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/maxis-nbiot-hackathon.
- You can either `git clone` this repo or simply download it.

Create Iot 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.

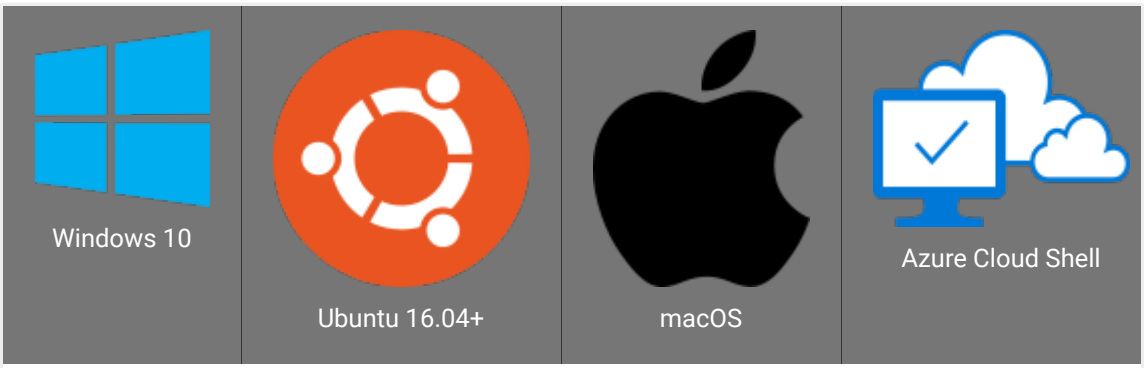
1. In a web browser, navigate to azure.microsoft.com, and click **Free Account**.
2. On the **Create your Azure free account today** page, click **Start free >** button.
3. Scroll down through the page - it lists the products you can access for free, as well as the free products available for the first year.
4. 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**.
9. Finally, check the box to accept the subscription agreement, privacy statement, and communications policy. The second checkbox is optional. Now click **Sign up**.

> **Reference**
>
> For reference, please visit Microsoft Documentation - Create an Azure account

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.



Download the MSI installer	Ubuntu install instructions	macOS install instructions	Run in your browser on Azure Cloud Shell

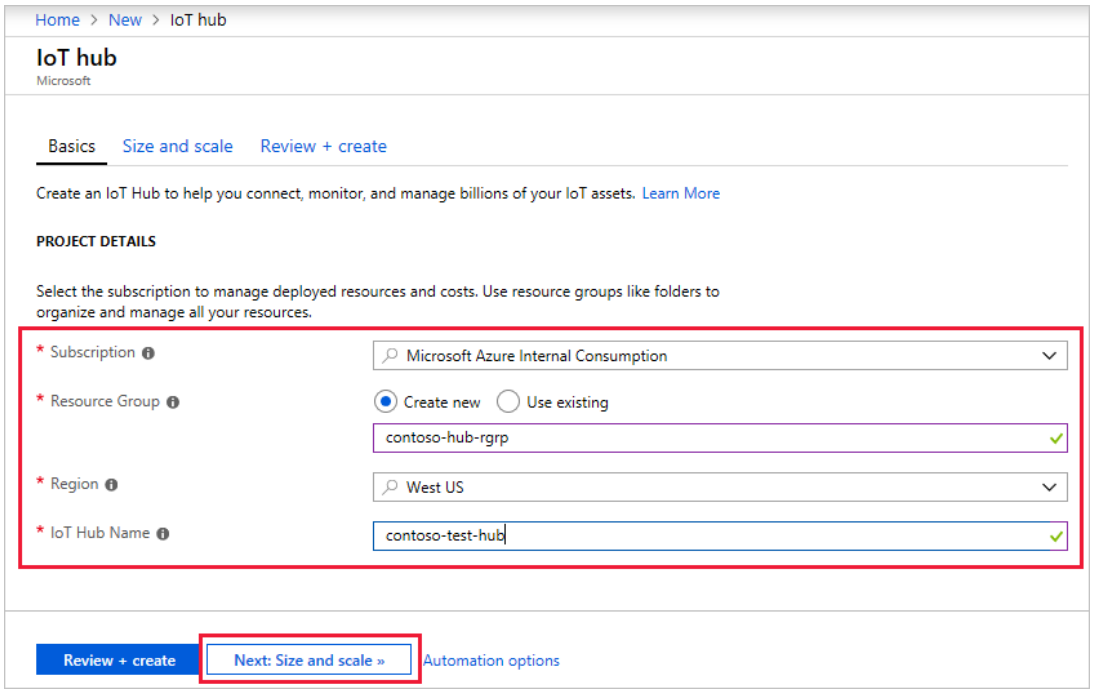
See all of the supported installation platforms.

> **Reference**
>
> For more references, please visit Microsoft Documentations - Overview of Azure CLI

Create IoT Hub

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

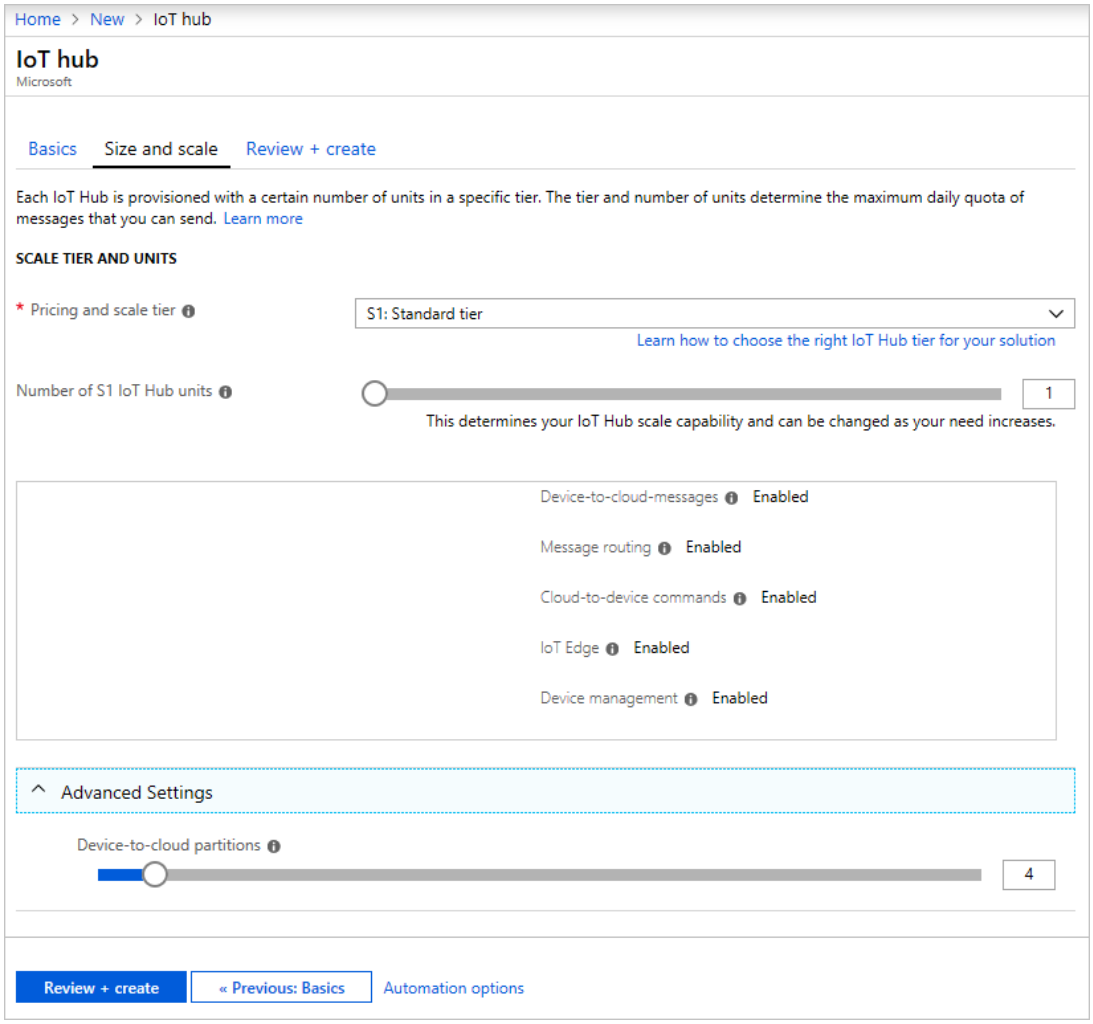
1. Log in to the Azure portal.
2. Choose **+Create a resource**, then *Search the Marketplace* for the **IoT Hub**.
3. Select **IoT Hub** and click the **Create** button. You see the first screen for creating an IoT hub.



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.

> **Important!!**
>
> 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.

