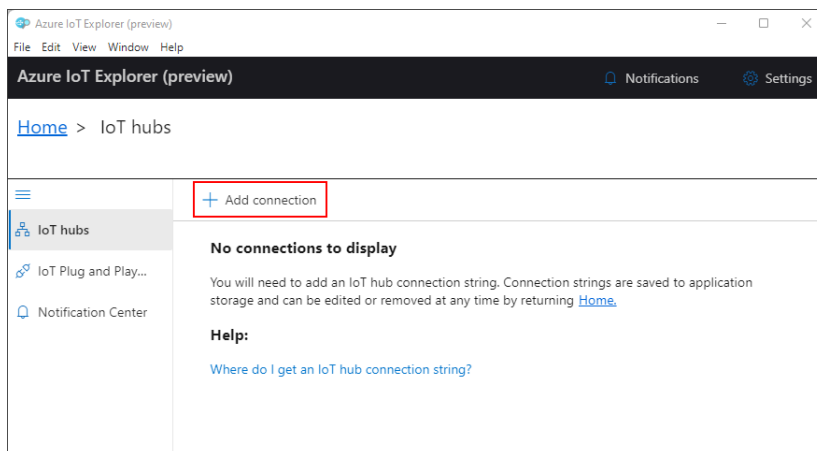
```
az iot hub connection-string show --hub-name $hubName
```

2. Copy the connection string without the surrounding quotation marks.



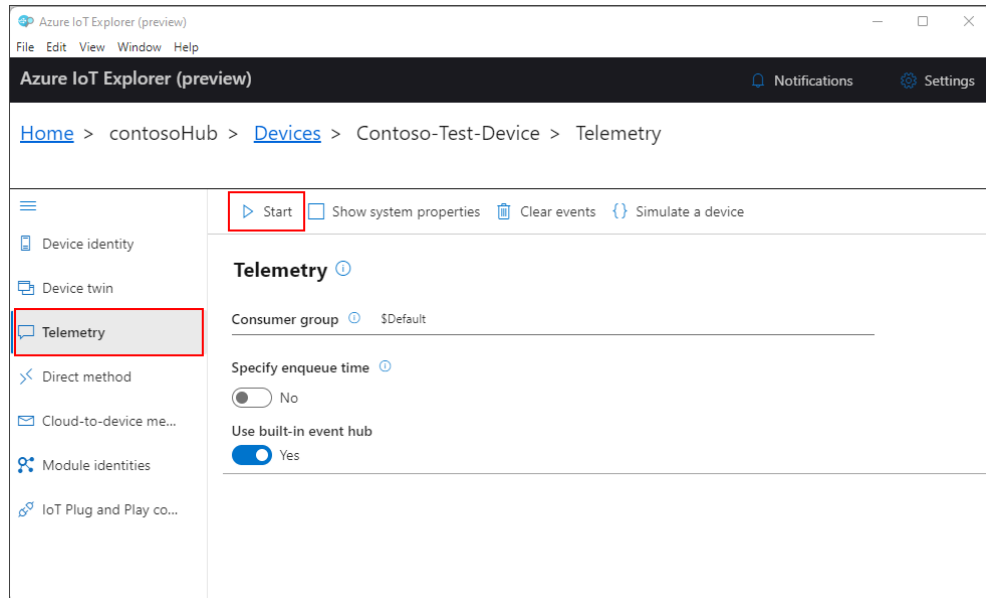
Now, use that connection string to configure IoT Explorer for your IoT hub.

