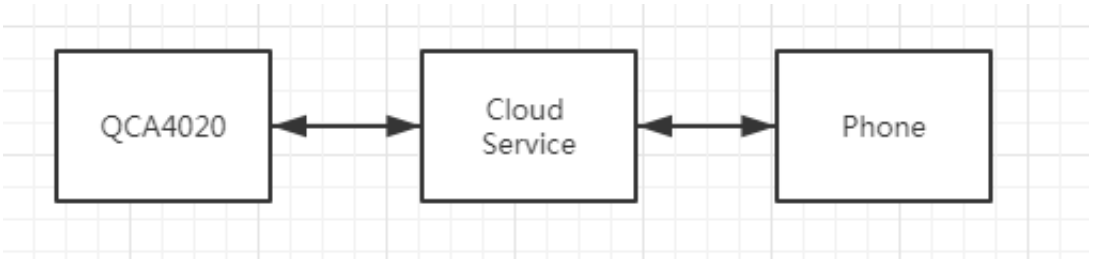
1 Preface

Remote control implementation of ZigBee Lamp can be divided into Trhee aspects: mobile terminal development, Azure cloud service configuration,and QCA4020 board application development.Therefore, the following three aspects are introduced in detail.

2 The overall architecture of remote control



3 Cloud configuration

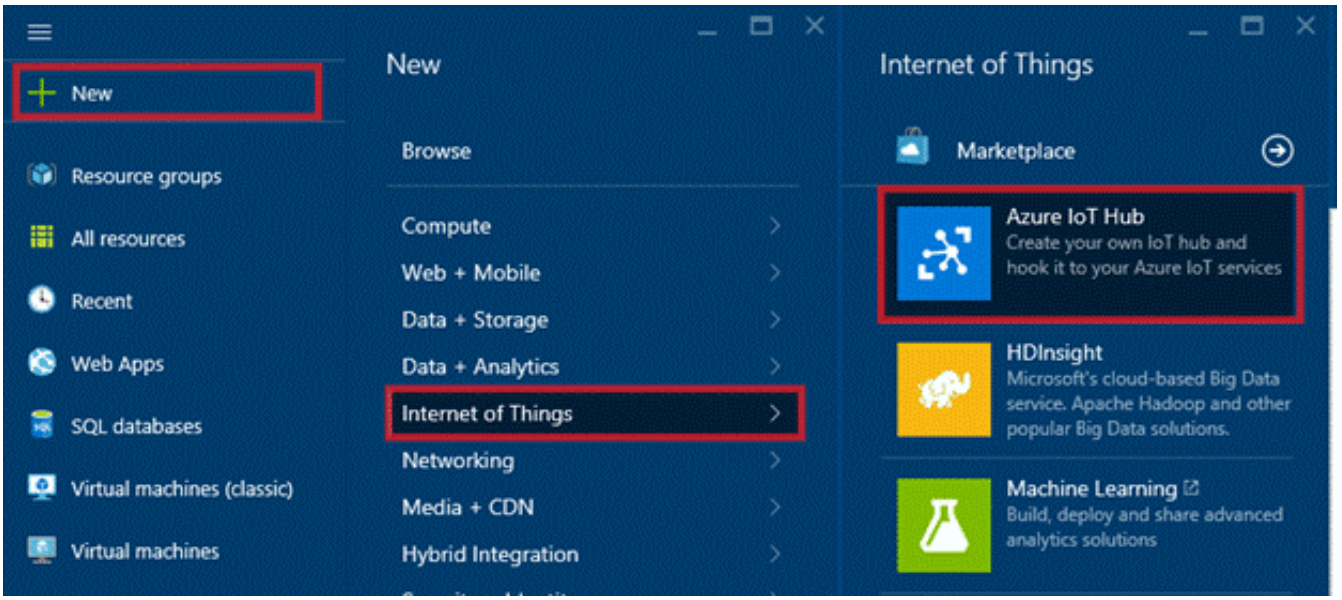
Azure IoT Hub is a fully managed service that enables reliable and secure bi-directional communications between millions of IoT devices and an application back end. Azure IoT Hub offers reliable device-to-cloud and cloud-to-device hyper-scale messaging, enables secure communications using per-device security credentials and access control, and includes device libraries for the most popular languages and platforms.

