

Alibaba Cloud IoT Platform

Best Practices

Issue: 20190702

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use

or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 Danger: Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 Warning: Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus, page names, and other UI elements.	Click OK.
Courier font	It is used for commands.	Run the <code>cd / d C :/ windows</code> command to enter the Windows system folder.
<i>Italics</i>	It is used for parameters and variables.	<code>bae log list --instanceid Instance_ID</code>
[] or [a b]	It indicates that it is an optional value, and only one item can be selected.	<code>ipconfig [-all -t]</code>

Style	Description	Example
<code>{}</code> or <code>{a b}</code>	It indicates that it is a required value, and only one item can be selected.	<code>switch {stand slave}</code>

Contents

Legal disclaimer.....	I
Generic conventions.....	I
1 Connect to IoT Platform using MQTT.fx.....	1
2 Connect Android Things to Alibaba Cloud IoT Platform.....	10
3 Connect NodeMCU (ESP8266) to IoT Platform.....	17
4 Upload temperature and humidity data to DingTalk chatbots.....	24
5 Set desired property values to control the light bulb status...	31
6 Configure service subscription.....	46
7 Control Raspberry Pi servers from IoT Platform.....	52

1 Connect to IoT Platform using MQTT.fx

This article uses MQTT.fx as an example to describe the method for using a third-party MQTT client to connect to IoT Platform. MQTT.fx is a MQTT client that is written in Java language and based on Eclipse Paho. It supports subscribing to messages and publishing messages through topics.

Prerequisites

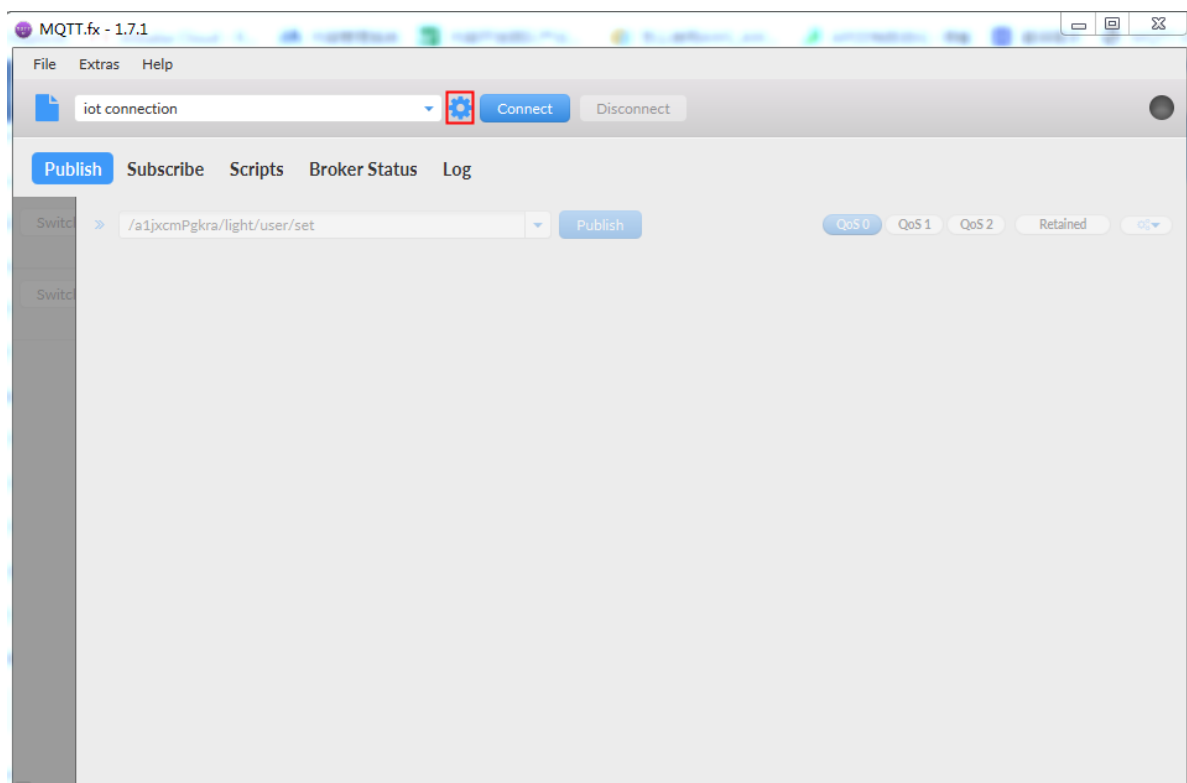
You have created products and devices in the [IoT Platform console](#), and have got the ProductKey, DeviceName, and DeviceSecret of the devices. When you set the connection parameters for MQTT.fx, you will use the values of the ProductKey, DeviceName, and DeviceSecret. See [Create a product](#), [Create a device](#), and [Create multiple devices at a time](#) for help when creating products and devices.

Procedure

1. Download and install the MQTT.fx software.

Download the MQTT.fx software for Windows from [MQTT.fx website](#).

2. Open MQTT.fx, and click the settings icon.



3. Set the connection parameters.

Currently, two types of connection modes are supported: TCP and TLS. These two modes only differ in settings of Client ID and SSL/TLS.

The procedure is as follows:

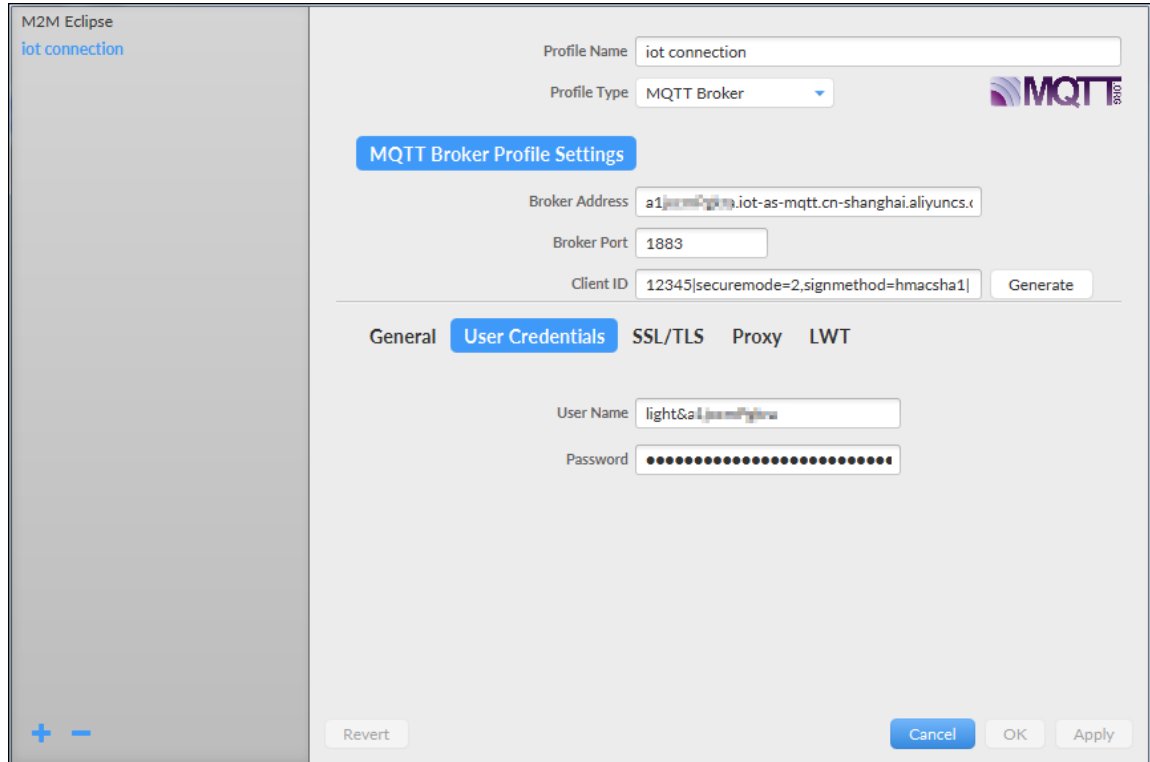
- a. Enter basic information. See the following table for parameter descriptions.

You can keep the default parameters for General, or set the values according to your needs.

Parameter	Description
Profile Name	Enter a custom profile name.
Profile Type	Select MQTT Broker.
Broker Address	Enter the connection domain in the format of <code>\${YourProductKey}.iot-as-mqtt.\${region}.aliyuncs.com</code> . In this format, variable <code>\${region}</code> indicates the region ID of your IoT Platform service region. For region IDs, see Regions and zones . Example: <code>alPUPCoxxxxx.iot-as-mqtt.cn-shanghai.aliyuncs.com</code> .
Broker Port	Set to 1883.

Parameter	Description
Client ID	Enter a value in the format of <code>\${clientId} securemode=3,signmethod=hmactsha1 </code>. Example: <code>12345 securemode=3,signmethod=hmactsha1 </code>. The parameters are described as follows: <code>\${clientId}</code> is a custom client ID. It can be any value within 64 characters. We recommend that you use the MAC address or SN code of the device as the value of <code>clientId</code>. <code>securemode</code> is the security mode of the connection. If you use the TCP mode, set it as <code>securemode=3</code>; if you use the TLS mode, set it as <code>securemode=2</code>. <code>signmethod</code> is the signature method that you want to use. IoT Platform supports <code>hmacmd5</code> and <code>hmactsha1</code>.  Note: Do not click Generate after you enter the Client ID information.

b. Click User Credentials, and enter your User Name and Password.



The screenshot shows the M2M Eclipse IoT connection dialog box. The left sidebar displays 'M2M Eclipse' and 'iot connection'. The main area is titled 'MQTT Broker Profile Settings' and contains the following fields:

- Profile Name: `iot connection`
- Profile Type: `MQTT Broker`
- Broker Address: `a1j...iot-as-mqtt.cn-shanghai.aliyuncs.c`
- Broker Port: `1883`
- Client ID: `12345|securemode=2,signmethod=hmactsha1|` (with a 'Generate' button next to it)

Below these fields are tabs for 'General', 'User Credentials', 'SSL/TLS', 'Proxy', and 'LWT'. The 'User Credentials' tab is selected, showing:

- User Name: `light&al...`
- Password: `.....`

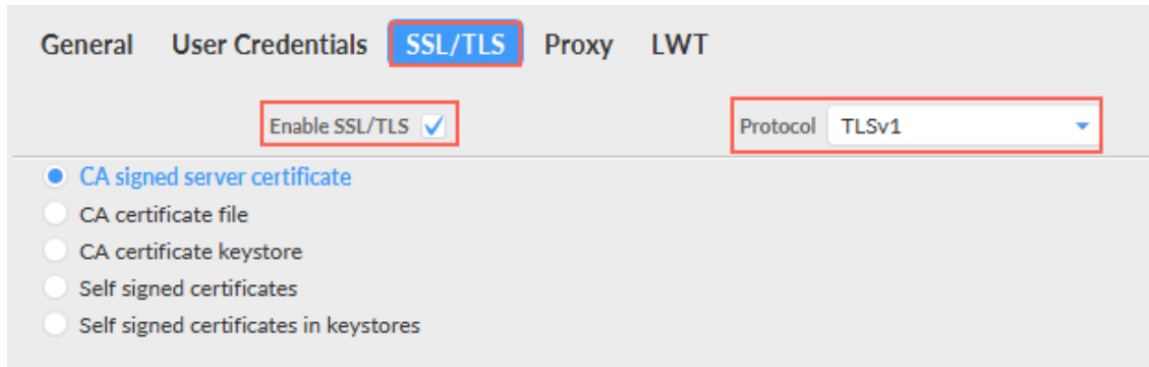
At the bottom, there are buttons for 'Revert', 'Cancel', 'OK', and 'Apply'.