1. Open IoT Explorer on your development machine.
2. Select **Add connection**.

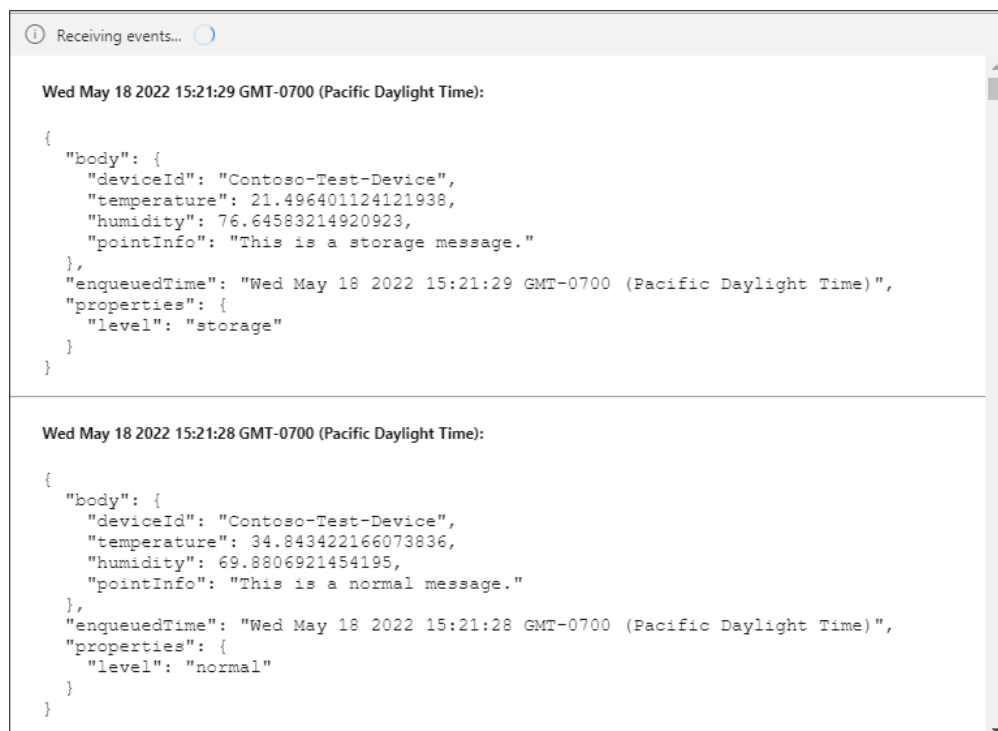


3. Paste your hub's connection string into the text box.
4. Select **Save**.

5. Once you connect to your IoT hub, you should see a list of devices. Select the device ID that you created for this tutorial.
6. Select **Telemetry**.
7. With your device still running, select **Start**. If your device isn't running you won't see telemetry.



8. You should see the messages arriving from your device, with the most recent displayed at the top.



Watch the incoming messages for a few moments to verify that you see three different types of messages: normal, storage, and critical. After seeing this, you can stop your device.

These messages are all arriving at the default built-in endpoint for your IoT hub. In the next sections, we're going to create a custom endpoint and route some of these messages to storage based on the message properties. Those messages will stop appearing in IoT Explorer because messages only go to the built-in endpoint when they don't match any other routes in IoT hub.

Set up message routing

You're going to route messages to different resources based on properties attached to the message by the simulated device. Messages that aren't custom routed are sent to the default endpoint (messages/events).

The sample app for this tutorial assigns a **level** property to each message it sends to IoT hub. Each message is randomly assigned a level of **normal**, **storage**, or **critical**.

The first step is to set up the endpoint to which the data will be routed. The second step is to set up the message route that uses that endpoint. After setting up the routing, you can view endpoints and message routes in the portal.

Create a storage account

Create an Azure Storage account and a container within that account, which will hold the device messages that are routed to it.

- [Azure portal](#)
 1. In the Azure portal, search for **Storage accounts**.
 2. Select **Create**.
 3. Provide the following values for your storage account:

Parameter	Value
Subscription	Select the same subscription that contains your IoT hub.
Resource group	Select the same resource group that contains your IoT hub.
Storage account name	Provide a globally unique name for your storage account.
Performance	Accept the default Standard value.

- 4.
5. You can accept all the other default values by selecting **Review + create**.
6. After validation completes, select **Create**.
7. After the deployment is complete, select **Go to resource**.
8. In the storage account menu, select **Containers** from the **Data storage** section.
9. Select **+ Container** to create a new container.

Azure CLI

1. Define the variables for your storage account and container.

GROUP_NAME: Replace this placeholder with the name of the resource group that contains your IoT hub.

STORAGE_NAME: Replace this placeholder with a name for your storage account. Storage account names must be lowercase and globally unique.