Before you can communicate with IoT Hub from a device you must create an IoT hub instance in your Azure subscription and then provision your device in your IoT hub. You must complete these steps before you try to run any of the sample IoT Hub device client applications in this repository (azure-iot-sdks).

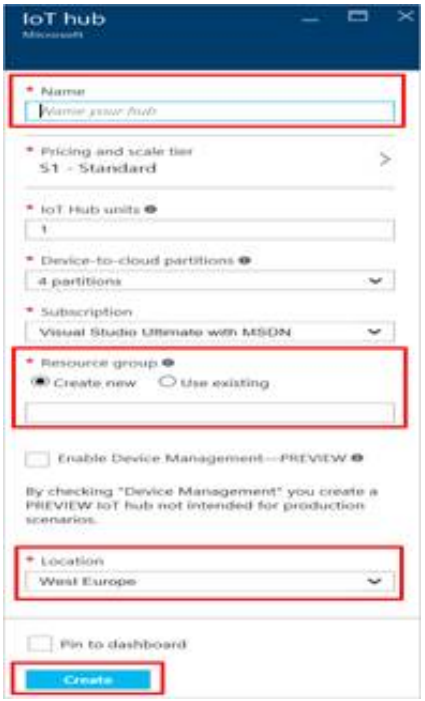
You can use the Azure Portal to create an IoT hub to use with your devices.

3.1 Log on to the Azure Portal.

3.2 In the jump bar, firstly click New, secondly click Internet of Things,lastly click Azure IoT Hub.



3.3 In the New IoT Hub blade, specify the desired configuration for the IoT Hub.



In the Name box, enter a name to identify your IoT hub. When the Name is validated, a green check mark appears in the Name box.

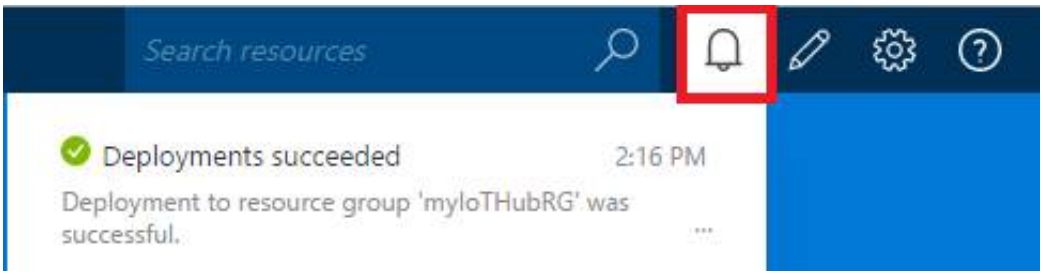
3.4 Change the Pricing and scale tier as desired

The getting started samples do not require a specific tier.

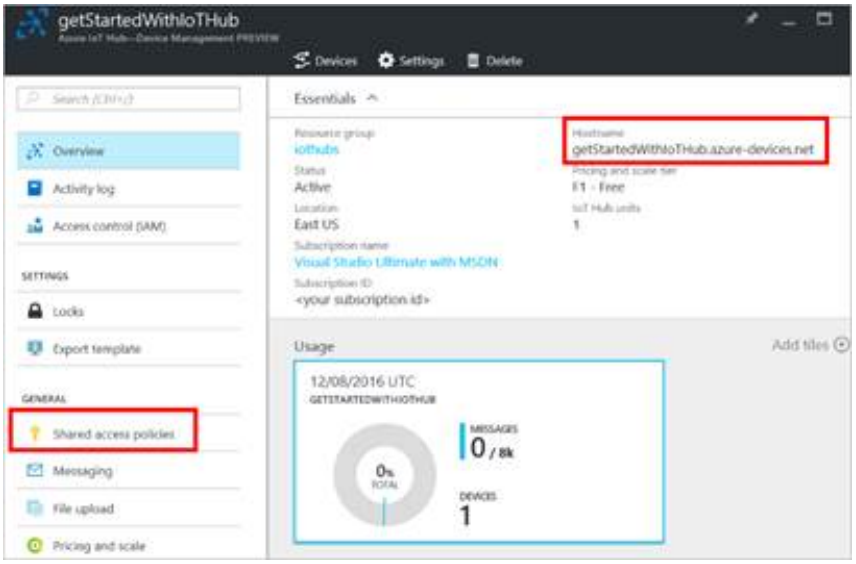
In the Resource group box, create a new resource group, or select and existing one. For more information, see Using resource groups to manage your Azure resources.