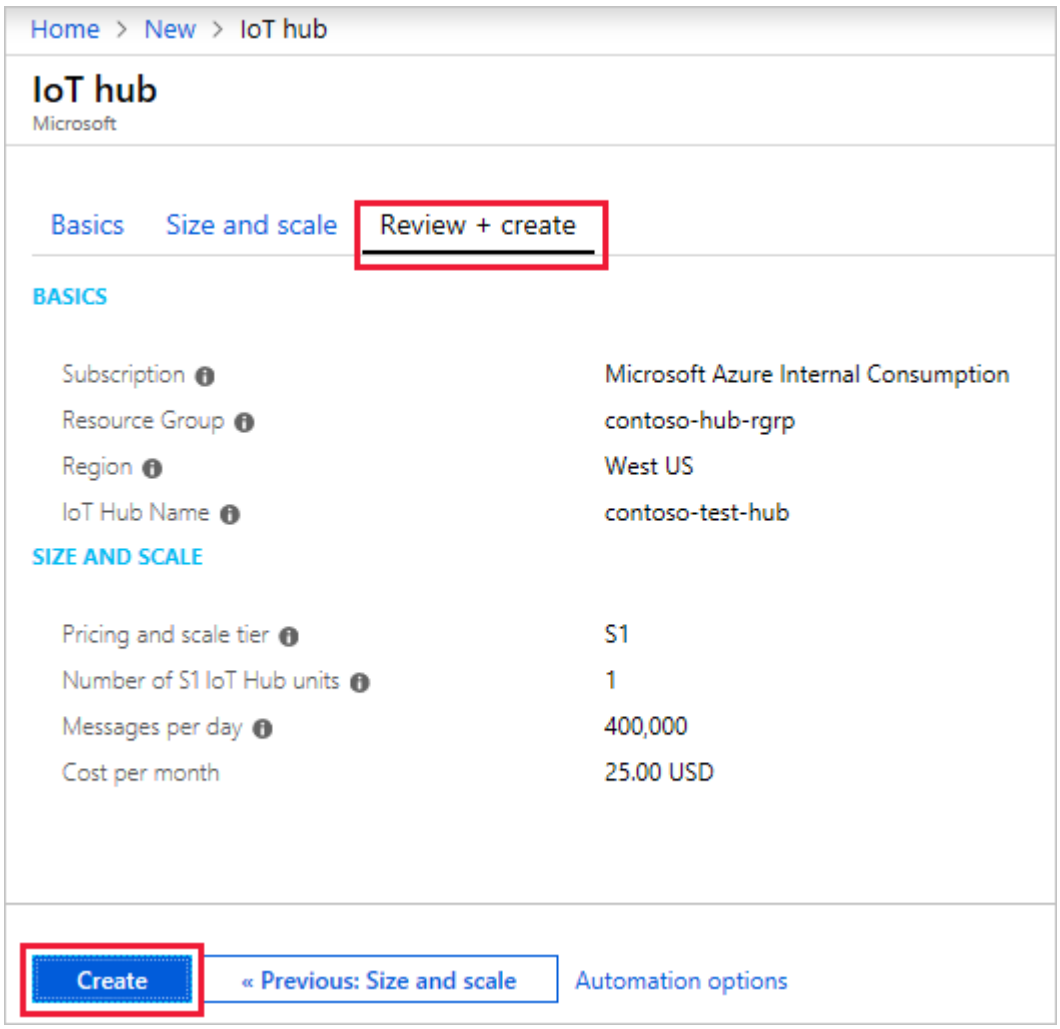


- **Pricing and scale tier**: You can choose from several tiers depending on how many features you want and how many messages you send through your solution per day. The free tier is intended for testing and evaluation. It allows 500 devices to be connected to the IoT hub and up to 8,000 messages per 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 device-to-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.

Reference

For more references, please visit Microsoft Documentations - Create an IoT hub using the Azure portal.

Register Iot 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>
   ```

   YourIoTHubName: 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
   ```

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

   > **Note**
   >
   > 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 quickstart.

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
   ```

   YourIoTHubName: 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;SharedAccessKeyName=service;SharedAccessKey={YourSharedAccessKey}`. 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.

2. 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
   ```

   Python2:

   ```
   brew install boost-python
   ```

   Python3:

   ```
   brew install boost-python3
   ```

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.

> **Note**
>
> Make a note of the device connection string, which looks like `HostName=<YourIoTHubName>.azure-devices.net;DeviceId=<NameOfYourDevice>;SharedAccessKey={YourSharedAccessKey}`.

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

   Python2:

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

   Python3:

   ```
   pip3 install azure-iothub-device-client
   ```

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

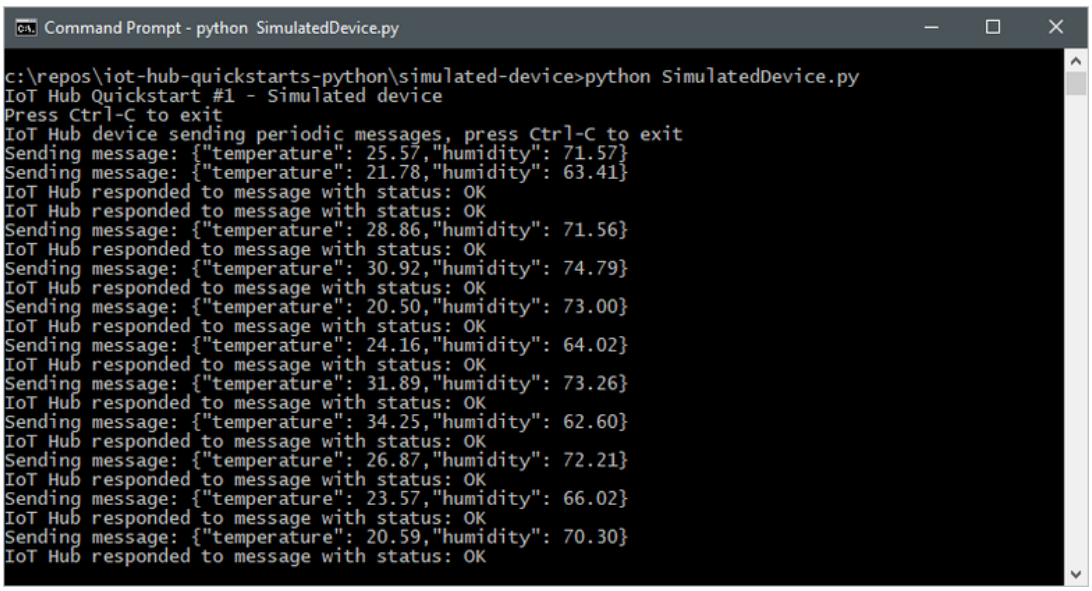
   Python2:

   ```
   python SimulatedDevice.py
   ```

   Python3:

   ```
   python3 SimulatedDevice.py
   ```

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



> **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: `pip install azure-iothub-device-client` .

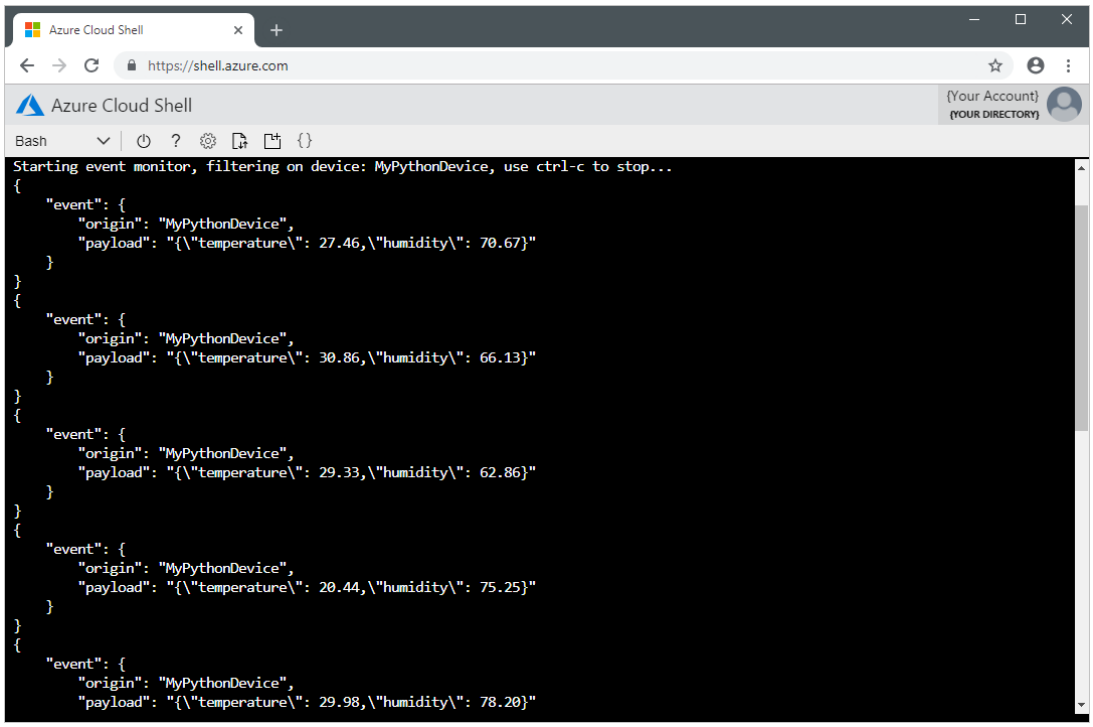
Read telemetry from Cloud

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/

Comments

Install Azure CLI with apt

If you are running a distribution that comes with `apt`, such as Ubuntu or Debian, there's an x86_64 package available for the Azure CLI. This package has been tested with and is supported for:

- Ubuntu trusty, xenial, artful, bionic, and disco
- Debian wheezy, jessie, and stretch

> **Note**
>
> The package for Azure CLI installs its own Python interpreter, and does not use the system Python.