CONTAINER_NAME: Replace this placeholder with a name for your container.

```
Azure CLICopy
Open Cloudshell
resourceGroup=GROUP_NAME
storageName=STORAGE_NAME
containerName=CONTAINER_NAME
```

2. Use the `az storage account create` command to create a standard general-purpose v2 storage account.

Azure CLICopy

Open Cloudshell

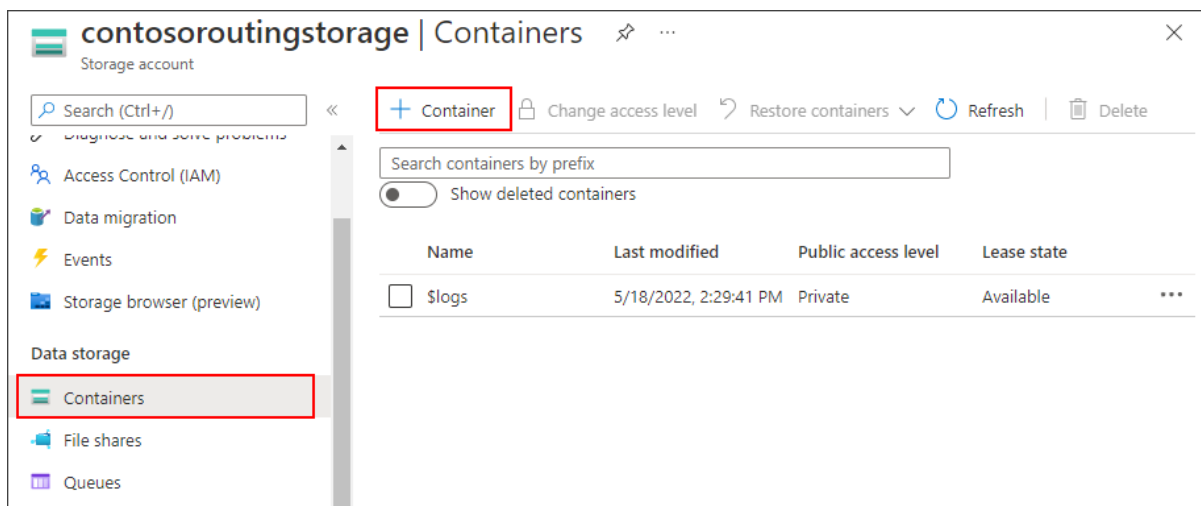
```
az storage account create --name $storageName --resource-group $resourceGroup
```

3. Use the `az storage container create` to add a container to your storage account.

Azure CLICopy

Open Cloudshell

```
az storage container create --auth-mode login --account-name $storageName --name $containerName
```



10. Provide a name for your container and select **Create**.

Route to a storage account

Now set up the routing for the storage account. In this section you define a new endpoint that points to the storage account you created. Then, create a route that filters for messages where the **level** property is set to **storage**, and route those to the storage endpoint.

Note

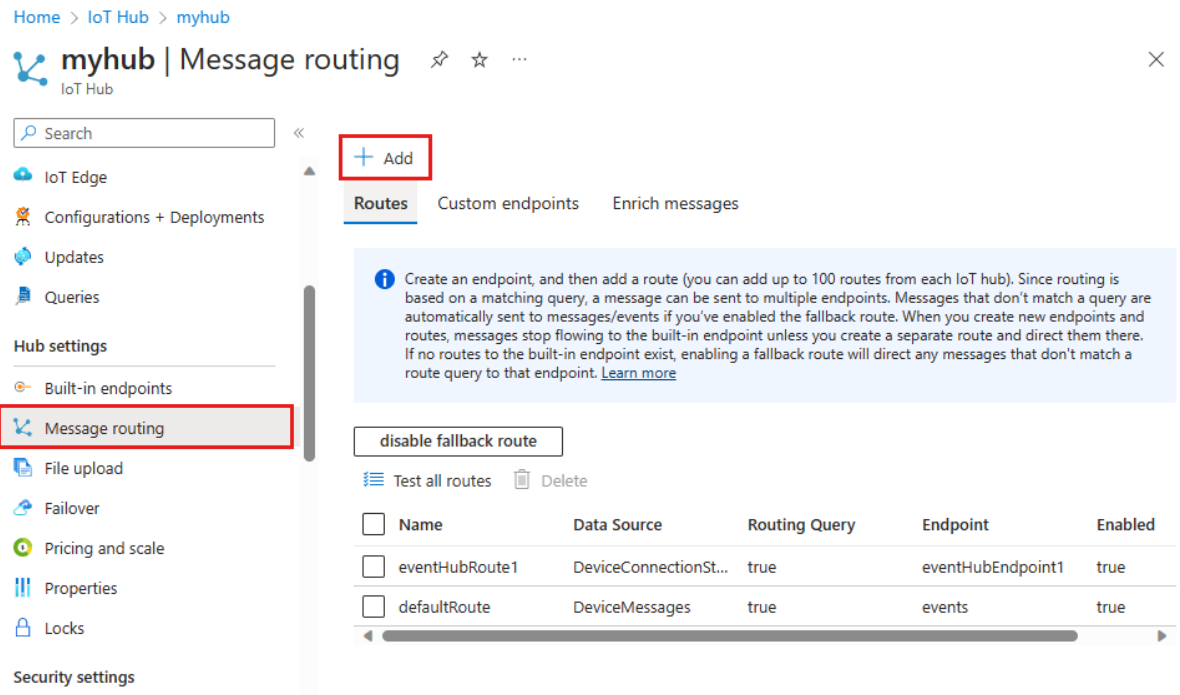
The data can be written to blob storage in either the **Apache Avro** format, which is the default, or JSON.