Parameter	Description
User Name	It must be the device name directly followed by the character "&" and the product key. Format: <code>\${YourDeviceName}&\${YourProductKey}</code> . For example, <code>device&f0At5H5TOWF</code> .

Parameter	Description
Password	You must enter an encrypted value of the input parameters. - IoT Platform provides a Password Generator for you to generate one easily. Parameters in the password generator: productKey: The unique identifier of the product to which the device belongs. You can view this information on the device details page in the console. deviceName: The name of the device. You can view this information on the device details page in the console. deviceSecret: The device secret. You can view this information on the device details page in the console. timestamp: (Optional) Timestamp of the current system time. clientId: The custom client ID, which must be the same as the value of <code>clientId</code> in Client ID. method: The signature algorithm, which must be the same as the value of <code>signmethod</code> in Client ID. - You can also encrypt a password by yourself. Generate a password manually: A. Sort and join the parameters. Sort and join the parameters <code>clientId</code> , <code>deviceName</code> , <code>productKey</code> ,and <code>timestamp</code> in a lexicographical order. (If you have not set a timestamp, do not include timestamp in the string.) Joint string example: <code>clientId12 345deviceNamedeviceSecretproductKeyf 0At5H5T0WF</code> B. Encrypt. Use the <code>deviceSecret</code> of the device as the secret key to encrypt the joint string by the signature algorithm defined in Client ID. Suppose the <code>deviceSecret</code> of the device is <code>abc123</code>, the encryption format is <code>hmacsha1 (</code> <code>abc123 , clientId12 345deviceNamedeviceSecretproductKeyf 0At5H5T0WF)</code>.

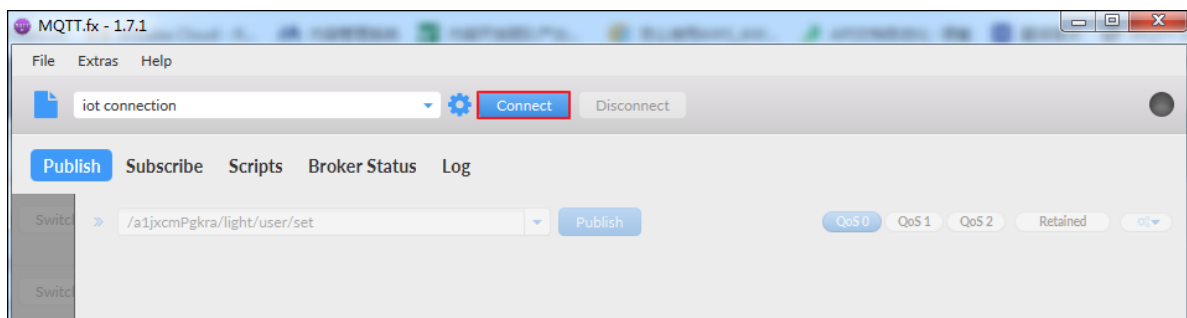
- c. If you use TLS connection mode, you are required to set information for SSL/TLS. SSL/TLS settings are not required when the connection mode is TCP.

Check the box for Enable SSL/TLS, and select TLSv1 as the protocol.



- d. Enter all the required information, and then click OK.

4. Click Connect to connect to IoT Platform.

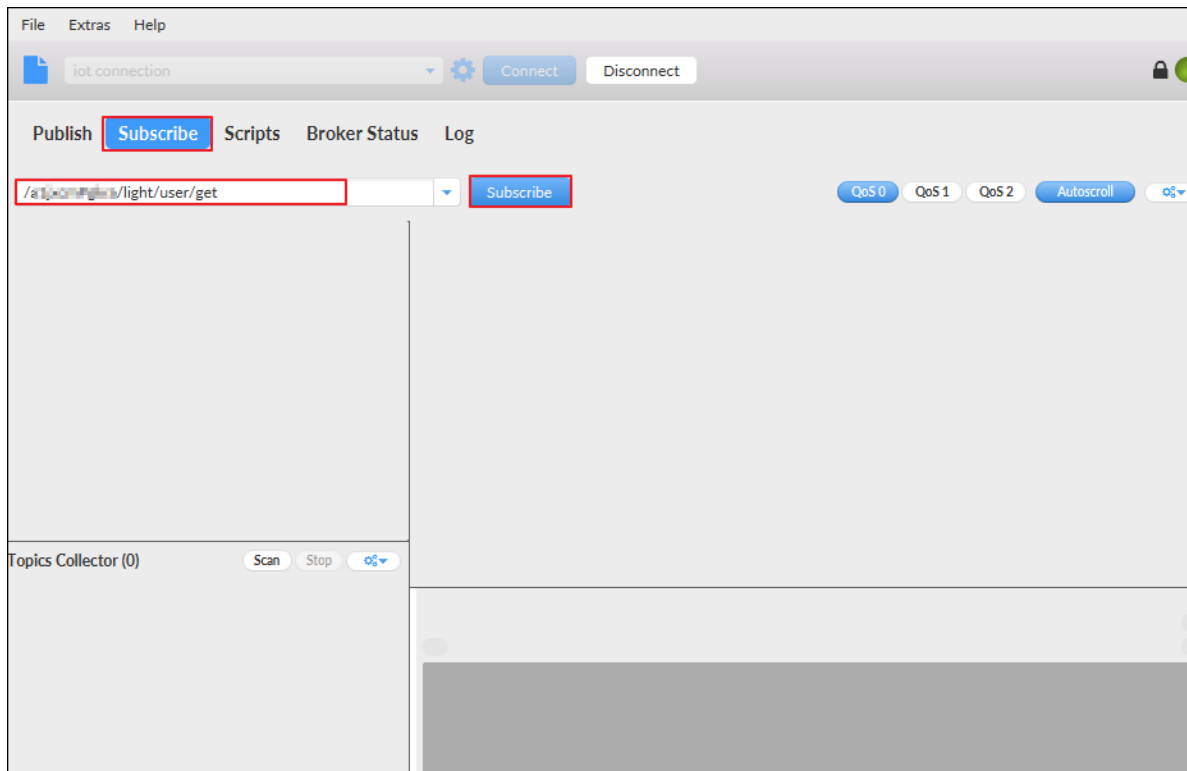


Message communication test

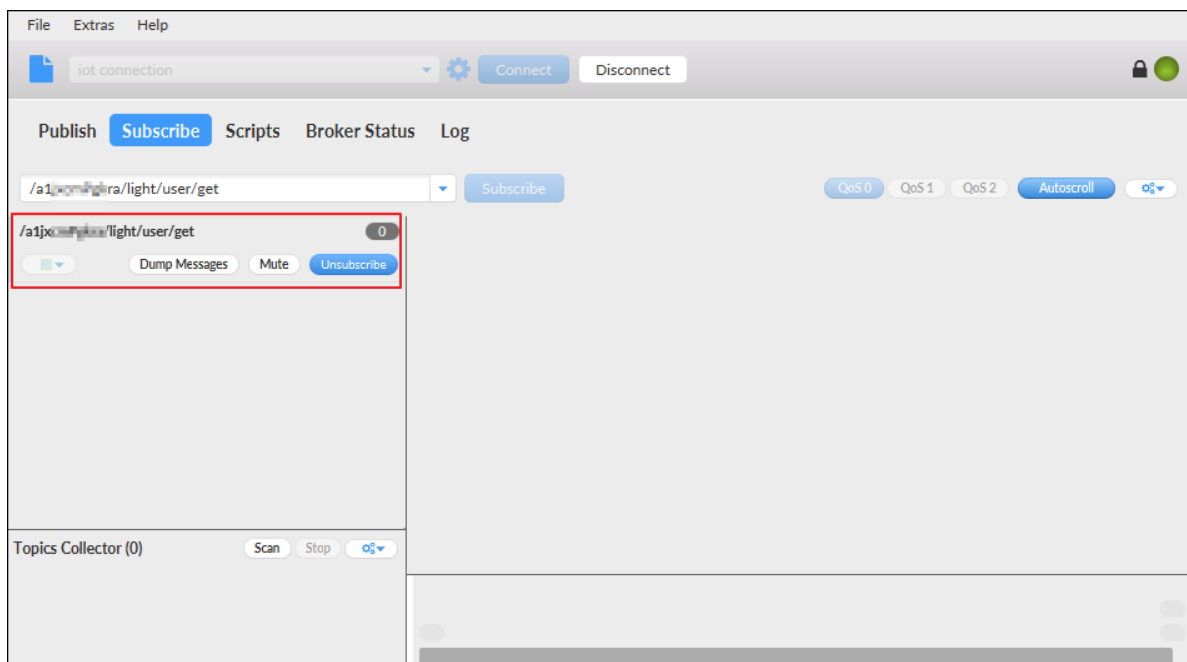
Test whether MQTT.fx and IoT Platform are successfully connected.

1. In MQTT.fx, click Subscribe.

2. Enter a topic of the device, and then click Subscribe.

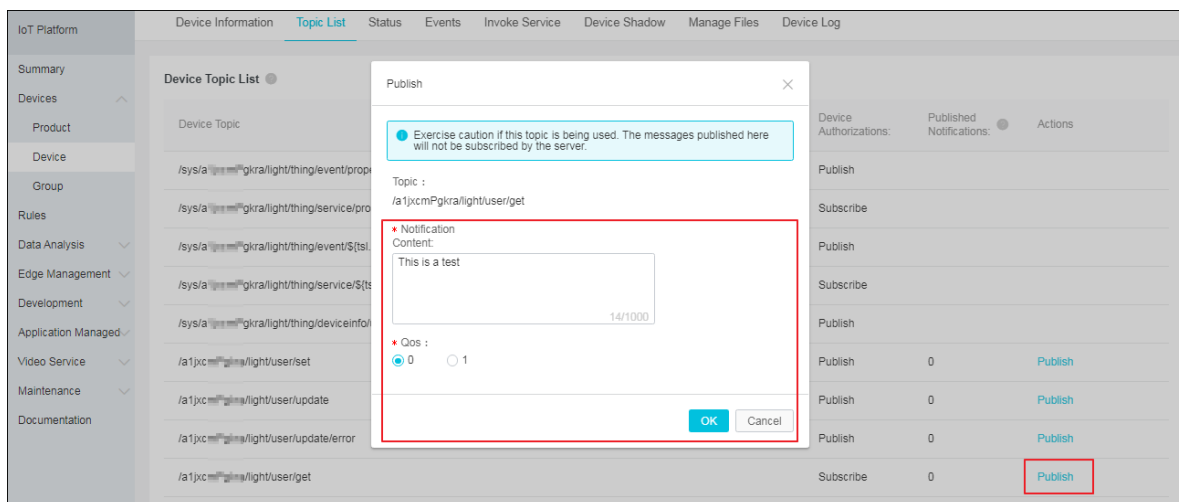


After you have successfully subscribed to a topic, it is displayed in the topic list.