Install

We offer two ways to install the Azure CLI with distributions that support `apt`: As an all-in-one script that runs the install commands for you, and instructions that you can run as a step-by-step process on your own.

Install with one command

We offer and maintain a script which runs all of the installation commands in one step. Run it by using `curl` and pipe directly to `bash`, or download the script to a file and inspect it before running.

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

> **Note**
>
> This script is only verified for Ubuntu 16.04+ and Debian 8+. It may not work on other distributions. If you're using a derived distribution such as Linux Mint, follow the manual install instructions and perform any necessary troubleshooting.

Manual install instructions

If you don't want to run a script as superuser, follow these manual steps to install the Azure CLI.

1. Get packages needed for the install process:

   ```
   sudo apt-get update
   sudo apt-get install curl apt-transport-https lsb-release gnupg
   ```

2. Download and install the Microsoft signing key:

   ```
   curl -sL https://packages.microsoft.com/keys/microsoft.asc | \
   gpg --dearmor | \
   sudo tee /etc/apt/trusted.gpg.d/microsoft.asc.gpg > /dev/null
   ```

3. Add the Azure CLI software repository:

   ```
   AZ_REPO=$(lsb_release -cs)
   echo "deb [arch=amd64] https://packages.microsoft.com/repos/
   azure-cli/ $AZ_REPO main" | \
   sudo tee /etc/apt/sources.list.d/azure-cli.list
   ```

4. Update repository information and install the `azure-cli` package:

   ```
   sudo apt-get update
   sudo apt-get install azure-cli
   ```

Run the Azure CLI with the `az` command. To sign in, use the `az login` command.

To learn more about different authentication methods, see Sign in with Azure CLI.

Troubleshooting

Here are some common problems seen when installing with `apt` . If you experience a problem not covered here, file an issue on github.

lsb_release does not return the correct base distribution version

Some Ubuntu- or Debian-derived distributions such as Linux Mint may not return the correct version name from `lsb_release`. This value is used in the install process to determine the package to install. If you know the code name of the Ubuntu or Debian version your distribution is derived from, you can set the `AZ_REPO` value manually when adding the repository. Otherwise, look up information for your distribution on how to determine the base distribution code name and set `AZ_REPO` to the correct value.

No package for your distribution

Sometimes it may be a while after a distribution is released before there's an Azure CLI package available for it. The Azure CLI designed to be resilient with regards to future versions of dependencies and rely on as few of them as possible. If there's no package available for your base distribution, try a package for an earlier distribution.

To do this, set the value of `AZ_REPO` manually when adding the repository. For Ubuntu distributions use the `bionic` repository, and for Debian distributions use `stretch`. Distributions released before Ubuntu Trusty and Debian Wheezy are not supported.

Proxy blocks connection

If you're unable to connect to an external resource due to a proxy, make sure that you've correctly set the HTTP_PROXY and HTTPS_PROXY variables in your shell. You will need to contact your system administrator to know what host(s) and port(s) to use for these proxies.

These values are respected by many Linux programs, including those which are used in the install process. To set these values:

No auth

```
export HTTP_PROXY=http://[proxy]:[port]
export HTTPS_PROXY=https://[proxy]:[port]
```

Basic auth

```
export HTTP_PROXY=http://[username]:[password]@[proxy]:[port]
export HTTPS_PROXY=https://[username]:[password]@[proxy]:[port]
```

> **Note**
>
> If you are behind a proxy, these shell variables must be set to connect to Azure services with the CLI. If you are not using basic auth, it's recommended to export these variables in your .bashrc file. Always follow your business' security policies and the requirements of your system administrator.

You may also want to explicitly configure `apt` to use this proxy at all times. Make sure that the following lines appear in an `apt` configuration file in `/etc/apt/apt.conf.d/`. We recommend using either your existing global configuration file, an existing proxy configuration file, `40proxies`, or `99local`, but follow your system administration requirements.

```
Acquire {
    http::proxy "http://[username]:[password]@[proxy]:[port]";
    https::proxy "https://[username]:[password]@[proxy]:[port]";
}
```

If your proxy does not use basic auth, **remove** the `[username]:[password]@` portion of the proxy URI. If you require more information for proxy configuration, see the official Ubuntu documentation:

- apt.conf manpage
- Ubuntu wiki - apt-get howto

In order to get the Microsoft signing key and get the package from our repository, your proxy needs to allow HTTPS connections to the following address:

https://packages.microsoft.com

CLI fails to install or run on Windows Subsystem for Linux

Since Windows Subsystem for Linux (WSL) is a system call translation layer on top of the Windows platform, you might experience an error when trying to install or run the Azure CLI. The CLI relies on some features that may have a bug in WSL. If you experience an error no matter how you install the CLI, there's a good chance it's an issue with WSL and not with the CLI install process.

To troubleshoot your WSL installation and possibly resolve issues:

- If you can, run an identical install process on a Linux machine or VM to see if it succeeds. If it does, your issue is almost certainly related to WSL. To start a Linux VM in Azure, see the create a Linux VM in the Azure Portal documentation.
- Make sure that you're running the latest version of WSL. To get the latest version, update your Windows 10 installation.
- Check for any open issues with WSL which might address your problem. Often there will be suggestions on how to work around the problem, or information about a release where the issue will be fixed.
- If there are no existing issues for your problem, file a new issue with WSL and make sure that you include as much information as possible.

If you continue to have issues installing or running on WSL, consider installing the CLI for Windows.

Update

Use `apt-get upgrade` to update the CLI package.

```
sudo apt-get update && sudo apt-get upgrade
```

> **Note**
>
> This command upgrades all of the installed packages on your system that have not had a dependency change. To upgrade the CLI only, use `apt-get install`.
>
> ```
> sudo apt-get update && sudo apt-get install --only-upgrade -y azure-cli
> ```

Uninstall

1. Uninstall with `apt-get remove`:

   ```
   sudo apt-get remove -y azure-cli
   ```

2. If you don't plan to reinstall the CLI, remove the Azure CLI repository information:

   ```
   sudo rm /etc/apt/sources.list.d/azure-cli.list
   ```

3. Remove the signing key:

   ```
   sudo rm /etc/apt/trusted.gpg.d/microsoft.asc.gpg
   ```

4. Remove any unneeded packages:

   ```
   sudo apt autoremove
   ```

> **References**
>
> For more references, please refer to Microsoft Documentation - Install Azure CLI with apt

Comments

Install Azure CLI on macOS

For the macOS platform, you can install the Azure CLI with homebrew package manager. Homebrew makes it easy to keep your installation of the CLI update to date. The CLI package has been tested on macOS versions 10.9 and later.

Install

Homebrew is the easiest way to manage your CLI install. It provides convenient ways to install, update, and uninstall. If you don't have homebrew available on your system, install homebrew before continuing.

You can install the CLI by updating your brew repository information, and then running the `install` command:

```
brew update && brew install azure-cli
```

> The Azure CLI has a dependency on the `python3` package in Homebrew, and will install it on your system, even if Python 2 is available. The Azure CLI is guaranteed to be compatible with the latest version of `python3` published on Homebrew.

You can then run the Azure CLI with the `az` command. To sign in, use `az login` command.

To learn more about different authentication methods, see Sign in with Azure CLI.

Troubleshooting

If you encounter a problem when installing the CLI through Homebrew, here are some common errors. If you experience a problem not covered here, file an issue on github.

Unable to find Python or installed packages

There may be a minor version mismatch or other issue during homebrew installation. The CLI doesn't use a Python virtual environment, so it relies on finding the installed Python version. A possible fix is to install and relink the `python3` dependency from Homebrew.

```
brew update && brew install python3 && brew upgrade python3
brew link --overwrite python3
```

CLI version 1.x is installed

If an out-of-date version was installed, it could be because of a stale homebrew cache. Follow the update instructions.

Proxy blocks connection

You may be unable to get resources from Homebrew unless you have correctly configured it to use your proxy. Follow the Homebrew proxy configuration instructions.