The encoding format can be only set at the time the blob storage endpoint is configured. The format cannot be changed for an endpoint that has already been set up. When using JSON encoding, you must set the contentType to JSON and the contentEncoding to UTF-8 in the message system properties.

For more detailed information about using a blob storage endpoint, please see [guidance on routing to storage](#).

- **Azure portal**

1. In the [Azure portal](#), go to your IoT hub.
2. In the resource menu under **Hub settings**, select **Message routing** then select **Add**.



3. On the **Endpoint** tab, create a Storage endpoint by providing the following information:

Parameter	Value
Endpoint type	Select Storage .
Endpoint name	Provide a unique name for this endpoint.
Azure Storage container	Select Pick a container . Follow the prompts to select the storage account and container that you created in the previous section.
Encoding	Select JSON . If this field is greyed out, then your storage account region does not support JSON. In that case, continue with the default AVRO .

Add a route

myhub

1 Endpoint 2 Route 3 Enrichment

Create an endpoint for your route—this will determine which Azure services will receive your messages. You can have a maximum of 10 endpoints for each IoT Hub. [Learn more](#)

Endpoint type ⓘ

Storage

Endpoint name *

StorageEndpoint

[Select existing](#)

Azure storage container *

Pick a container

Batch frequency * ⓘ

100

Chunk size window * ⓘ

100

Encoding * ⓘ

☐ AVRO

☒ JSON

File name format * ⓘ

{iothub}/{partition}/{YYYY}/{MM}/{DD}/{HH}/{mm}

If multiple files are created within the same minute, the filename format would be {iothub}/0/2023/5/11/13/19-01.avro

Authentication type *

Choose the authentication type for this routing endpoint. [Learn more](#)

☒ Key-based

☐ System-assigned

☐ User-assigned

ⓘ System-assigned identity is switched off and cannot be used as an authentication type.

-
-
-
-
4. Accept the default values for the rest of the parameters and select **Create + next**.
6. On the **Route** tab, provide the following information to create a route that points to the Storage endpoint you created:

Parameter	Value
Name	Create a name for your route.
Data source	Verify that Device Telemetry Messages is selected from the dropdown list.
Enable route	Verify that this field is checked.
Routing query	Enter level="storage" as the query string.

Add a route ...

myhub

✓ Endpoint 2 Route 3 Enrichment

Create a route, and add a query to filter data before routing it to the endpoint. [Learn more](#)

Name *

StorageRoute

☒ Enable route

Data source *

Device Telemetry Message

Routing query

1 level="storage"

> Test

- 7.
8. Select **Create + skip enrichments**.
- 9.

Azure CLI

1. Configure the variables that you need for the endpoint and route commands.

ENDPOINT_NAME: Provide a name for the endpoint that represents your storage container.