Use Location to specify the geographic location in which to host your IoT hub.

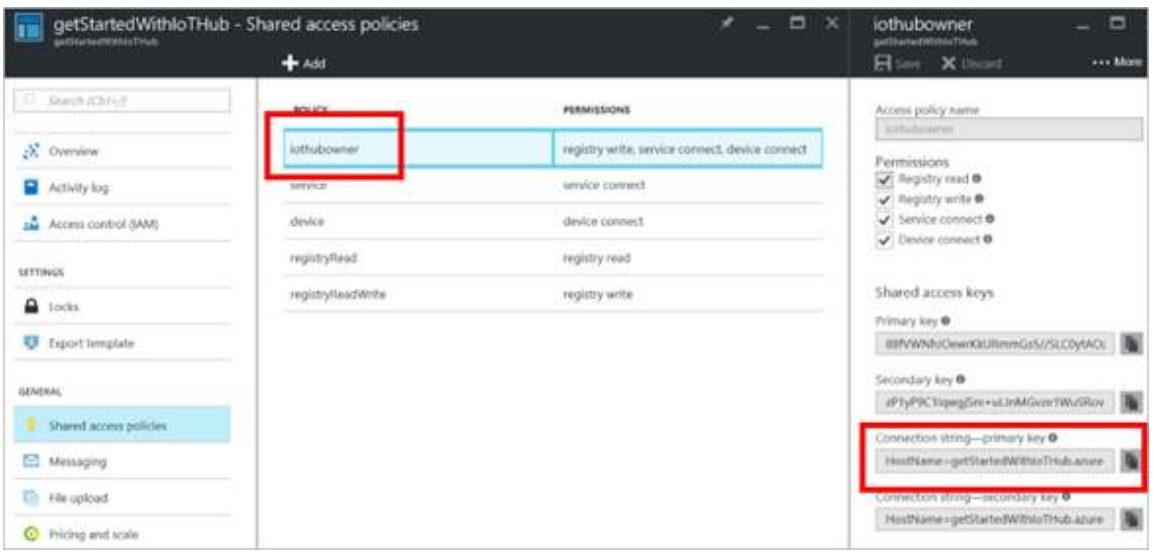
Once the new IoT hub options are configured, click Create. It can take a few minutes for the IoT hub to be created. To check the status, you can monitor the progress on the Start board. Or, you can monitor your progress from the Notifications section.



3.5 After the IoT hub has been created successfully, open the blade of the new IoT hub, take note of the host name URI, and click Shared access policies.



3.6 Select the Shared access policy called iothubowner, then copy and take note of the connection string on the right blade.



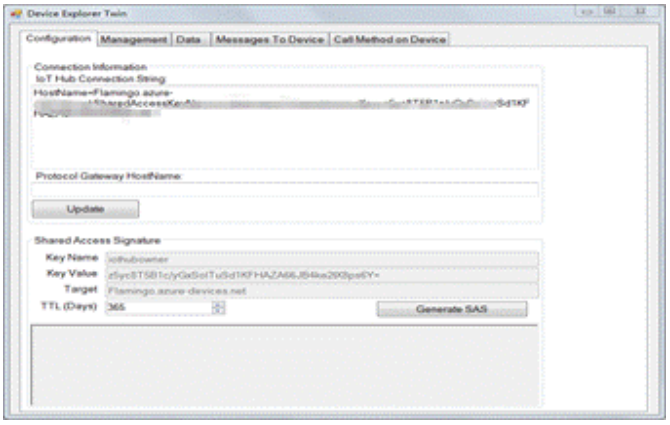
Your IoT hub is now created, and you have the connection string you need to use the iothub-explorer or the Device Explorer tool. This connection string enables applications to perform management operations on the IoT hub such as adding a new device to the IoT hub.