> If you are behind a proxy, `HTTP_PROXY` and `HTTPS_PROXY` must be set to connect to Azure services with the CLI. If you are not using basic auth, it's recommended to export these variables in your `.bashrc` file. Always follow your business' security policies and the requirements of your system administrator.

In order to get the bottle resources from Homebrew, your proxy needs to allow HTTPS connections to the following addresses:

- `https://formulae.brew.sh`
- `https://homebrew.bintray.com`

Update

The CLI is regularly updated with bug fixes, improvements, new features, and preview functionality. A new release is available roughly every two weeks. Update your local repository information and then upgrade the `azure-cli` package.

```
brew update && brew upgrade azure-cli
```

Uninstall

Use homebrew to uninstall the `azure-cli` package.

```
brew uninstall azure-cli
```

Other installation methods

If you can't use homebrew to install the Azure CLI in your environment, it's possible to use the manual instructions for Linux. Note that this process is not officially maintained to be compatible with macOS. Using a package manager such as Homebrew is always recommended. Only use the manual installation method if you have no other option available.

For the manual installation instructions, see Install Azure CLI on Linux manually.

Comments

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.

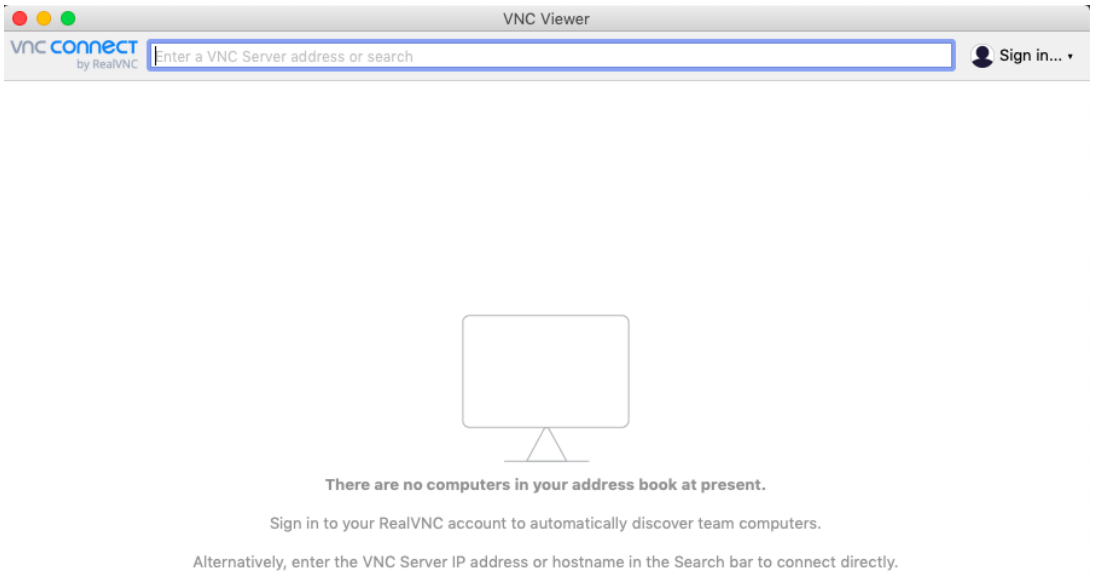
Getting IP address

First of all, we will need IP address of Raspberry Pi. Power up your Raspberry Pi with NB-IoT Hat attached. Then access to website https://maxis-challenge.firebaseapp.com/ to obtain your Raspberry Pi Information by referring IMEI from NB-IoT HAT.

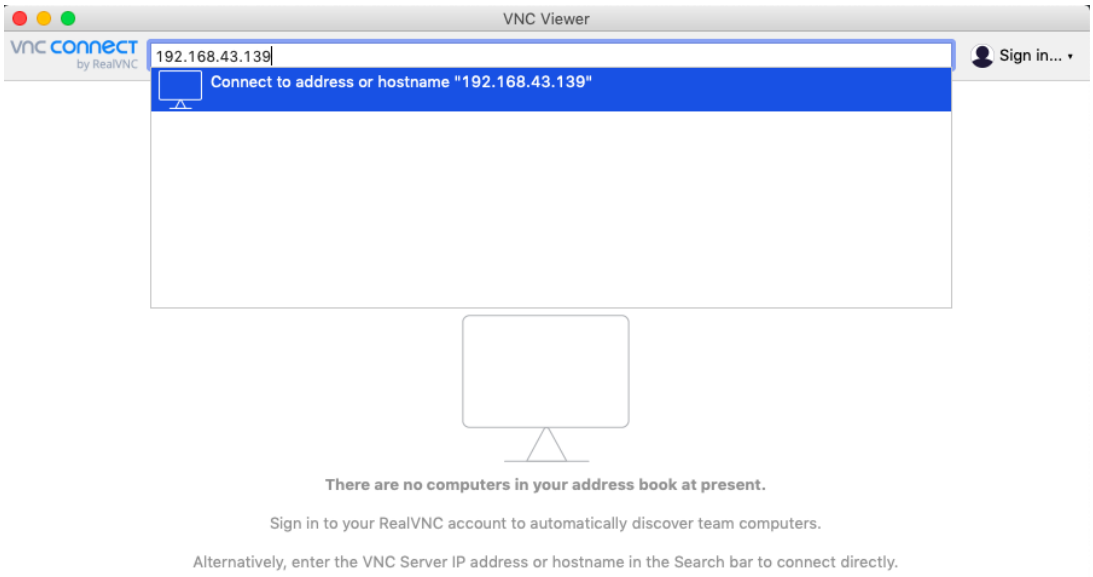
VNC

1. Download VNC Viewer from here. Choose appropriate installer for your PC which will remote access to Raspberry Pi.

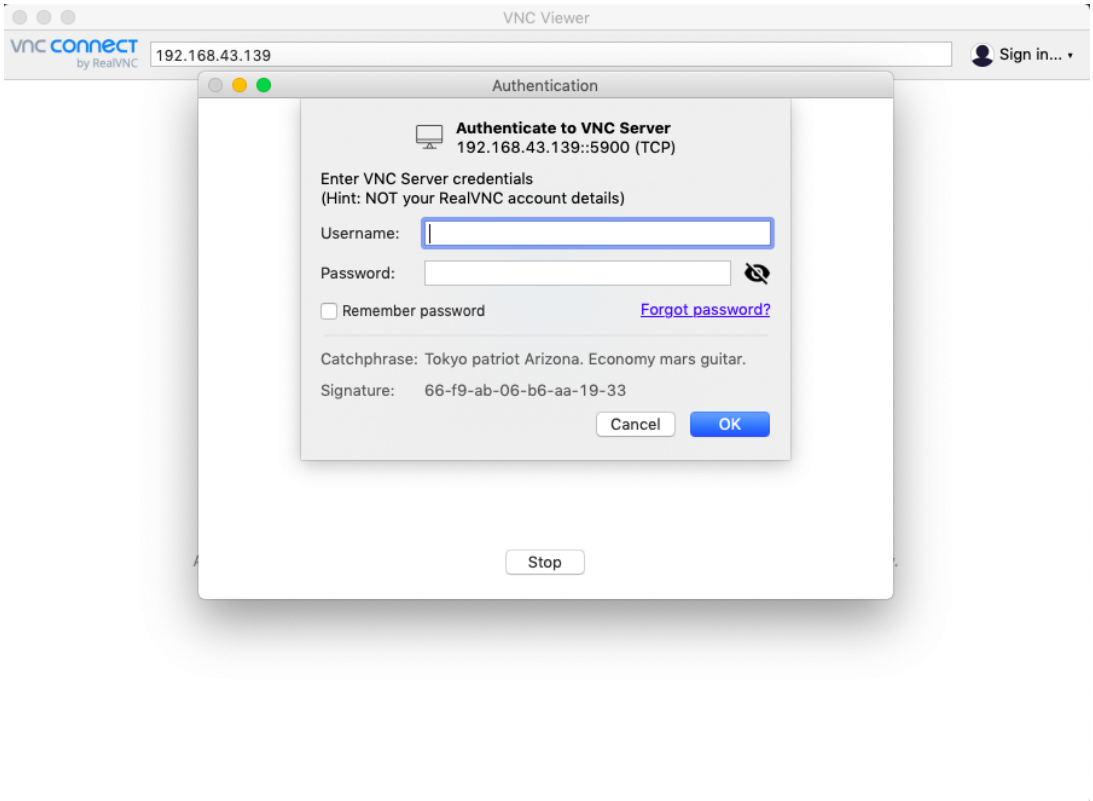
2. Open VNC Viewer.