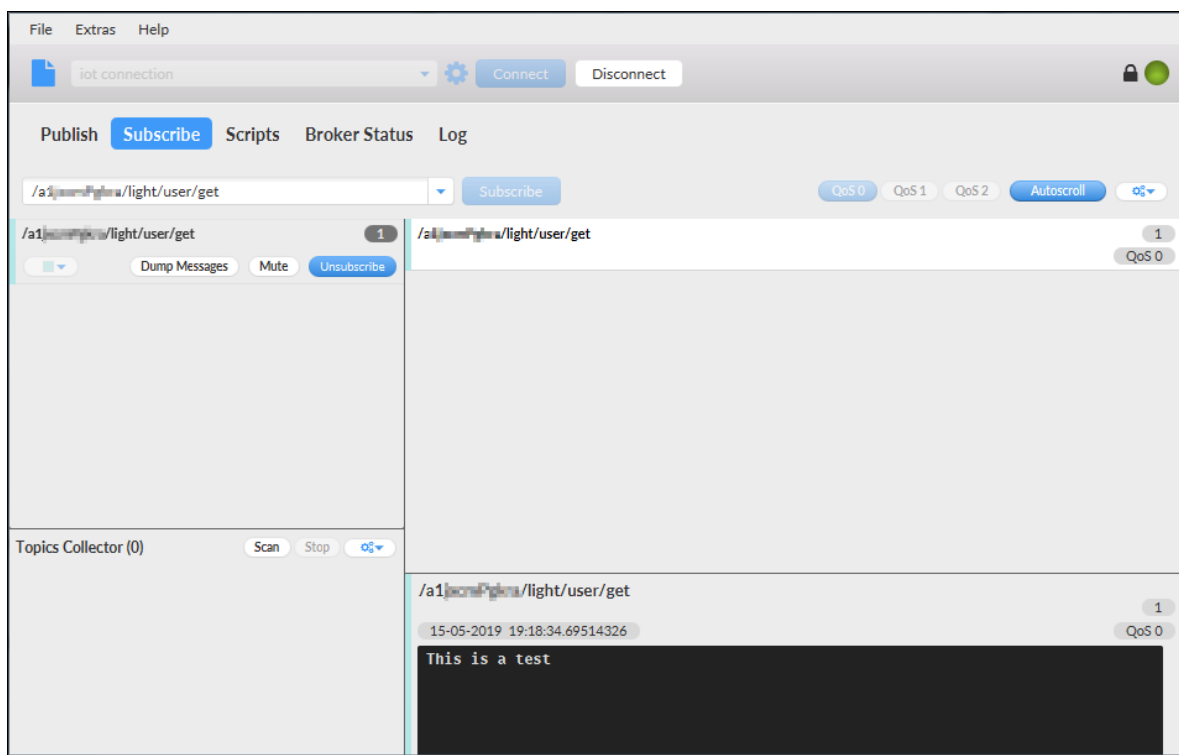


3. In the *IoT Platform console*, in the Topic List of the Device Details page, click the Publish button of the topic that you have subscribed to.

4. Enter message content, and click OK.

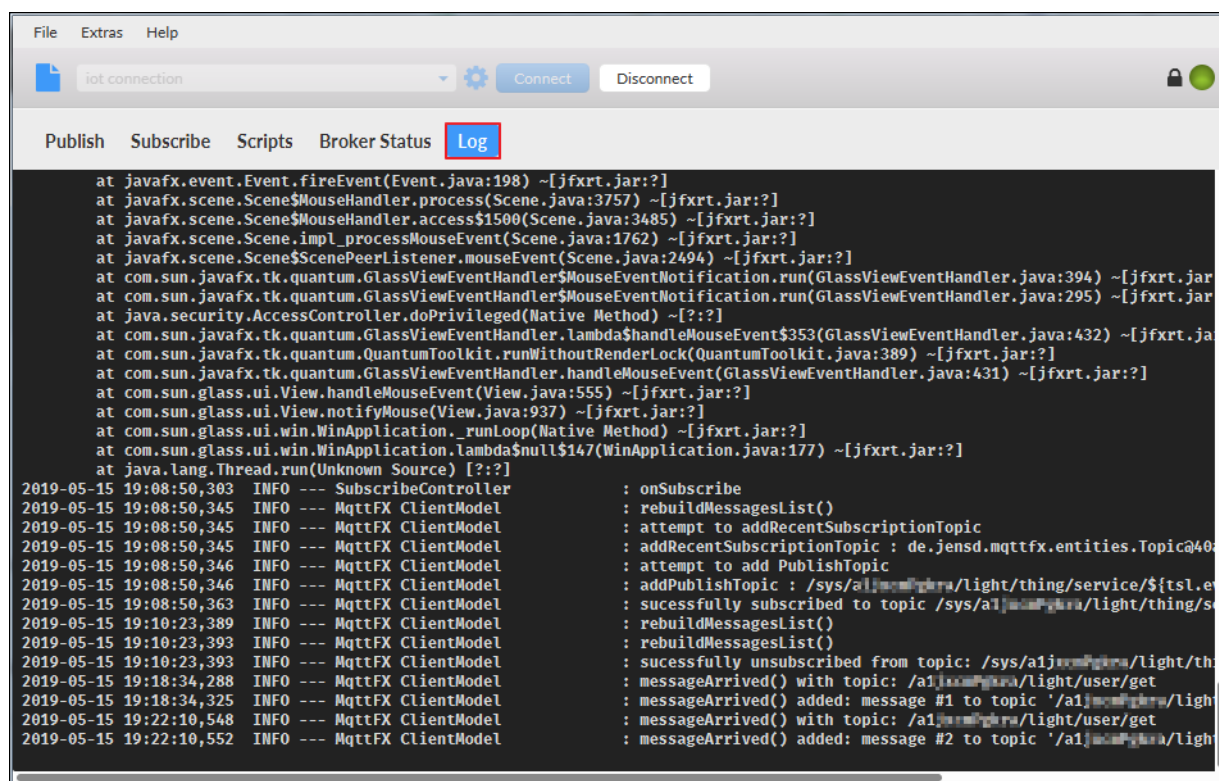


5. Go back to MQTT.fx to check if the message has been received.



View logs

In MQTT.fx, click Log to view the operation logs and error logs.



The screenshot shows the MQTT.fx application window. At the top, there is a menu bar with 'File', 'Extras', and 'Help'. Below the menu bar is a toolbar with a dropdown menu set to 'iot connection', a 'Connect' button, and a 'Disconnect' button. A status bar at the bottom shows a lock icon and a green circle. The main area of the window is a log window with a 'Log' button highlighted in red. The log window displays a series of Java stack traces and MQTT client activity. The stack traces are from the JavaFX runtime, showing the sequence of events from a mouse click to the application's response. The MQTT client activity is shown as a series of log messages from the 'MqttFX ClientModel' class, including subscription and message arrival events.

```
at javafx.event.Event.fireEvent(Event.java:198) ~[jfxrt.jar:?]
at javafx.scene.Scene$MouseHandler.process(Scene.java:3757) ~[jfxrt.jar:?]
at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485) ~[jfxrt.jar:?]
at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762) ~[jfxrt.jar:?]
at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:394) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:295) ~[jfxrt.jar:?]
at java.security.AccessController.doPrivileged(Native Method) ~[?:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$353(GlassViewEventHandler.java:432) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:431) ~[jfxrt.jar:?]
at com.sun.glass.ui.View.handleMouseEvent(View.java:555) ~[jfxrt.jar:?]
at com.sun.glass.ui.View.notifyMouse(View.java:937) ~[jfxrt.jar:?]
at com.sun.glass.ui.win.WinApplication._runLoop(Native Method) ~[jfxrt.jar:?]
at com.sun.glass.ui.win.WinApplication.lambda$null$147(WinApplication.java:177) ~[jfxrt.jar:?]
at java.lang.Thread.run(Unknown Source) [?:?]
2019-05-15 19:08:50,303 INFO --- SubscribeController : onSubscribe
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : attempt to addRecentSubscriptionTopic
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : addRecentSubscriptionTopic : de.jensd.mqttfx.entities.Topic@40a
2019-05-15 19:08:50,346 INFO --- MqttFX ClientModel : attempt to add PublishTopic
2019-05-15 19:08:50,346 INFO --- MqttFX ClientModel : addPublishTopic : /sys/a1jwcnfpgw/light/thing/service/${tsl.e
2019-05-15 19:08:50,363 INFO --- MqttFX ClientModel : successfully subscribed to topic /sys/a1jwcnfpgw/light/thing/s
2019-05-15 19:10:23,389 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:10:23,393 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:10:23,393 INFO --- MqttFX ClientModel : successfully unsubscribed from topic: /sys/a1jwcnfpgw/light/th
2019-05-15 19:18:34,288 INFO --- MqttFX ClientModel : messageArrived() with topic: /a1jwcnfpgw/light/user/get
2019-05-15 19:18:34,325 INFO --- MqttFX ClientModel : messageArrived() added: message #1 to topic '/a1jwcnfpgw/ligh
2019-05-15 19:22:10,548 INFO --- MqttFX ClientModel : messageArrived() with topic: /a1jwcnfpgw/light/user/get
2019-05-15 19:22:10,552 INFO --- MqttFX ClientModel : messageArrived() added: message #2 to topic '/a1jwcnfpgw/ligh
```

2 Connect Android Things to Alibaba Cloud IoT Platform

This article uses an indoor air test project as an example to explain how to connect Google Android Things to Alibaba Cloud IoT Platform.