3.7 Add devices to IoT Hub

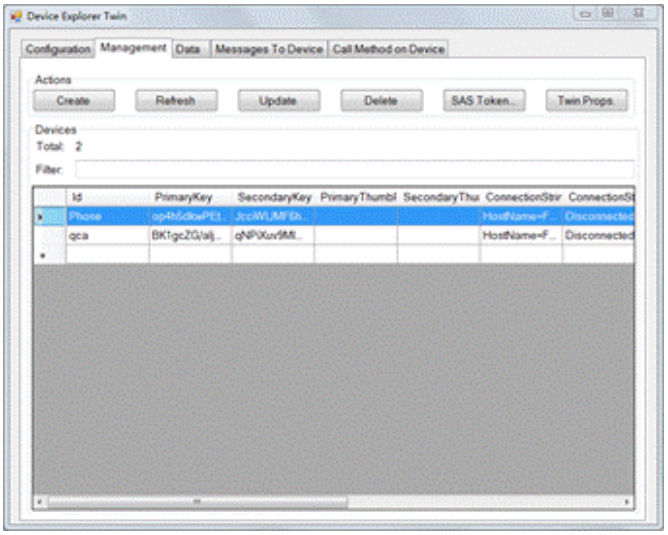
You must add details your device to IoT Hub before that device can communicate with the hub. When you add a device to an IoT hub, the hub generates the connection string that the device must use when it establishes the secure connection to the IoT hub.

To add a device to your IoT hub, you can use the iothub-explorer or the Device Explorer utility in this repository (azure-oit-sdks). These tools will generate a device specific connection string that you need to copy and paste in the source code of the application running on the device.

Iothub-explorer:



Create Devices:



After you Create Device you can copy connect string from iothub-explorer.

4.The QCA program receives information

4.1 Add the method to receive the message

Log based on QCLI_demo can see where our QCA program receives the information.

```
wlan enable
wlan SetWpaPassphrase 123456789
wlan SetWpaParameters WPA2 CCMP CCMP
wlan connect god

net dhcpv4c wlan0 new
net dnsc start
net dnsc addsrv 8.8.8.8

net sntpc start
net sntpc addsvr 24.56.178.140

ecosystem azure-simple HostName=Flamingo.azure-
devices.net;DeviceId=qca;SharedAccessKey=BK1gcZG/aljMRNaVkEWvMajJm8iXC
0rdY4R9WcVz1rM=
```

qca4020-or-2-0_qca_oem_sdk-cdb.git\target\quartz\demo\QCLI_demo\src\ecosystem\azure\simplesample_mqtt.c

Add the following code to the file method:

```
 QCLI_Printf(qcli_ecosystem_handle,"failed to malloc %s %d
%s",temp,size,temp[82]);

switch(temp[82]){
    case 49:
    QCLI_Printf(qcli_ecosystem_handle,"open zigbee light %s",temp);
    char* color=malloc(6);
    char* bright=malloc(5);
    char* six=malloc(4);
    six="0x";
    char* destination=malloc(10);

    strncpy(color,temp+65,4);

    strncpy(destination,six,4);
    strcat(destination,color);
```

```
    strncpy(bright,temp+51,3);
    long int_color=strtol(destination,NULL,16);

    int int_bringht=atoi(bright);
    QCLI_Printf(qcli_ecosystem_handle,"int_color and int_bright %d
%d",int_bringht,int_color);

    QCLI_Printf(qcli_ecosystem_handle,"color and bright %s
%s",destination,bright);
    break;
    case 50:
    QCLI_Printf(qcli_ecosystem_handle,"close zigbee light");
    break;
    case 51:
    QCLI_Printf(qcli_ecosystem_handle,"open qmesh light");
    break;
    case 52:
    QCLI_Printf(qcli_ecosystem_handle,"close qmesh light");
    break;
    case 54:
    QCLI_Printf(qcli_ecosystem_handle,"open lock");
    break;
    case 53:
    QCLI_Printf(qcli_ecosystem_handle,"close lock");
    break;
}
```

4.2 Code and Envirment

QCA402X SDK contains sample applications that demonstrate usage of Qualcomm APIs (QAPI) and provides a CLI -based interface to test chip features.Refer to the Programmers Guide on steps to build the sample applications.

The SDK contains build scripts and Makefiles for the following toolchains:

GNU embedded toolchain for ARM- GNU toolchain for ARM-based processors. The toolchain supports Windows and Linux platform and can be downloaded from the ARM website at:https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads.

Add the path to toolchain binaries to 'PATH' environment variable(Supported Version: 6.2).