3. Enter Raspberry Pi IP Address and Press Enter Button.



4. Enter username and password for Raspberry Pi. By default, username is `pi` and password is `raspberry`.

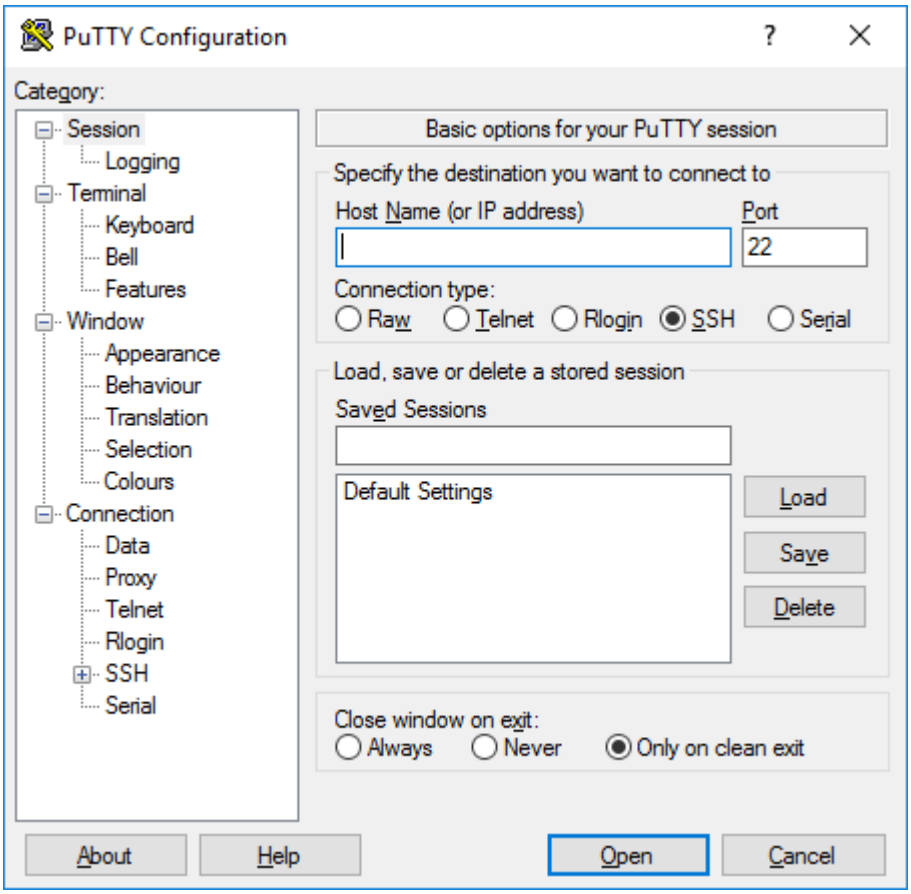


5. Viola! Now you can access to Rasberry Pi remotely.

SSH

- Windows
    a. Download Putty software from here.
    b. Open downloaded Putty.exe.

c. Select SSH and enter your Raspberry Pi IP Address.

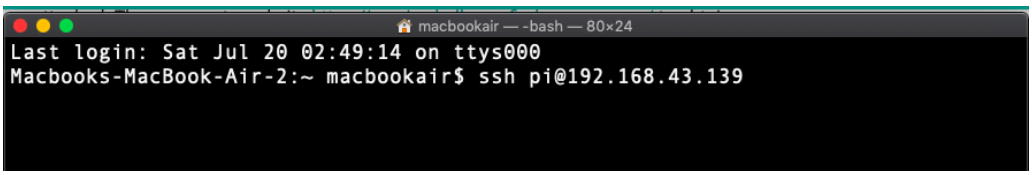


d. Press `Open` button to open SSH Terminal.

e. When `login as` appears, enter `pi`, we are going to login as pi.

f. By default the password is `raspberry`.

- Mac OS, Linux
  - Run command `ssh pi@<youripaddress>` in local terminal.



  - Enter password `raspberry`.

Comments

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 reboot
```

Firmware Update (Important)

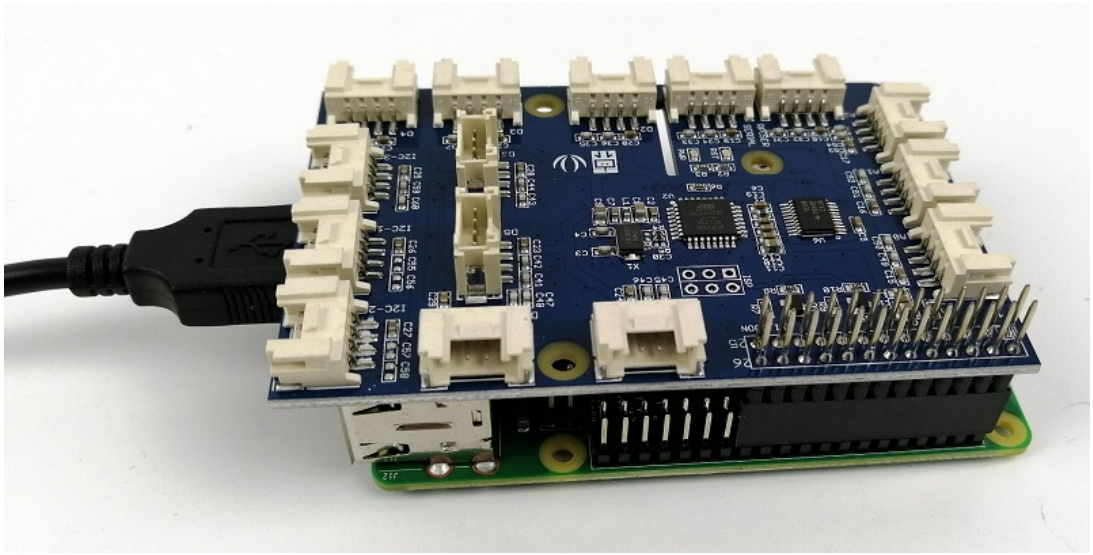
```
$ cd Dexter/GrovePi/Firmware
$ sudo chmod +x firmware_update.sh
$ sudo bash firmware_update.sh
```

Getting started with Example DHT Temperature & Humidity Sensor

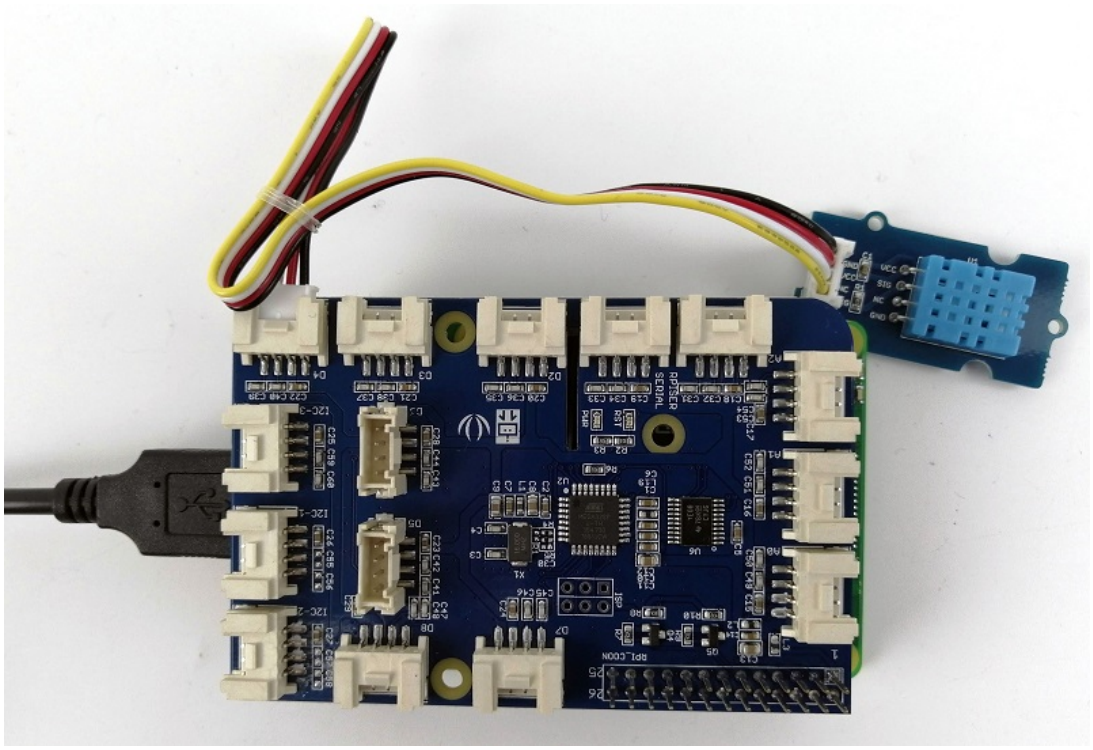
Sounds familiar? Temperature and humidity are the values that are being sent to Azure Cloud if you are following previous course. Right now we can send those values from actual hardware DHT11 Temperature & Humidity Sensor.

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
from time import sleep
# 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.

try:
    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))
            sleep(1)

        except IOError:
            print ("Error")

except KeyboardInterrupt:
    print ("Pressed Ctrl+C. Exiting now..")
```