ROUTE_NAME: Provide a name for the route that filters messages for the storage endpoint

```
Azure CLICopy
Open Cloudshell
endpointName=ENDPOINT_NAME
routeName=ROUTE_NAME
```

2. Use the `az iot hub routing-endpoint create` command to create a custom endpoint that points to the storage container you made in the previous section.

```
Azure CLICopy
Open Cloudshell
az iot hub routing-endpoint create \
  --connection-string $(az storage account show-connection-string --name
$storageName --query connectionString -o tsv) \
  --endpoint-name $endpointName \
  --endpoint-resource-group $resourceGroup \
  --endpoint-subscription-id $(az account show --query id -o tsv) \
  --endpoint-type azurestoragecontainer
--hub-name $hubName \
```

```
--container $containerName \  
--resource-group $resourceGroup \  
--encoding json
```

3. Use the `az iot hub route create` command to create a route that passes any message where level=storage to the storage container endpoint.

```
Azure CLICopy  
Open Cloudshell  
az iot hub route create \  
  --name $routeName \  
  --hub-name $hubName \  
  --resource-group $resourceGroup \  
  --source devicemessages \  
  --endpoint-name $endpointName \  
  --enabled true \  
  --condition 'level="storage"'
```

View routed messages

Once the route is created in IoT Hub and enabled, it will immediately start routing messages that meet its query condition to the storage endpoint.

Monitor the built-in endpoint with IoT Explorer

Return to the IoT Explorer session on your development machine. Recall that the IoT Explorer monitors the built-in endpoint for your IoT hub. That means that now you should be seeing only the messages that are *not* being routed by the custom route we created.

Start the sample again by running the code. Watch the incoming messages for a few moments and you should only see messages where level is set to normal or critical.

View messages in the storage container

Verify that the messages are arriving in the storage container.

1. In the [Azure portal](#), navigate to your storage account.
2. Select **Containers** from the **Data storage** section of the menu.
3. Select the container that you created for this tutorial.
4. There should be a folder with the name of your IoT hub. Drill down through the file structure until you get to a **.json** file.
5. Select the JSON file, then select **Download** to download the JSON file.
Confirm that the file contains messages from your device that have the level property set to storage.
6. Stop running the sample.

Clean up resources

If you want to remove all of the Azure resources you used for this tutorial, delete the resource group. This action deletes all resources contained within the group. If you don't want to delete the entire resource group, use the Azure portal to locate and delete the individual resources.

If you intend to continue to the next tutorial, keep the resources that you created here.

- **Azure portal**
 1. In the Azure portal, navigate to the resource group that contains the IoT hub and storage account for this tutorial.
 2. Review all the resources that are in the resource group to determine which ones you want to clean up.
 - If you want to delete all the resource, select **Delete resource group**.
 - If you only want to delete certain resource, use the check boxes next to each resource name to select the ones you want to delete. Then select **Delete**.

Azure CLI

1. Use the az resource list command to view all the resources in your resource group.

Azure CLICopy

Open Cloudshell

```
az resource list --resource-group $resourceGroup --output table
```

2. Review all the resources that are in the resource group to determine which ones you want to clean up.
 - If you want to delete all the resources, use the az group delete command.

Azure CLICopy

Open Cloudshell

```
az group delete --name $resourceGroup
```

- If you only want to delete certain resources, use the az resource delete command. For example:

Azure CLICopy

Open Cloudshell

```
az resource delete --resource-group $resourceGroup --name $storageName
```

Practical No: 1

Aim: Face Detection using IoT device. (Pi Camera or anything else).

Raspberry Pi is a powerful tool, and when coupled with OpenCV library can be used for many image processing projects. At the end of this article you will learn to build one such application '**Face detection**'. Face detection is exactly what it sounds like, the camera will capture an image and find the faces in the image and show the user. For this purpose, we will use a cascade classifier that OpenCV already has in order to detect the face. In this project we will also use the Raspberry Pi camera module to take the pictures for analysis.













Hardware requirements:

1. Raspberry Pi with OS installed (available on the official website)
2. Camera module
3. Power cable
4. Monitor
5. HDMI connector
6. USB or Bluetooth mouse
7. USB or Bluetooth keyboard

Setup:

Software:

Before starting the project, download OpenCV and extract it from the official website. After which, if you open the file you should see a sub folder called sources. In sources, click on data sub-folder, in which you can open a file called haarcascades.