Hardware

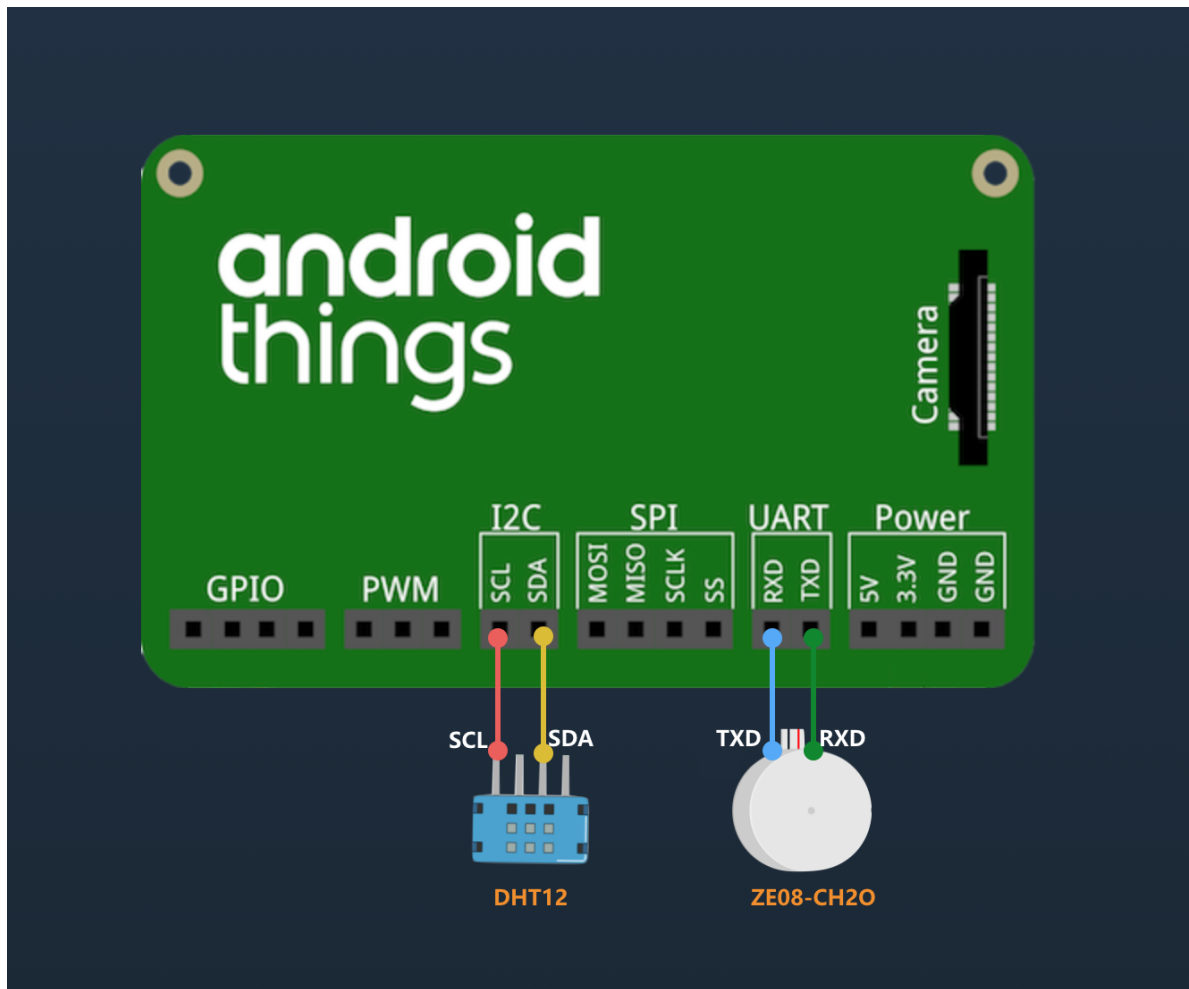
- Hardware list for the project

The following table lists the hardware required by the indoor air test project.

Hardware	Picture	Remarks
NXP Pico i.MX7D Development board		Android Things 1.0  Note: You can also use Raspberry Pi instead.
DHT12 Temperature and humidity sensor		Supports I2C data communication method.

Hardware	Picture	Remarks
ZE08-CH2O Formaldehyde detection sensor	 A photograph of a ZE08-CH2O Formaldehyde detection sensor module. The module consists of a green printed circuit board (PCB) with a square silver-colored metal housing in the center. Inside the housing is a circular white sensor element. At the bottom of the PCB, there is a white 5-pin header connector with pins numbered 1 to 5. The PCB has four mounting holes at the corners.	Supports UART data communicat ion method.

- Diagram of the hardware connection



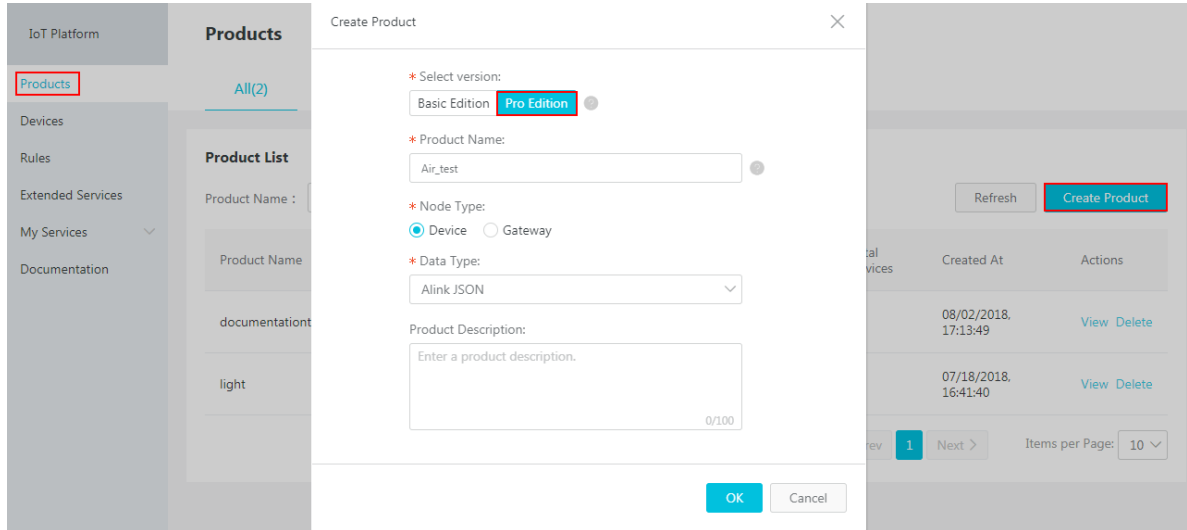
- Connect the SCL (clock line) and SDA (data line) pins of the temperature and humidity sensor (DHT12) with the I2C SCL and SDA pins of the development board.
- Connect the TXD (transmit data) pin of the formaldehyde detection sensor (ZE08-CH20) with the RXD (receive data) pin of the development board, and connect the RXD pin of ZE08-CH20 with the TXD pin of the development board.

Create a product and device in the Alibaba Cloud IoT Platform console

1. Log on to the [IoT Platform console](#).

2. Create a product in IoT Platform.

On the Products page, click Create Product. For more information, see [Create a product](#).



3. Define features for the newly created product.

On the Product Details page, click Define Feature > Add, and then define properties for the product. For more information, see [Overview](#).

Table 2-1: Properties required for the indoor air test project

Property name	Identifier	Data type	Value range	Description
Temperature	temperature	float	-50~100	Detected by DHT12.
Humidity	humidity	float	0~100	Detected by DHT12.

Property name	Identifier	Data type	Value range	Description
Formaldehyde concentration	ch2o	double	0~3	Detected by ZE08.

IoT Platform

Products > Product Details

Air_test [Pro Edition](#)

ProductKey : [Copy](#) ProductSecret : [Show](#) Total Devices:0 [Manage](#)

Product Information Notifications **Define Feature** Device Log Online Debugging

Define Feature A standard feature is automatically created based on the device type of the product. You can also add optional features or create your own custom features. [View TSL](#) [Add](#)

Feature Type	Feature Name:	Identifier:	Data Type	Data Definition	Actions
Properties	Temperature	temperature	float	Value Range:-50 - 100	Edit Delete
Properties	Humidity	humidity	float	Value Range:0 - 100	Edit Delete
Properties	Formaldehyde concentration	ch2o	double	Value Range:0 - 3	Edit Delete

4. Create a device

On the Devices page, select the name of the newly created product, click Add Device, and then create a device. For more information, see [Create a device](#).

IoT Platform

Products

Devices

Device List

Device Name:

Product :

DeviceName :

[Add Device](#)

OK Cancel

Develop the Android Things client

1. Use Android Studio to create an Android Things project, and add the permission for Internet.

```
< uses-permission android:name="android.permission.INTERNET" />
```

2. Add eclipse.paho.mqtt to Gradle file.

```
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.0'
```

3. Set that data of DHT12 is read through I2C.

```
private void readDataFromI2C () {
    try {
        byte[] data = new byte[5];
        i2cDevice.readRegBuffer(0x00, data, data.length);

        // check data
        if ((data[0] + data[1] + data[2] + data[3]) % 256 != data[4]) {
            humidity = temperature = 0;
            return;
        }
        // humidity data
        humidity = Double.valueOf(String.valueOf(data[0]) + "." + String.valueOf(data[1]));
        Log.d(TAG, "humidity: " + humidity);
        // temperature data
        if (data[3] < 128) {
            temperature = Double.valueOf(String.valueOf(data[2]) + "." + String.valueOf(data[3]));
        } else {
            temperature = Double.valueOf("-" + String.valueOf(data[2]) + "." + String.valueOf(data[3] - 128));
        }

        Log.d(TAG, "temperature: " + temperature);
    } catch (IOException e) {
        Log.e(TAG, "readDataFromI2C error " + e.getMessage(), e);
    }
}
```

4. Set that data of Ze08-CH2O is read through UART.

```
try {
    // data buffer
    byte[] buffer = new byte[9];

    while (uartDevice.read(buffer, buffer.length) > 0) {
```

```

        if ( checksum ( buffer )) {
            ppbCh2o = buffer [ 4 ] * 256 + buffer
[ 5 ];
            ch2o = ppbCh2o / 66 . 64 * 0 . 08 ;
        } else {
            ch2o = ppbCh2o = 0 ;
        }
        Log . d ( TAG , " ch2o : " + ch2o );
    }

    } catch ( IOException e ) {
        Log . e ( TAG , " Ze08CH2O read data error "
+ e . getMessage (), e );
    }
}

```

5. Connect Alibaba Cloud IoT Platform and the client, and report data.

```

/*
Payload format
{
    " id ": 123243 ,
    " params ": {
        " temperatur e ": 25 . 6 ,
        " humidity ": 60 . 3 ,
        " ch2o ": 0 . 048
    },
    " method ": " thing . event . property . post "
}
*/
MqttMessag e message = new MqttMessag e ( payload .
getBytes ( " utf - 8 " ));
message . setQos ( 1 );

String pubTopicYour Pc = "/ sys / ${ YourProduc tKey } / ${
YourDevice Name } / thing / event / property / post ";

mqttClient . publish ( pubTopic , message );

```

View real-time data of the device

After the device is enabled, you can view the real-time data of the device from the Status column on the Device Details page in IoT Platform console.

The screenshot shows the 'Status' tab of the 'Temperature-sensor' device in the IoT Platform console. The page displays the following data:

Formaldehyde concentration	Temperature	Humidity
0.03mg/m³	10C°	27%
Last update: 2018/07/19 15:50:16	Last update: 2018/07/19 15:50:16	Last update: 2018/07/19 15:50:16
View logs	View logs	View logs

Additional interface elements include a 'Real-time Refresh' toggle, 'Chart' and 'Form' buttons, and a sidebar with navigation options like 'Products', 'Rules', and 'My Services'.

3 Connect NodeMCU (ESP8266) to IoT Platform

You can use NodeMCU to develop IoT applications and connect your NodeMCU to Alibaba Cloud IoT Platform. In this way, you can control your NodeMCU application server from IoT Platform. This topic describes how to connect a NodeMCU application server to IoT Platform by using C programming code in an example.