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

1. 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

Comments

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

NB-IoT AT command testing

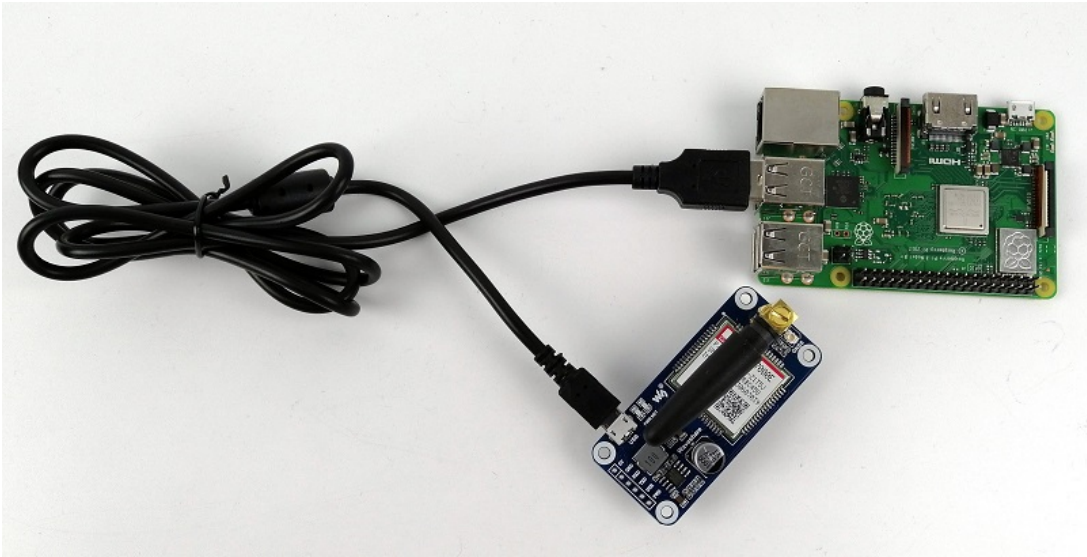
AT command test is basic step to check if NB-IoT is working in good condition.

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
OK

# 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
OK

-> AT+CNMP=38
OK

-> AT+CBANDCFG="NB-IOT",8
OK
```

```
-> AT+CSTT="M2MXNBIOT"
OK

-> 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
OK

STATE: IP START

# start establish connection
-> AT+CIICR
OK

-> AT+CIPSTATUS
OK

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

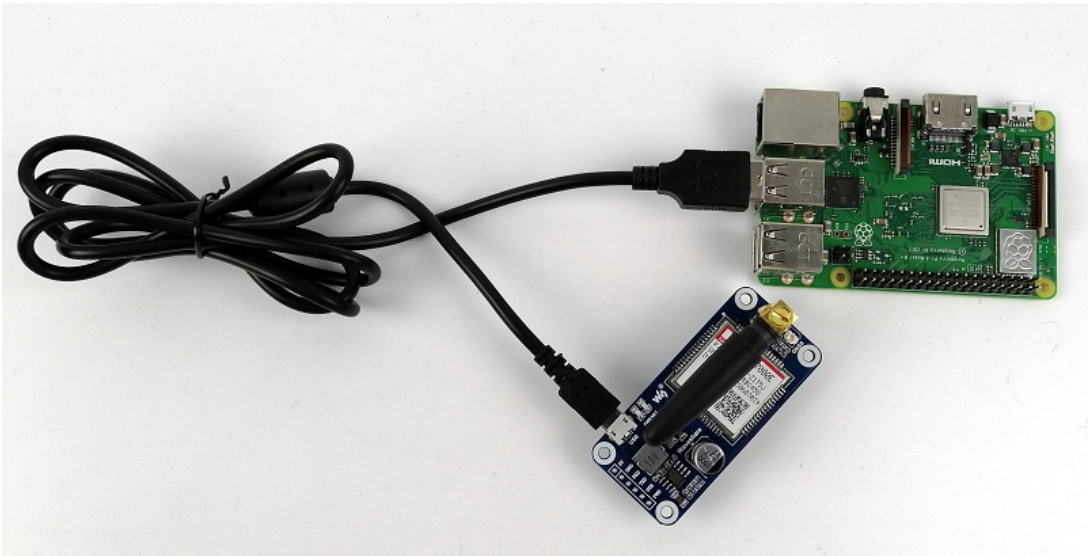
NB-IoT HAT offers PPP (Point-to-Point Protocol) for us to connect Raspberry Pi to Internet easily. The following steps show how to establish Internet connection with Raspberry Pi.

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.

```
$ cd "Raspberry Pi/PPP Installer"
$ sudo chmod +x install.sh
$ sudo ./install.sh
```

5. Run command below to start the connection.

```
sudo systemctl start nbiot.service
```

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.

```
sudo systemctl stop nbiot.service
```

9. To automate the connection during Raspberry Pi boot-up, run command `sudo systemctl enable nbiot.service`. To see the effect, run `sudo reboot` to restart Raspberry Pi.

References

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

Comments

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
   sudo apt-get install python-dev
   ```

   Python3:

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

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 and OpenSSL library are installed.

```
sudo apt-get install libboost-python-dev libcurl4-openssl-dev
```

5. Navigate to the root folder of the downloaded sample Python project. Then navigate to the **iot-hub\Quickstarts\simulated-device** folder.
6. 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.

   > **Note**
   >
   > Make a note of the device connection string, which looks like `HostName=<YourIoTHubName>.azure-devices.net;DeviceId=<NameOfYourDevice>;SharedAccessKey={YourSharedAccessKey}`.

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

   Python2:

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

   Python3:

   ```
   sudo pip3 install azure-iothub-device-client
   ```

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

   Python2:

   ```
   sudo python SimulatedDevice.py
   ```

   Python3:

   ```
   sudo python3 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
```

Comments

Global Navigation Satellite Systems (GNSS)

Example Python application to extract GNSS data from NB-IoT HAT

Prerequisite

Install Python Library **micropyGPS**

Python2:

```
sudo pip install git+https://github.com/inmcm/micropyGPS.git
```

Python3:

```
sudo pip3 install git+https://github.com/inmcm/micropyGPS.git
```

Python Example Code

```
import serial, sys
from time import sleep, time
from micropyGPS import MicropyGPS

NMEA_PORT='/dev/ttyUSB1'
AT_PORT='/dev/ttyUSB2'

def initGNSS():
    sys.stdout.write('Init GNSS\n')
    with serial.Serial(AT_PORT, 115200, timeout=5, rtscts=True,
dsrdtr=True) as ser:
        ser.write("AT+CGNSCFG=1\r\n".encode())
        sleep(1)
        ser.write("AT+CGNSPWR=1\r\n".encode())
        sleep(1)

try:
    sys.stdout.write('Started GNSS reader\n')
```

```
    reader = MicropyGPS()
    initGNSS()
    while True:
        try:
            with serial.Serial(NMEA_PORT, 115200, timeout=5,
rtscts=True, dsrdtr=True) as ser:
                ts = 0
                while True:
                    # update
                    data = ser.read().decode()
                    reader.update(data)
                    # read in every 2 seconds
                    if time() - ts > 2:
                        ts = time()
                        print("UTC_Date={}, UTC_Time={}, lat={},
lng={}".format(reader.date_string(), reader.timestamp,
reader.latitude_string(), reader.longitude_string()))

        except Exception as e:
            print("Error: {}".format(str(e)))
            sleep(1)