 haarcascade_eye	25-01-2015 12:01 PM	XML Document	334 KB
 haarcascade_eye_tree_eyeglasses	25-01-2015 12:01 PM	XML Document	588 KB
 haarcascade_frontalcatface	08-03-2015 09:36 AM	XML Document	370 KB
 haarcascade_frontalcatface_extended	08-03-2015 09:36 AM	XML Document	353 KB
 haarcascade_frontalface_alt	25-01-2015 12:01 PM	XML Document	661 KB
 haarcascade_frontalface_alt_tree	25-01-2015 12:01 PM	XML Document	2,627 KB
 haarcascade_frontalface_alt2	25-01-2015 12:01 PM	XML Document	528 KB
 haarcascade_frontalface_default	25-01-2015 12:01 PM	XML Document	909 KB
 haarcascade_fullbody	08-02-2015 11:50 AM	XML Document	466 KB
 haarcascade_lefteye_2splits	25-01-2015 12:01 PM	XML Document	191 KB
 haarcascade_licence_plate_rus_16stages	25-01-2015 12:01 PM	XML Document	47 KB
 haarcascade_lowerbody	08-02-2015 11:50 AM	XML Document	387 KB

In this list, copy the 'haarcascade_frontalface_default' XML and paste it in the same folder that you are going to use to save your program.

Hardware:

1. Connect the Raspberry Pi camera module and enable it (for any help on that, check out our article on that <https://iot4beginners.com/how-to-capture-image-and-video-in-raspberry-pi/>).
2. Use the HDMI cable to connect your Raspberry Pi to a display
3. Connect your mouse and keyboard as well (through USB or Bluetooth)
4. Make sure to install NumPy and OpenCV library on your Raspberry Pi before you start



Code:

```
import numpy as np
import cv2
import io
import picamera
```

```
stream = io.BytesIO()
```

```
with picamera.PiCamera() as camera:
```

```
    camera.capture(stream,format='jpeg')
```

```
numpyarray = numpy.fromstring(stream.getvalue(),dtype=numpy.uint8)
```

```
image = cv2.imdecode(numpyarray,1)
```

```
faceDetectset= cv2.CascadeClassifier('haarcascade_frontalface_default.xml');
```

```
#convert to gray scale
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
#now we can look for faces
faces = faceDetectset.detectMultiScale(gray,1.3,5);
for (x,y,w,h) in faces:
    #draw box on original
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,0),2)
```

```
#show the output in window
cv2.imshow("Detected",img)
if cv2.waitKey()==ord('q'):
    break;
```

```
stream.truncate(0)
cv2.destroyAllWindows()
```

Code Explanation:

First we start the code by **importing the libraries** necessary for the program.

```
import numpy as np
```

```
import cv2
```

```
import io
```

```
import picamera
```

```
import time
```

We then create a **stream** to avoid using memory space on your Pi for every image you capture.

After this, we use the Raspberry Pi camera to **capture an image** after showing preview.

```
stream = io.BytesIO()
```

```
#capture image
```

```
with picamera.PiCamera() as camera:
```

```
    camera.start_preview()
```

```
    time.sleep(5)
```

```
    camera.capture(stream,format='jpeg')
```

To use the image captured with OpenCV and make the processing speed faster, we **convert the image to a NumPy array**. We then assign the variable name '**image**' to read data from the '**numpyarray**' and decode to image format.

```
numpyarray = numpy.fromstring(stream.getvalue(),dtype=numpy.uint8)
```

```
image = cv2.imdecode(numpyarray,1)
```

Remember that XML file we found and pasted in the same folder as your program? We are going to finally use that by assigning a variable name to it. It is also worthy to note that classifiers usually only work for **gray scale images**, but the image your pi cam takes is color (i.e.,RGB). As a result, we have to convert the **image to gray scale**.

```
faceDetectset= cv2.CascadeClassifier('haarcascade_frontalface_default.xml');
```

```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

Now we can look for faces and using the function **.detectMultiScale()** we can find the coordinates of the object we are looking for (also as I've specified in the program feel free to play around with the scaling factors). Which in this case, is the face! After finding the face in the image, we then draw a rectangle around the location of the face to indicate it. For this, the parameters we specify are the variable name of the image, the start and end coordinates, the color specification for the rectangle (in this case, black) and then the thickness.

After the box is drawn, we then display the original colored image with the box drawn on it in a separate window called '**Detected**' until the key 'e' is clicked to exit the window.

```
faces = faceDetectset.detectMultiScale(gray,1.3,5);#scaling factors can be varied
```

for (x,y,w,h) in faces:

```
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,0),2)
```

```
cv2.imshow("Detected",img)
```

```
if cv2.waitKey()==ord('e'):
```

```
    break;
```

```
stream.truncate(0)
```

```
cv2.destroyAllWindows()
```

Output:



There's me being detected!