Background information

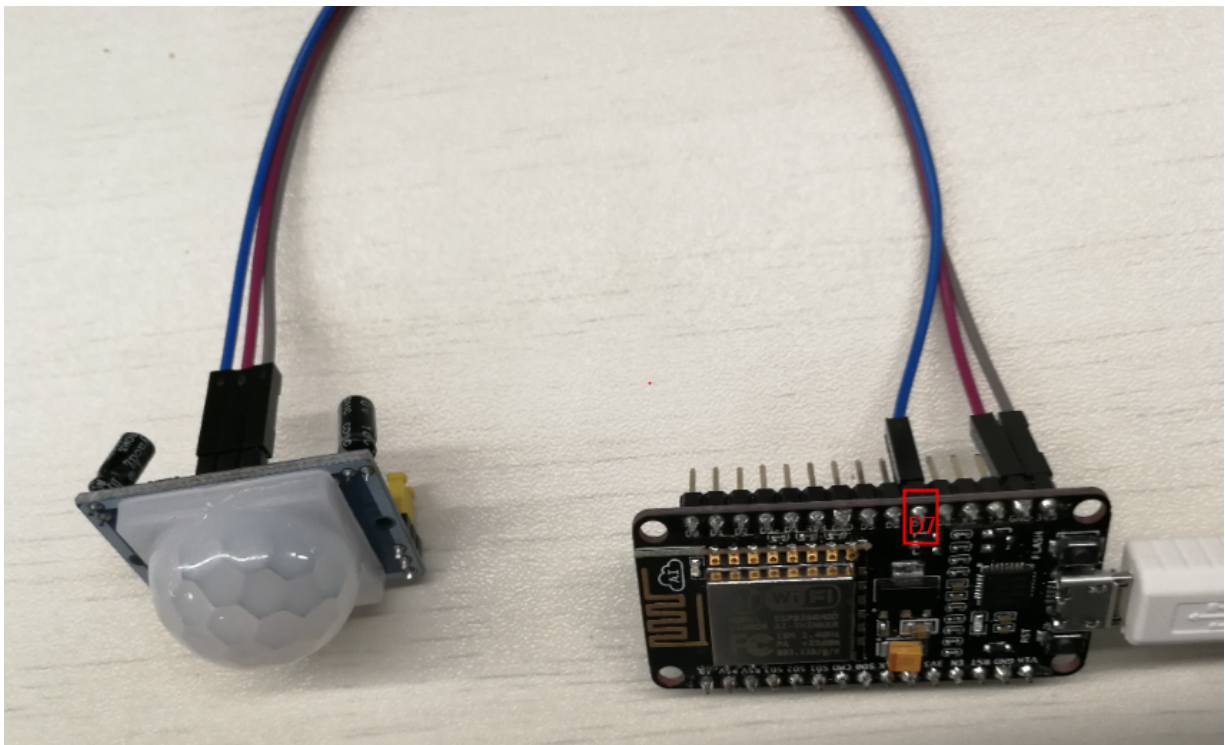
NodeMCU is a development board based on the ESP8266 chip. The ESP8266 chip integrates the Wi-Fi function. Therefore, the NodeMCU application server can be used to control its connected devices in the same LAN. However, if you cannot directly operate the NodeMCU application server, you cannot control the devices. To resolve this issue, connect your NodeMCU application server to Alibaba Cloud IoT Platform so that you can control the NodeMCU application server from IoT Platform.

The following section illustrates how to use NodeMCU to control a light bulb device.



Note:

Connect the data cable of the sensor to the D7 pin, which corresponds to GPIO 13, on the NodeMCU development board.



Prerequisites

Before you begin, you must see the official NodeMCU IDE Tutorial for development software preparation . This example uses Arduino IDE as development software.

Connect NodeMCU to IoT Platform

After the NodeMCU development environment is ready, you can connect NodeMCU to IoT Platform.

1. Log on to the [IoT Platform console](#).
2. Create a product, as shown in [Create a product](#).
3. Define features for the product, as shown in [Define features](#).

This example uses the following properties:

The screenshot shows the 'Define Feature' page in the IoT Platform console. The product is 'Hygrothermograph' (Pro Edition). The 'Define Feature' tab is selected. The page displays two sections: 'Standard Feature' and 'Self-Defined Feature'. The 'Standard Feature' section shows 'No standard feature'. The 'Self-Defined Feature' section contains a table with one feature defined.

Feature Type	Feature Name	Identifier	Data Type	Data Definition	Actions
Prope...	Idle	idle	bool	Boolean Value: on - 0 ; off - 1 ;	Edit Delete

4. Create a device, and obtain the device certificate information, including ProductKey, DeviceName, and DeviceSecret. For more information, see [Create a device](#).
5. Write code.
 - a. Connect your computer to the NodeMCU development board by using a USB cable.
 - b. Open the Arduino IDE and write esp8266.ino code.
 - c. Upload code.

The sample code is as follows:



Note:

You can generate the value of MQTT_PASSWORD by using the mqttPassword encryption method described in [Establish MQTT connections over TCP](#).

```
# include < ESP8266WiFi . h >
/* The PubSubClient 2 . 4 . 0 dependency */
# include < PubSubClient . h >
/* The ArduinoJson 5 . 13 . 4 dependency */
# include < ArduinoJson . h >

# define SENSOR_PIN 13

/* The SSID and password for Wi - Fi connection */
# define WIFI_SSID " SSID "
# define WIFI_PASSWORD " Password "

/* Device certificate information */
# define PRODUCT_KEY " The ProductKey of the device "
# define DEVICE_NAME " The DeviceName of the device "
# define DEVICE_SECRET " The DeviceSecret of the device "
# define REGION_ID " The region ID , such as cn - shanghai "

/* The domain name and port number that do not require any modifications . */
# define MQTT_SERVER PRODUCT_KEY ". iot - as - mqtt ." REGION_ID ". aliyuncs . com "
# define MQTT_PORT 1883
# define MQTT_USERNAME DEVICE_NAME "&" PRODUCT_KEY

# define CLIENT_ID " esp8266 | securemode = 3 , timestamp = 1234567890 , signmethod = hmacsha1 | "
// See " Establish MQTT connections over TCP " ( https :// www . alibabacloud . com / help / doc - detail / 73742 . html ) to generate a password .
// The plain text for encryption is a combination of parameter - value pairs in lexicographical order . For example , clientId esp8266deviceName ${ deviceName } productKey ${ productKey } timestamp 1234567890 .
// The encryption key is the DeviceSecret .
# define MQTT_PASSWORD " f1ef80ee8a dd322c519a 5a6bec326d c499f854b8 "

# define ALINK_BODY_FORMAT "{ \" id \": \" 123 \", \" version \": \" 1 . 0 \", \" method \": \" thing . event . property . post \", \" params \": { s } } "
# define ALINK_TOPIC_C_PROP_POST "/ sys /" PRODUCT_KEY "/ " DEVICE_NAME "/ thing / event / property / post "

unsigned long lastMs = 0 ;
WiFiClient espClient ;
PubSubClient client ( espClient );

void callback ( char * topic , byte * payload , unsigned int length )
{
    Serial . print ( " Message arrived [ " );
    Serial . print ( topic );
```

```

    Serial . print ("] ");
    payload [ length ] = '\ 0 ';
    Serial . println (( char *) payload );
}

void  wifiInit ()
{
    WiFi . mode ( WIFI_STA );
    WiFi . begin ( WIFI_SSID ,  WIFI_PASSW D );
    while ( WiFi . status () != WL_CONNECT ED )
    {
        delay ( 1000 );
        Serial . println (" WiFi  not  Connect ");
    }

    Serial . println (" Connected  to  AP ");
    Serial . println (" IP  address : ");
    Serial . println ( WiFi . localIP ());

    Serial . print (" espClient  [");

        client . setServer ( MQTT_SERVE R , MQTT_PORT );  /*
Connect to the MQTT server after connecting to the
Wi - Fi */
        client . setCallbac k ( callback );
    }

    void  mqttCheckC onnect ()
    {
        while (! client . connected ())
        {
            Serial . println (" Connecting to MQTT Server ...");
            if ( client . connect ( CLIENT_ID , MQTT_USRNA ME ,
MQTT_PASSW D ))
            {
                Serial . println (" MQTT  Connected !") ;
            }
            else
            {
                Serial . print (" MQTT  Connect  err :");
                Serial . println ( client . state ());
                delay ( 5000 );
            }
        }
    }

    void  mqttInterv alPost ()
    {
        char  param [ 32 ];
        char  jsonBuf [ 128 ];

        sprintf ( param , "{\  idle \":% d }", digitalRea d ( 13 ));
        sprintf ( jsonBuf , ALINK_BODY _FORMAT , param );
        Serial . println ( jsonBuf );
    }
}

```

```
        boolean d = client . publish ( ALINK_TOPI  C_PROP_POS  T ,
        jsonBuf );
        Serial . print ( " publish : 0   Failure ; 1   Success " );
        Serial . println ( d );
    }

    void  setup ()
    {

        pinMode ( SENSOR_PIN ,   INPUT );
        /* initialize serial for debugging */
        Serial . begin ( 115200 );
        Serial . println ( " Demo   Start " );

        wifiInit ();
    }

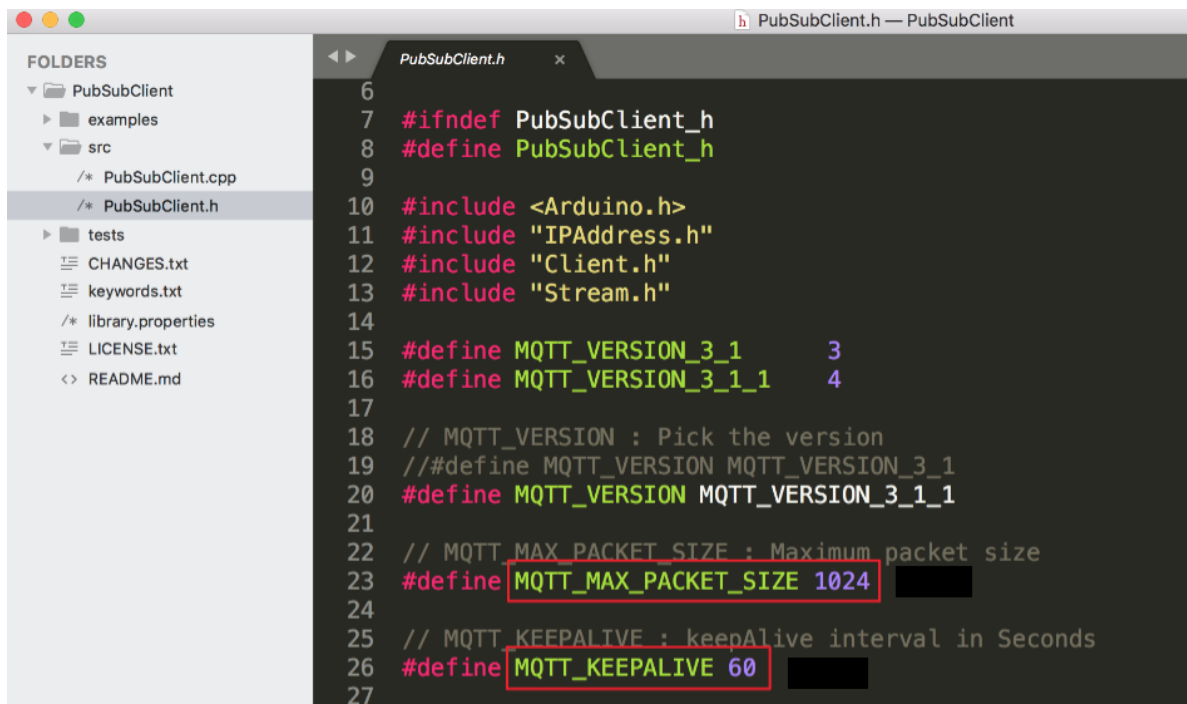
    // the loop function runs over and over again
    forever
    void  loop ()
    {
        if ( millis () - lastMs  >=  5000 )
        {
            lastMs = millis ();
            mqttCheckC onnect ();

            /* The heartbeat interval */
            mqttInterv alPost ();
        }

        client . loop ();
        if ( digitalRea d ( SENSOR_PIN ) ==  HIGH ){
        Serial . println ( " Motion   detected !" ) ;
        delay ( 2000 );
        }
        else {
        Serial . println ( " Motion   absent !" ) ;
        delay ( 2000 );
        }
    }
}
```

```
}
```

6. Modify the values of `MQTT_MAX_PACKET_SIZE` and `MQTT_KEEPA_LIVE` on the MQTT client as needed.
 - a. Locate `PubSubClient` in the library manager of Arduino IDE.
 - b. Open the `PubSubClient.h` file under `src`.
 - c. Modify the `MQTT_MAX_PACKET_SIZE` and `MQTT_KEEPA_LIVE` parameters. To set the `MQTT_MAX_PACKET_SIZE` and `MQTT_KEEPA_LIVE` parameters, see [Limits](#).