except KeyboardInterrupt:
    sys.stderr.write('Ctrl-C pressed, exiting GNSS reader\n')
```

Comments

IoT Web Application using Node Red

Running Node Red on your Local Machine

1. 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 `Ctrl+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

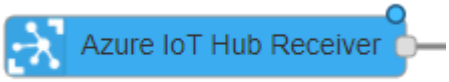
In this application, we are going to learn to:

- 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.

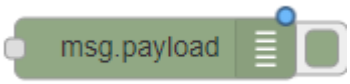
Read Telemetry from Azure IoT Hub

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

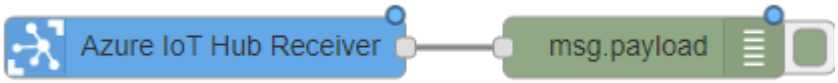
2. Install `node-red-contrib-azure-iot-hub` via **Manage Palette**. Manage Palette can be found under dropdown menu from menu button at top right corner of browser window.
3. From left pane of the browser window, scroll and search for **Azure IoT Hub Receiver** node under **Cloud** category, then drag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



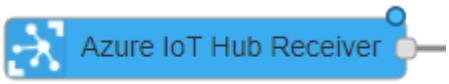
4. From the same left pane under **output** category, drag **debug** node to the dashboard. (It will be renamed as msg.payload once it is placed at dashboard)

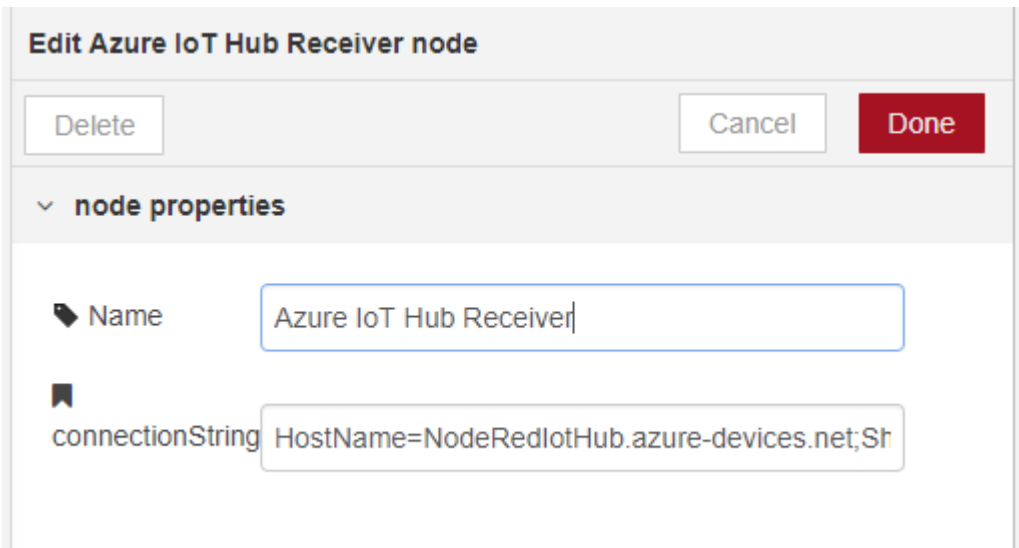


5. Link it to **Azure IoT Hub Receiver** node.



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

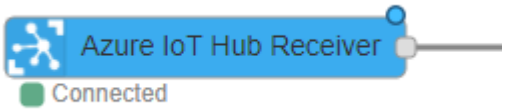




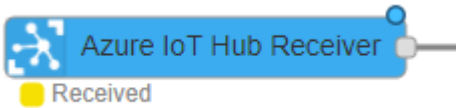
7. Click **Deploy**.

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!
```



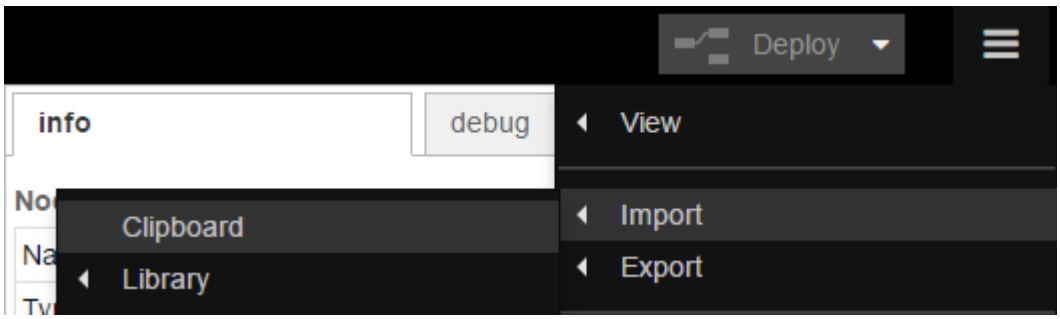
9. Once you have messages coming into your Azure IoT Hub, you should see them in the debug pane. The Azure IoT Hub Receiver node should now say 'Received'.



Create a Web UI to display device telemetry

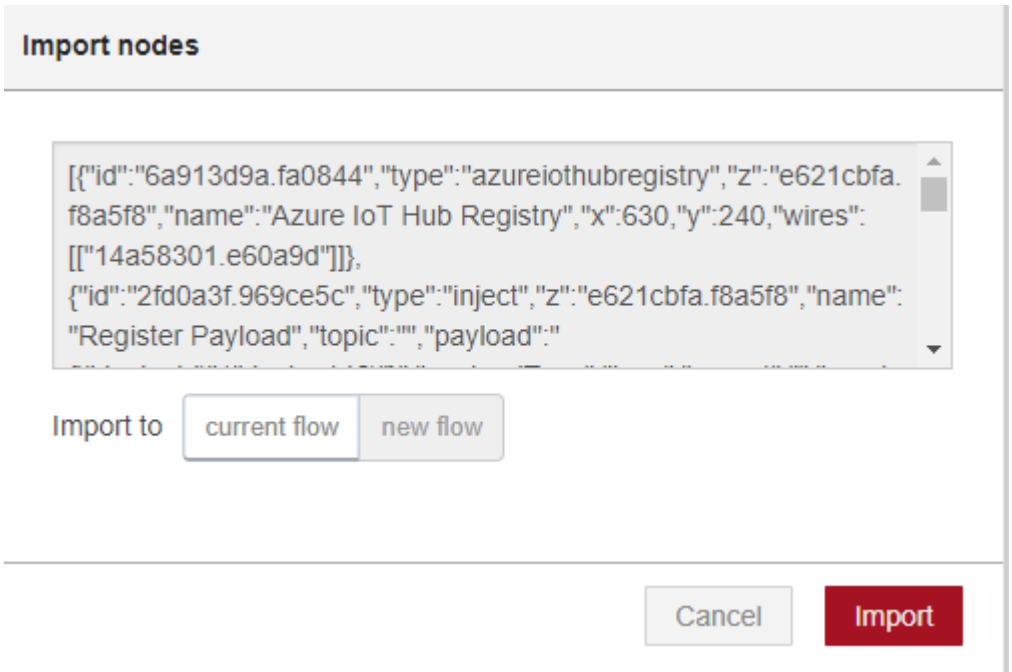
We are going to create 2 widgets which show temperature and humidity of the sensor.

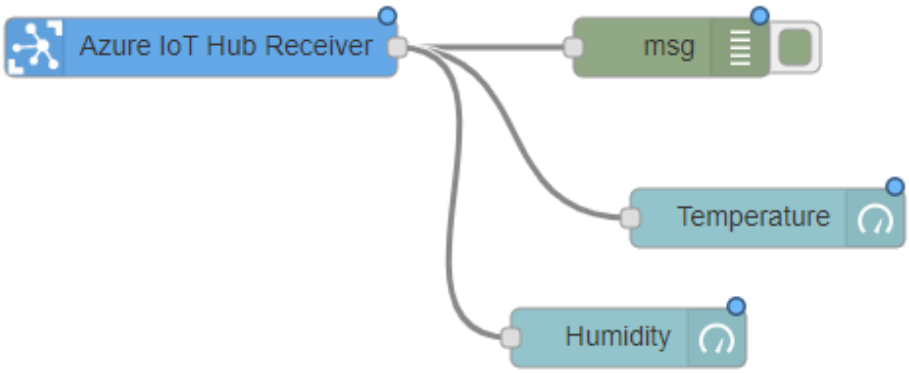
1. Install `node-red-dashboard` using **Manage Palette**.
2. Go to Hamburger Menu -> Import -> Clipboard.