Linux: export PATH=$PATH:/path/to/bin

Windows: set %PATH%=%PATH%:\path\to\bin

Example:If ARM GNU toolchain is installed under C:\Program Files (x86)\GNU Tools ARM Embedded\4.9 2015q2\bin, set path as follows:

set %PATH%=%PATH%:C:\Program Files (x86)\GNU Tools ARM Embedded\4.9 2015q2\bin.

Python: Some of the support scripts are Python- based, Python 2.7.2 or higher.

4.3 After installation, add the path to python

Example: If you have python.exe in C:\CRMApps\Apps\Python276-64 folder, set path as follows: set %PATH%=%PATH%:C:\CRMApps\Apps\Python276-64

J-Link JTAG:J-Link is an USB based JTAG debugger used by flash and debug scripts are provided in the SDK.

The drivers for J-Link USB interface can be downloaded from:https://www.segger.com/downloads/jlink

4.4 Download and install J-Link Software and Documentation Pack

After installation, add the path to 'JLinkGDBServer'.

Example:

If you have JLinkGDBServer.exe in C:\Program Files (x86)\SEGGER\JLink_V502h folder, then, set path as follows.

set %PATH%=%PATH%:C:\Program Files (x86)\SEGGER\JLink_V502h

4.5 Download AZURE IOT SDK for C

Below steps assume that QCA4020 SDK has been installed at [target] location.

(1)Azure SDK in Quartz has been tested against branch 2017-07-14. Git client is required to clone the repository.

Navigate to [target]\thirdparty directory

(2)Run the following command to clone:

git clone --recursive --branch 2017-07-14 https://github.com/Azure/azure-iot-sdk-c.git azure

(3)Building QCLI Demo with AZURE IOT SDK for C

Following steps should be followed to build the QCLI Demo with Azure support for the first time.

Azure SDK requires some utility functions from arm-none-eabi-gcc libs.

To link with the required libraries, set the following environment variables

SET NEWLIBPATH=C:\Program Files (x86)\GNU Tools ARM Embedded\6.2 2016q4\arm-none-eabi\lib\thumb

SET TOOLLIBPATH=C:\Program Files (x86)\GNU Tools ARM Embedded\6.2 2016q4\lib\gcc\arm-none-eabi\6.2.1\thumb

SET Ecosystem=azure

Navigate to the location of build scripts.

cd target\quartz\demo\QCLI_Demo\build\gcc

Install Device configuration files.

build.bat prepare

4.6 Prepare command will install device configuration files under src/export directory

These device configuration files can now be edited to change system configuration.

This command should only be run once, unless the configuration files are deleted.

Build the sample application. The configuration parameters can be passed as command line parameters.

Parameter	Possible Values	Description
1	f	FreeRTOS RTOS build
	t	Threadx RTOS build
	prepare	Install Device Configuration files
	clobber	Delete Device Configuration files
2	4020	Chipset variant

Following command will build threadx version of the application for QCA4020 module.

build.bat t 4020

Refer to QCA4020 Programmers Guide on how to flash and run the QCLI Demo application.

5. Android Client Develop

5.1 Develop environment

JDK1.8

Android 8.0

Android Studio

5.2 Configure Azure services

```
private static final String connectionString = "";
private static final String deviceId = "";

 public void sendMSGToBoard(String fuction) {
        new Thread() {
            @Override
            public void run() {
                try {
                    ServiceClient serviceClient =
ServiceClient.createFromConnectionString(connectionString, protocol);

                    if (serviceClient != null) {
                        serviceClient.open();
                        FeedbackReceiver feedbackReceiver =
serviceClient.getFeedbackReceiver();
                        if (feedbackReceiver != null)
feedbackReceiver.open();


                        Message messageToSend = new Message(fuction);

 messageToSend.setDeliveryAcknowledgement(DeliveryAcknowledgement.Full
);

                        serviceClient.send(deviceId, messageToSend);
                        System.out.println("Message sent to device");

                        FeedbackBatch feedbackBatch =
feedbackReceiver.receive(10000);
                        if (feedbackBatch != null) {
```

```
                            System.out.println("Message feedback
received, feedback time: " +
feedbackBatch.getEnqueuedTimeUtc().toString());
                        }

                        if (feedbackReceiver != null)
feedbackReceiver.close();
                        serviceClient.close();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }.start();

    }
```