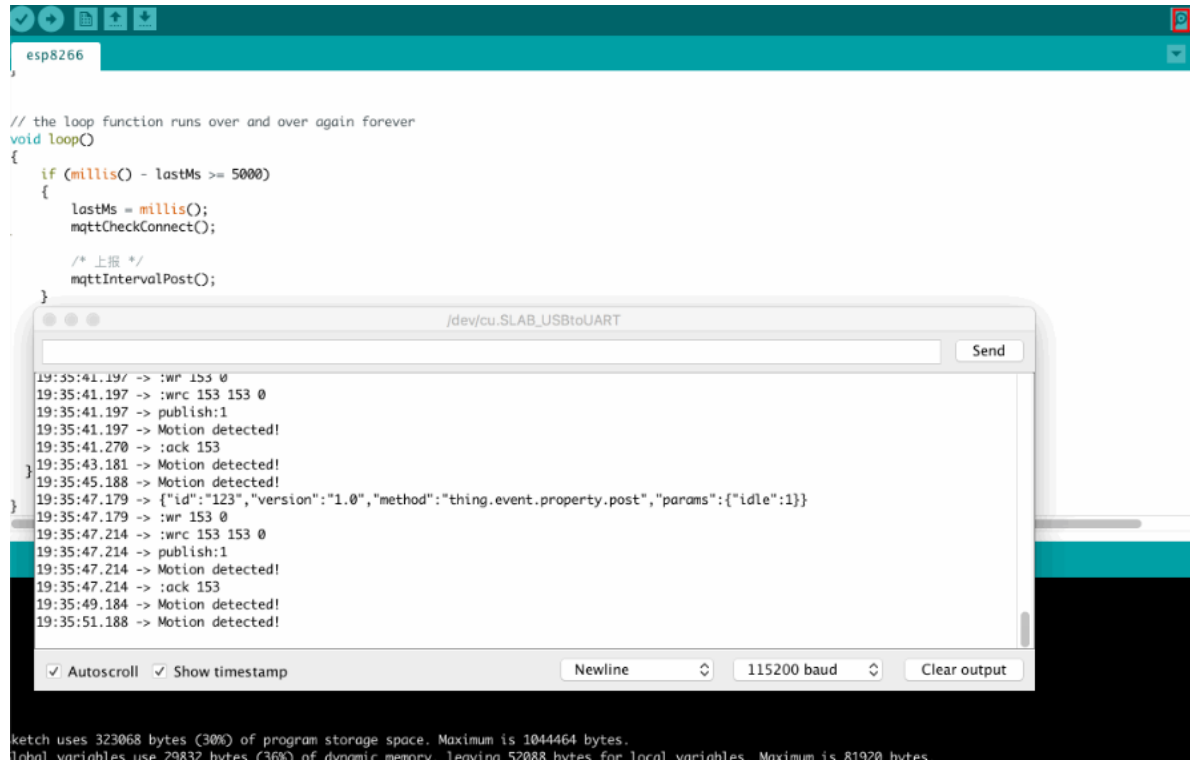
Perform a test

After the development is complete, you can use the device to report properties to test whether the NodeMCU application server and IOT Platform can receive reported properties from the device.

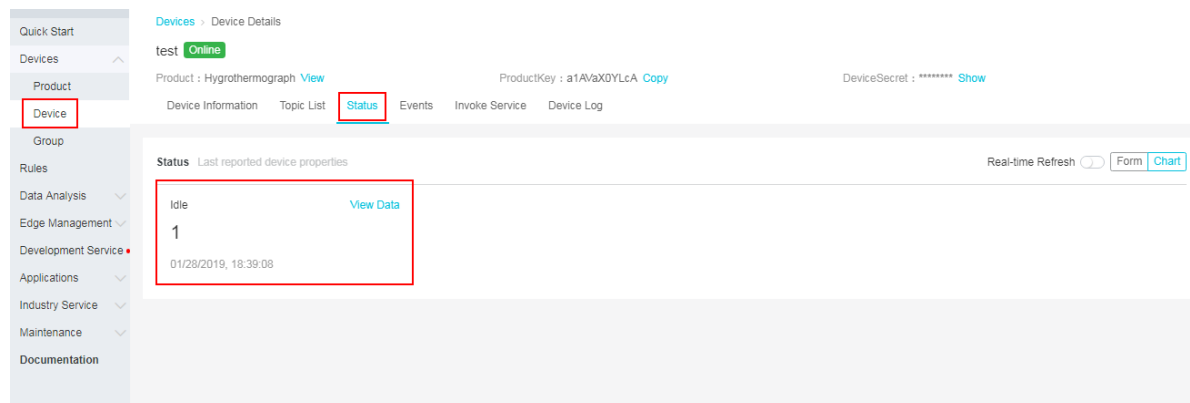
After the device reports properties, you can view the running status of the device on the development software and in the IOT Platform console.

- On the development software, click the serial port listener icon in the upper-right corner of the code editing page.

You can view the running status of the program, as shown in the following figure:



- In the IoT Platform console, go to the Device Details page of the corresponding device. On the Status tab page, view the reported properties and property history, as shown in the following figure:



4 Upload temperature and humidity data to DingTalk chatbots

Context

- Scenario:

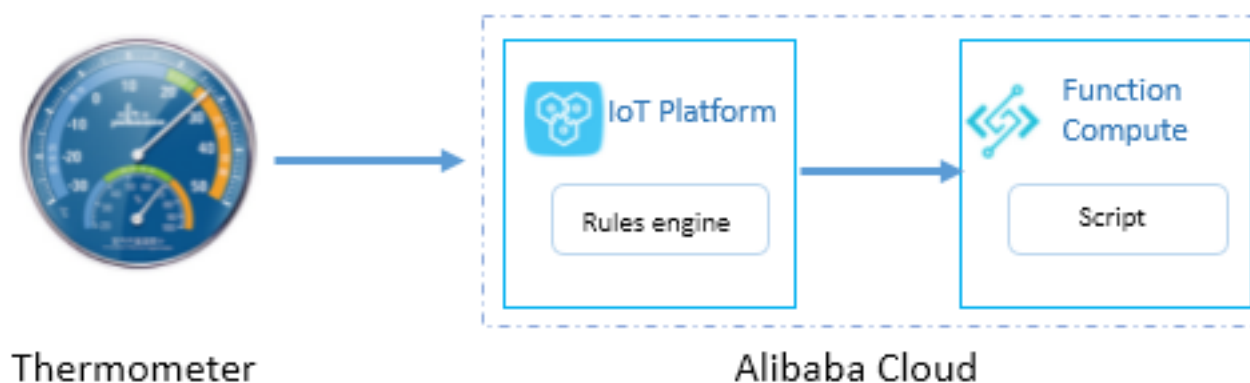
Upload the data that is collected by the temperature and humidity sensor to DingTalk chatbots.

- Business logic:

Connect the temperature and humidity sensor to IoT Platform through MQTT . Configure the rules engine to send the temperature and humidity data to the pushData2DingTalk function in Function Compute. The function processes the data and posts the results to the Webhook address of the specified DingTalk chatbot. The DingTalk group can then receive the temperature and humidity data sent by this DingTalk chatbot.

The data flow diagram is shown in [Figure 4-1: Data flow diagram](#).

Figure 4-1: Data flow diagram



Procedure

1. Configure the DingTalk chatbot.

- a) Log on to the desktop version of DingTalk.
- b) In the upper-right corner of the DingTalk group page, click **...**, and then select ChatBot.
- c) Click Add Robot, select Custom, then click Add.
- d) Enter a name for the robot, click Next, and then click Finish.

2. Create a function.

- a) Activate Alibaba Cloud Function Compute service first.

Function Compute is an event-driven, fully managed computing service. Currently supported languages include Java, Node.js and Python. For more information, see [How to use Function Compute](#).

- b) Write the function script code. In this example, Node.js is used. The function obtains the device location, device number, temperature, humidity, and the time of recording from IoT Platform, and splices the data according to the specified

DingTalk message format. Data will be sent to the Webhook address of the specified DingTalk chatbot using HTTPS Post methods.

- c) Log on to the Function Compute console and create a service named IoT_Service.
- d) Click Create Function, and select the Empty Function template.
- e) Select No Trigger and specify the basic configurations as shown in the following picture.

Figure 4-2: Basic configurations

The screenshot shows the 'Function Information' and 'Code Configuration' sections of the Function Compute console. In the 'Function Information' section, the 'Service Name' is 'IoT_Service' and the 'Function Name' is 'pushData2DingTalk'. The 'Runtime' is set to 'nodejs6'. In the 'Code Configuration' section, the 'In-line Edit' option is selected, showing the following code:

```
1 const https = require('https');
2 const accessToken = 'Specify the webhook accessToken of the DingTalk robot';
3 module.exports.handler = function(event, context, callback) {
4   var eventJson = JSON.parse(event.toString());
5   // DingTalk message format
6   const postData = JSON.stringify({
7     "msgtype": "markdown",
8     "markdown": {
9       "title": "Temperature and humidity sensor",
10      "text": "#### Temperature and humidity details\n" +
11        "> Device location: " + eventJson.tag + "\n\n" +
12        "> Device number: " + eventJson.isn + "\n\n" +
13        "> Temperature: " + eventJson.temperature + "°C\n\n" +
14        "> Humidity: " + eventJson.humidity + "%\n\n" +
15        "##### " + eventJson.time + " published by [ Alibaba Cloud IoT Platform ]( https:// www . aliyun . com / product / iot ) \n "
16      },
17      "at": {
18        "isAtAll": false
19      }
20    });
21   https.post(accessToken, postData, callback);
22 }
```

The function pushData2DingTalk is declared as follows:

```
const https = require('https');
const accessToken = 'Specify the webhook accessToken of the DingTalk robot';
module.exports.handler = function(event, context, callback) {
  var eventJson = JSON.parse(event.toString());
  // DingTalk message format
  const postData = JSON.stringify({
    "msgtype": "markdown",
    "markdown": {
      "title": "Temperature and humidity sensor",
      "text": "#### Temperature and humidity details\n" +
        "> Device location: " + eventJson.tag + "\n\n" +
        "> Device number: " + eventJson.isn + "\n\n" +
        "> Temperature: " + eventJson.temperature + "°C\n\n" +
        "> Humidity: " + eventJson.humidity + "%\n\n" +
        "##### " + eventJson.time + " published by [ Alibaba Cloud IoT Platform ]( https:// www . aliyun . com / product / iot ) \n "
    },
    "at": {
      "isAtAll": false
    }
  });
  https.post(accessToken, postData, callback);
}
```

```
}
});
const options = {
  hostname: 'oapi.dingtalk.com',
  port: 443,
  path: '/robot/send?access_token=' + accessToken,
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': Buffer.byteLength(postData)
  }
};
const req = https.request(options, (res) => {
  res.setEncoding('utf8');
  res.on('data', (chunk) => {});
  res.on('end', () => {
    callback(null, 'success');
  });
});
// When an exception returns
req.on('error', (e) => {
  callback(e);
});
// Write the data
req.write(postData);
req.end();
};
```

3. Configure IoT Platform.

- a) In the IoT Platform console, select Products and then create a product for the temperature and humidity sensor.
- b) On the Topic Categories tab page of the product details page, create a topic category / productKey / \${ deviceName } / user / data whose Device Operation Authorizations is Publish.
- c) On the Define Feature tab page, define two properties: temperature and humidity.
- d) Select Devices and then register a device.
- e) Click View next to the device name. On the Device Tag tab, click Add to add two device tags.