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"],"seg1":"","seg2":"","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}]
```

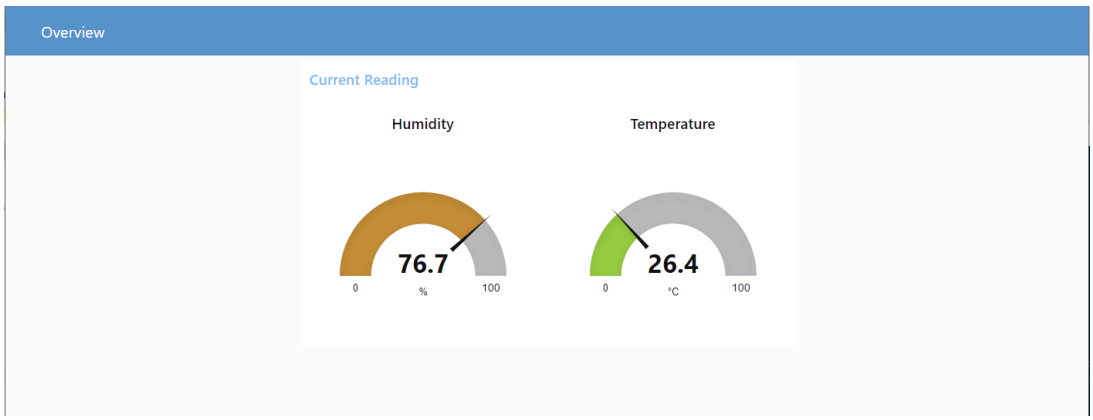





> **Important !!**
>
> **Import flow including Azure IoT Hub Receiver node does not include connectionString by default. You will have to provide connectionString again.** Double-click on the Azure IoT Hub Receiver node and enter your Iot Hub connectionString for your Azure IoT Hub and click Done.

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!

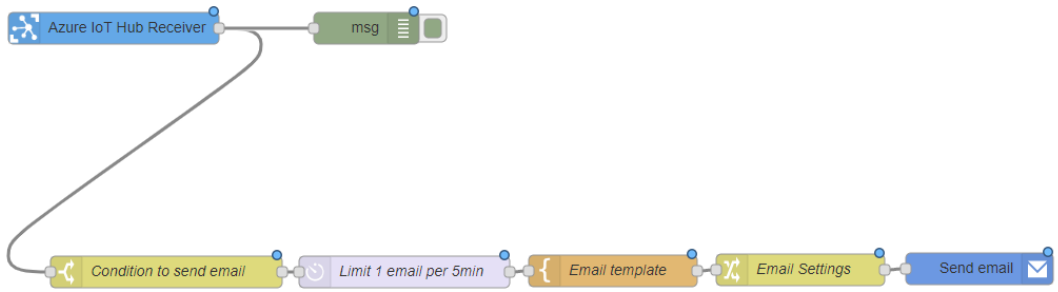
Send an email when alert is triggered

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.



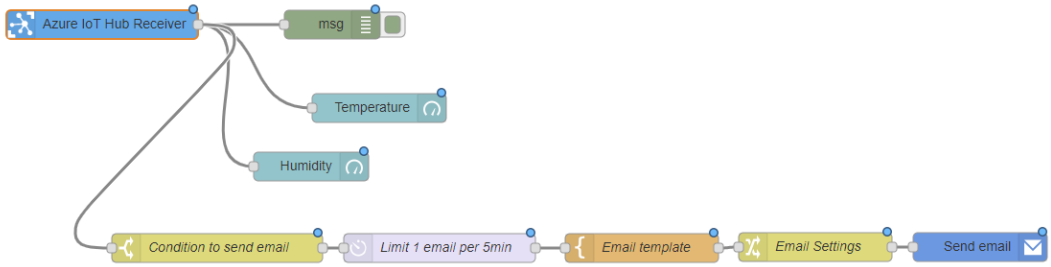
3. Get API key from SendGrid website.
4. Double click on **Send email** node (which is a sendgrid node), fill in necessary info and paste the API key.
5. Click **Deploy**.

Explanation:

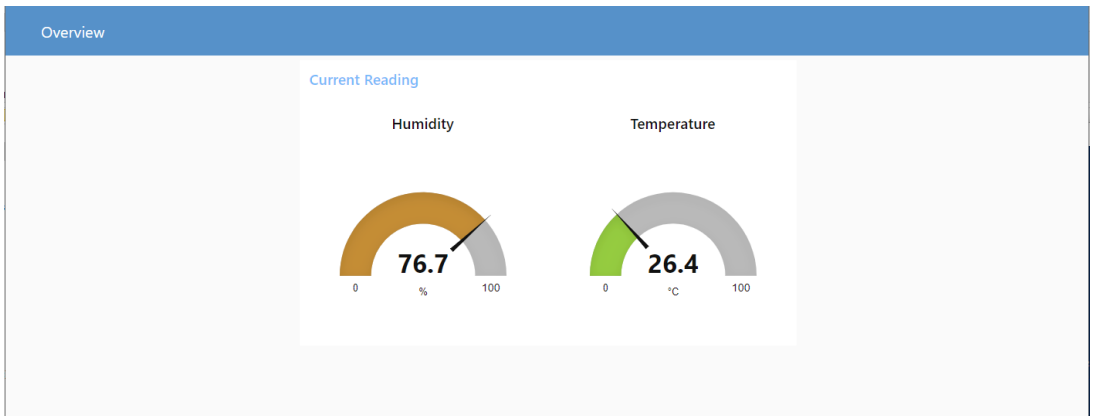
- Condition to send email is handled by `Condition to send email` node. When the temperature exceeds 30°C, the email will be sent.
- Email has a cooldown period of 5 minutes, during this time the email will not be sent although the temperature exceeds 30°C. This is handled by `Limit 1 email per 5 min` 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":"Overview",
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.

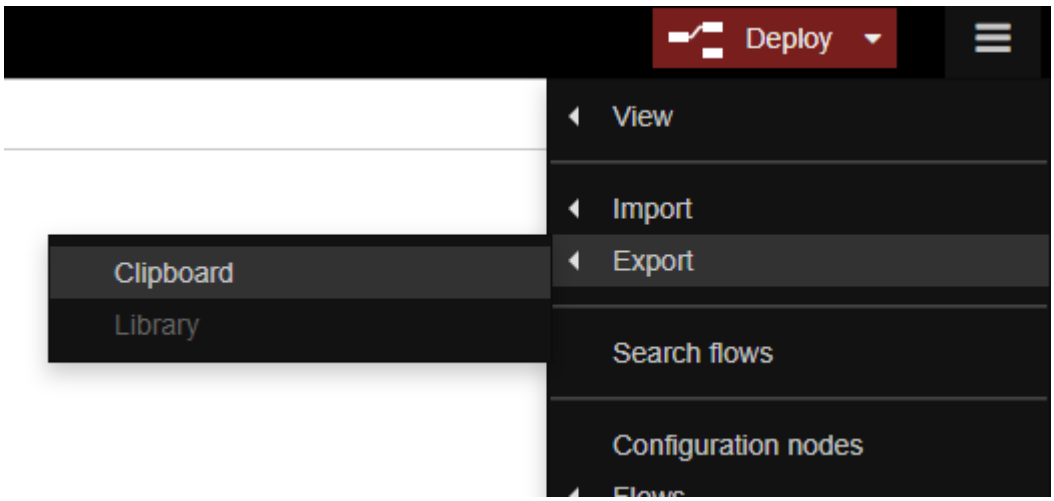
You can look into https://github.com/Azure/node-red-contrib-azure for more libraries and modules for Azure. It is up to your creativity. The other libraries you may find useful for your application:

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

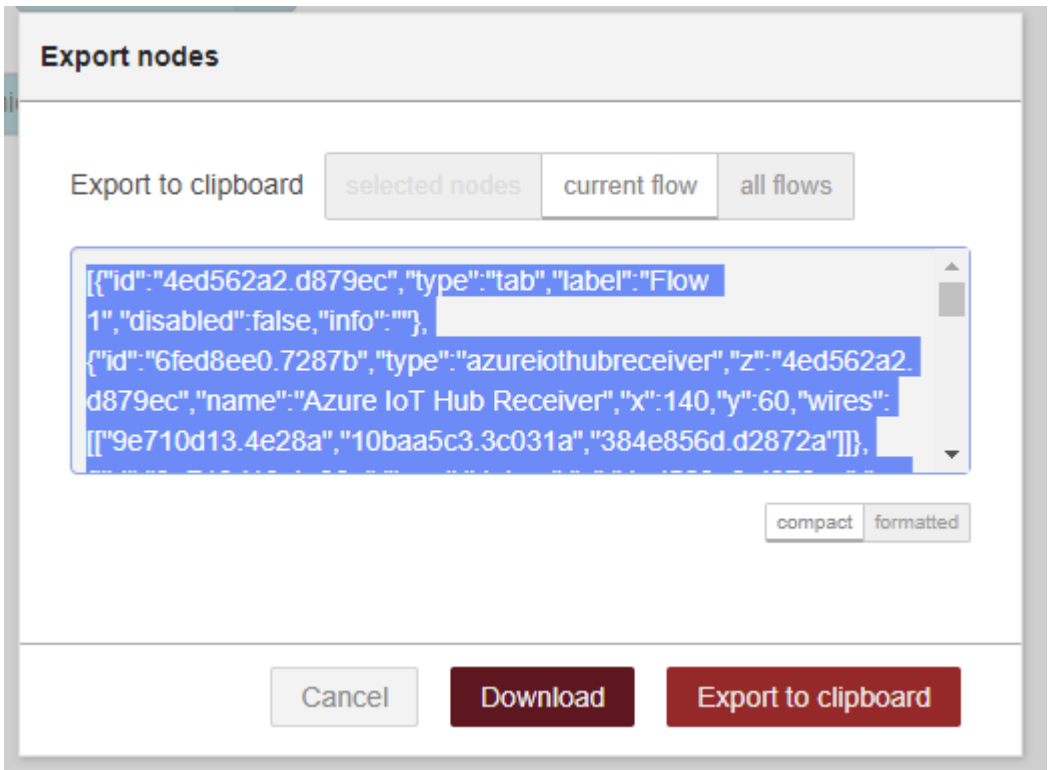
Save your work!!!

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 node-red 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.

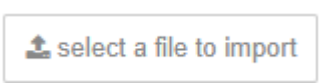
5. Once started, you can access the editor at `http://<your-instance-ip>:1880/`.
6. To get Node-RED to start automatically whenever your instance is restarted, you can use pm2:

```
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.



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

Comments

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/maxis-nbiot-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`

Comments