Tag Key	Tag Value	Description
tag	Room 007S, 3rd Floor, Building 2, Cloud Town	Device location

Tag Key	Tag Value	Description
deviceISN	T20180102X nbKjmoAnUb	Device number

- f) Select Rules to create and enable a rule. A complete rule contains three parts: basic information, data processing SQL, and data forwarding. You can specify multiple forwarding actions for a rule.

A. Configure the data processing script.

The rules engine supports [SQL statements](#).

The device name (deviceName) and attributes (tag and deviceISN) are required.

This SQL is to obtain the temperature and humidity values from the payload of the messages that are sent from the temperature and humidity sensor.

Write SQL

Rule Query Expression:

Copy Statement

```
SELECT
deviceName() as deviceName, attribute("tag") as tag, attribute('deviceISN') as isn, temperature, humidity, timestamp('yyyy-MM-dd HH:mm:ss') as time
FROM  "/a1ROnnrL7IA/+user/data"
WHERE
```

Field:

deviceName() as deviceName, attribute("tag") as tag, attribute('deviceISN') as isn, temperature, humidity, timestamp('yyyy-MM-dd HH:mm:ss') as time

Topic :

/a1ROnnrL7IA/+user/data

Custom

MQdevice

All equipment (+)

user/data

Conditions (Optional)

You can use Rules Engine functions, such as: deviceName()

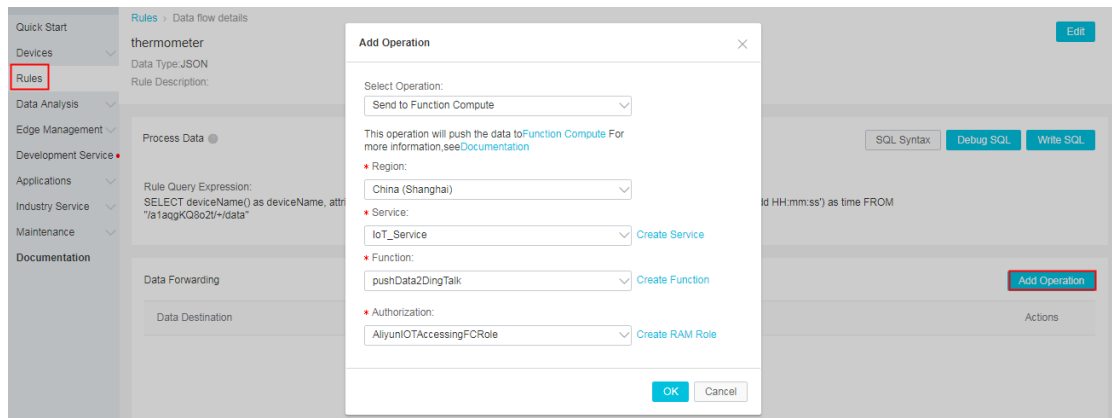
OK

Cancel

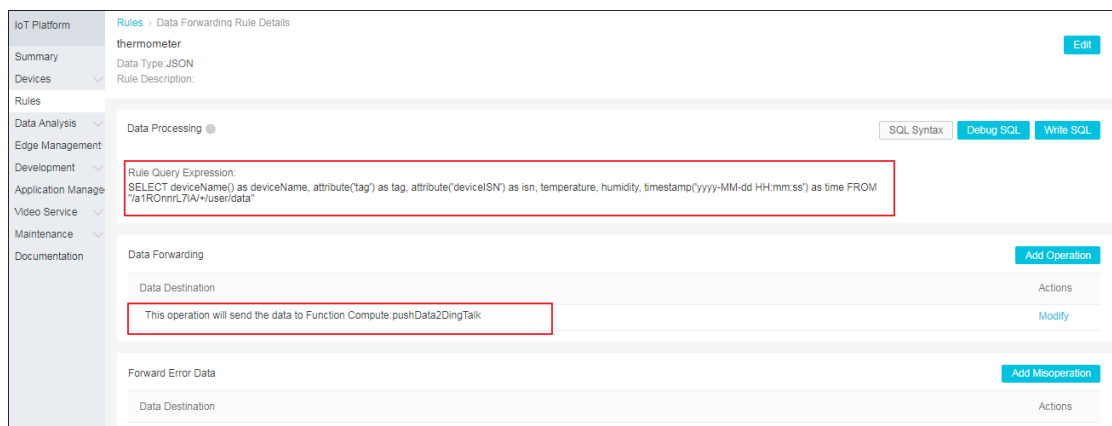
The SQL statements are as follows:

```
SELECT
deviceName () as deviceName ,
attribute (' tag ') as tag ,
attribute (' deviceISN ') as isn ,
temperatur e ,
humidity ,
timestamp (' yyyy - MM - dd HH : mm : ss ') as time
FROM
"/ Specify the productKey /+ / user / data "
```

B. Configure the data forwarding action.



The complete rule is as follows:



g) On the rules page, click Enable to enable the rule.

4. Temperature and humidity sensor.

To facilitate testing, a Node.js program is used to simulate the temperature and humidity sensor and send the temperature and humidity data. The [aliyun-iot-mqtt library](#) is used in this example. The complete code is as follows:

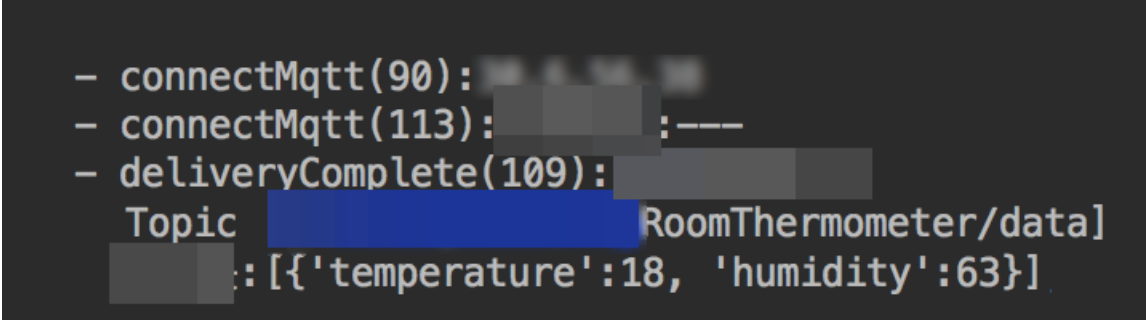
```
const mqtt = require (' aliyun - iot - mqtt ');
const client = mqtt . getAliyunIoTmqttClient ({
```

```
productKey : " Specify the productKey ",  
deviceName : " Specify the deviceName ",  
deviceSecret : " Specify the deviceSecret "  
});  
const topic = ' Specify the topic with forwarding  
actions '  
const data = {  
  temperature : 18 ,  
  humidity : 63 ,  
};  
client . publish ( topic , JSON . stringify ( data ));
```

5. DingTalk robot receives messages.

a) The program sends the temperature and humidity data.

```
$ npm install  
$ node demo . js
```



```
- connectMqtt(90): [REDACTED]  
- connectMqtt(113): [REDACTED]: ---  
- deliveryComplete(109): [REDACTED]  
  Topic [REDACTED]RoomThermometer/data]  
  [REDACTED]: [{ 'temperature': 18, 'humidity': 63 }]
```

b) DingTalk robot receives the data, and sends a message to the DingTalk group.

5 Set desired property values to control the light bulb status

You can set a desired property value for a device on the cloud. IoT Platform caches the value and pushes the value to the device when it is online .

Background information

A light bulb does not have storage capability. If you want to control the light bulb from IoT Platform, the light bulb must stay online always. However, the light bulb is rarely online.

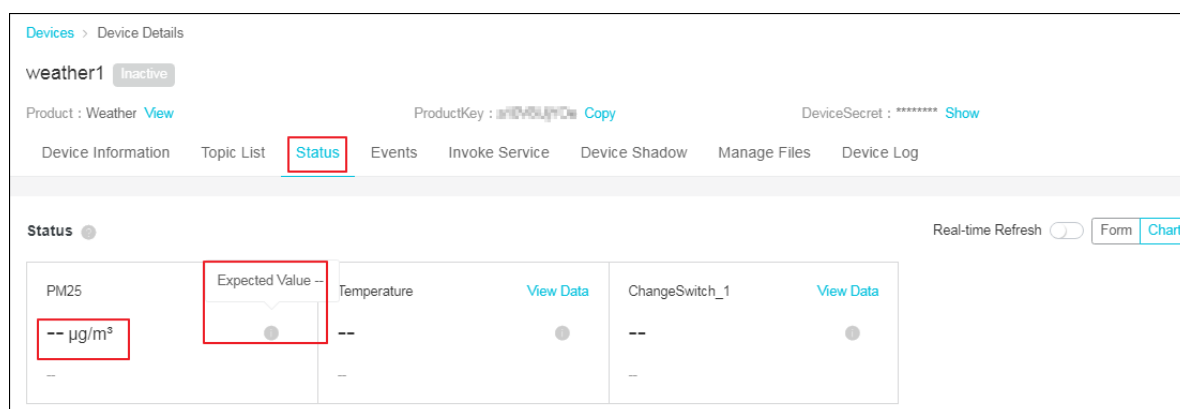
To resolve this issue, IoT Platform allows you to set desired property values.

When the light bulb is connected to IoT Platform, it reads the desired property value stored in IoT Platform. The device then updates the actual property based on the desired value and reports the updated property to IoT Platform. The reported property value will be stored as the latest property value.

Create a product and a device in the IoT Platform console

1. Log on to the [IoT Platform console](#).
2. Create a product named Street Lamp and a light bulb device named Lamp.
3. Go to the Device Details page of the device, and click the Status tab.

For a newly created device, the latest value and the desired value of a property are both "null", and the desired value version is 0.



4. Record the device certificate information, including ProductKey, DeviceName, and DeviceSecret, and the region information (RegionId).

Develop on the cloud

You call the related APIs to obtain the latest property values of a device and set desired property values to control the device.

For more information, see [API reference \(cloud\)](#). This example uses the [Java SDK \(cloud\)](#).

- Call `QueryDeviceDesiredProperty` to query the desired property values of the device.

```
DefaultProfile profile = DefaultProfile.getProfile (
    "< your - region - id >", // Your region ID
    "< your - access - key - id >", // AccessKey ID of
    your Alibaba Cloud account
    "< your - access - key - secret >"); // AccessKey Secret
    of your Alibaba Cloud account
IAcsClient client = new DefaultAcsClient ( profile );

// Set parameters to call the API
QueryDeviceDesiredPropertyRequest request = new
QueryDeviceDesiredPropertyRequest ();
request.setProductKey ( productKey );
request.setDeviceName ( deviceName );
// The list of property identifier s that you want
to query If you do not specify the property
identifier list, the desired values of all properties
except the read - only properties will be queried .
request.setIdentifiers ( Arrays.asList ( " LightStatus "));

// Initiate the request and handle the response or
exception
QueryDeviceDesiredPropertyResponse response ;
try {
    response = client.getAcsResponse ( request );

    QueryDeviceDesiredPropertyResponse.Data data =
response.getData ();
    for ( QueryDeviceDesiredPropertyResponse.Data .
DesiredPropertyInfo propInfo :
        data.getList () ) {
        // Return the desired value and version of
each property , such as property = LightStatus desired =
1 version = 5
        System.out.println ( " property =" + propInfo .
getIdentifier () );
        System.out.println ( " desired =" + propInfo . getValue
());
        System.out.println ( " version =" + propInfo .
getBizVersion () );
    }
} catch ( ServerException e ) {
    e.printStackTrace ();
} catch ( ClientException e ) {
    e.printStackTrace ();
}
```

- Call `SetDeviceDesiredProperty` to set desired property values for a device.

```
DefaultProfile profile = DefaultProfile.getProfile (
```

```

        "< your - region - id >", // Your region ID
        "< your - access - key - id >", // AccessKey ID of
your Alibaba Cloud account
        "< your - access - key - secret >"); // AccessKey Secret
of your Alibaba Cloud account
IAcsClient client = new DefaultAcsClient ( profile );

// Set parameters to call the API
SetDeviceDesiredPropertyRequest request = new
SetDeviceDesiredPropertyRequest ();
request . setProductKey ( productKey );
request . setDeviceName ( deviceName );
// The identifiers and values of the properties that
you want to set .
request . setItems ("{\"LightStatus\": 1}");
// You can set the version of a desired property
value if other RAM users also set desired values
for the property for distinguishment . Typically , you
can choose not to specify the version .
request . setVersions ("{}");

// Initiate the request and handle the response or
exception .
SetDeviceDesiredPropertyResponse response ;
try {
    response = client . getAcsResponse ( request );

    SetDeviceDesiredPropertyResponse . Data data =
response . getData ();
    // If the operation is successful , the version of
the desired property value is returned , such as {"
LightStatus": 5 }
    System . out . println ( data . getVersion s ());
} catch ( ServerException e ) {
    e . printStackTrace ();
} catch ( ClientException e ) {
    e . printStackTrace ();
}
}

```

Develop on the device

When the light bulb is connected to IoT Platform, it requests the desired property values from IoT Platform. If a desired property value changes while the light bulb is on, IoT Platform pushes the change in real time to the light bulb.

For more information, see [Developer Guide \(Devices\)](#). This example uses the Java SDK.

The last section of this topic provides a complete set of device demo code for your reference.

1. Enter the device certificate and region information.

```

/**
 * ProductKey , DeviceName , and DeviceSecret
 */
private static String productKey = "*****";
private static String deviceName = "*****";
private static String deviceSecret = "*****";
/**

```

```
* MQTT connection information
*/
private static String productKey = "*****";
```

2. Add the following methods. These methods can be used to change the actual property value of the light bulb and automatically report the changes to IoT Platform. The latest property value in IoT Platform will be replaced.

```
/**
 * When the device handles a property change, the
 * methods will be called in the following scenarios :
 * Scenario 1 Pull mode : After the device is
 * connected to IoT Platform, the device automatically
 * requests the latest desired property value from IoT
 * Platform .
 * Scenario 2 Push mode : While the device is online
 * , it receives the desired property value that IoT
 * Platform pushes to the topic property . set .
 * @param identifier The property identifier
 * @param value The desired property value
 * @param needReport Indicates whether to subscribe
 * to the topic property . post to send the property
 * value to IoT Platform
 * In scenario 2 , the property report capability is
 * already integrated into the processing function and
 * the needReport parameter will be set to " false ".
 * @ return
 */
private boolean handlePropertySet ( String identifier ,
ValueWrapper value , boolean needReport ) {
    ALog . d ( TAG , " The device handled property changes
= [" + identifier + "], value = [" + value + "]" );
    // Identify whether the property settings are
    successful according to the response . In this example
    , a success message is returned .
    boolean success = true ;
    if ( needReport ) {
        reportProperty ( identifier , value );
    }
    return success ;
}

private void reportProperty ( String identifier ,
ValueWrapper value ) {
    if ( StringUtil . isEmptyString ( identifier ) || value
== null ) {
        return ;
    }

    ALog . d ( TAG , " Reported property identity =" +
identifier );

    Map < String , ValueWrapper > reportData = new HashMap
<>();
    reportData . put ( identifier , value );
    LinkKit . getInstance () . getDeviceThing () . thingPrope
rtyPost ( reportData , new IPublishResourceListener () {

        public void onSuccess ( String s , Object o ) {
            // The property is reported
            ALog . d ( TAG , " Report success onSuccess ()
called with : s = [" + s + "], o = [" + o + "]" );
```

```

    }

    public void onError (String s , AError aError ) {
        // The property fails to be reported
        ALog . d ( TAG , " Report failure onError ()
called with : s = [" + s + "], aError = [" + JSON .
toJSONStri ng ( aError ) + "]" );
    }
});
}

```

3. When the light bulb is online, if a desired property value is set for the light bulb in IoT Platform, the value will be pushed to the light bulb. The light bulb then processes the message and changes the status accordingly.

If the device is offline when the desired value is set, use the `connectNot` `ifyListene r` method to register a function to respond to service calls and property settings. For information about the Alink protocol, see [Devices report properties](#).

After the device receives an asynchronous message from IoT Platform, the device calls the `mCommonHan dler` method and then the `handleProp ertySet` method to update the property value.

```

/**
 * Register a function to respond to service calls
 * and property settings .
 * When IoT Platform calls a service from the
 * device , the device must respond to the call and
 * return a response .
 */
public void connectNot ifyListene r () {
    List < Service > serviceLis t = LinkKit . getInstanc e ().
getDeviceT hing (). getService s ();
    for ( int i = 0 ; serviceLis t != null && i <
serviceLis t . size (); i ++ ) {
        Service service = serviceLis t . get ( i );
        LinkKit . getInstanc e (). getDeviceT hing ().
setService Handler ( service . getIdentif ier (), mCommonHan
dler );
    }
}

private IResReque stHandler mCommonHan dler = new
IResReque stHandler () {
    public void onProcess ( String serviceIde ntifier
, Object result , IResRespo nseCallbac k itResRespo
nseCallbac k ) {
        ALog . d ( TAG , " onProcess () called with : s = ["
+ serviceIde ntifier + "], " +
" o = [" + result + "], itResRespo nseCallbac
k = [" + itResRespo nseCallbac k + "]" );
        ALog . d ( TAG , " Received an asynchro nous service
call from IoT Platform " + serviceIde ntifier );
        try {
            if ( SERVICE_SE T . equals ( serviceIde ntifier )) {

```

```

        Map < String , ValueWrapp er > data = ( Map <
String , ValueWrapp er >)(( InputParam s ) result ). getData ();
        ALog . d ( TAG , " Received asynchrone us
downstream data " + data );
        // Set the property value of the device
        and then report the property value to IoT
Platform .

        boolean isSetPrope rtySuccess =
            handleProp ertySet (" LightStatu s ",
data . get (" LightStatu s " ), false );
        if ( isSetPrope rtySuccess ) {
            if ( result instanceof InputParam s ) {
                // Return a response to IoT
Platform
                itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , null );
            } else {
                itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , null );
            }
        } else {
            AError error = new AError ();
            error . setCode ( 100 );
            error . setMsg (" setProper tyFailed .");
            itResRespo nseCallbac k . onComplete (
serviceIde ntifier , new ErrorInfo ( error ), null );
        }
    } else if ( SERVICE_GE T . equals ( serviceIde
ntifier )) {
    } else {
        // The device operation varies by service
        .
        ALog . d ( TAG , " The returned values
correspond ing to the service .");
        OutputPara ms outputPara ms = new
OutputPara ms ();
        // outputPara ms . put (" op ", new ValueWrapp
er . IntValueWr apper ( 20 ));
        itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , outputPara ms );
    }
} catch ( Exception e ) {
    e . printStack Trace ();
    ALog . d ( TAG , " The data returned from IoT
Platform is incorrectl y formatted ");
}

    }

    public void onSuccess ( Object o , OutputPara ms
outputPara ms ) {
        ALog . d ( TAG , " onSuccess () called with : o = ["
+ o + "], outputPara ms = [" + outputPara ms + "]);
        ALog . d ( TAG , " The service has been registered
");
    }

    public void onFail ( Object o , ErrorInfo errorInfo )
    {
        ALog . d ( TAG , " onFail () called with : o = [" + o
+ "], errorInfo = [" + errorInfo + "]);
        ALog . d ( TAG , " Service registrati on failed .");
    }
}

```

```
};
```

4. When the light bulb is offline, if a desired property value is set for the light bulb in IoT Platform, this value will be stored in IoT Platform.

After the light bulb comes online again, it will request for the desired property value from IoT Platform and call the `handlePropertySet` method to update the property value.

```
LinkKit . getInstance (). init ( params , new ILinkKitConnectListener () {
    public void onError ( AError aError ) {
        ALog . e ( TAG , " Init Error error =" + aError );
    }
    public void onInitDone ( InitResult initResult ) {
        ALog . i ( TAG , " onInitDone result =" + initResult );

        connectNotifyListener ();

        // Obtain the latest desired property value
        getDesiredProperty ( deviceInfo , Arrays . asList ( "
LightStatus " ), new IConnectSessionListener () {
            public void onResponse ( ARequest aRequest ,
AResponse aResponse ) {
                if ( aRequest instanceof MqttPublishRequest
&& aResponse . data != null ) {
                    JSONObject jsonObject = JSONObject .
parseObject ( aResponse . data . toString () );
                    ALog . i ( TAG , " onResponse result =" +
jsonObject );
                    JSONObject dataObj = jsonObject .
getJSONObject ( " data " );
                    if ( dataObj != null ) {
                        if ( dataObj . getJSONObject ( "
LightStatus " ) == null ) {
                            // No desired value is set
                        } else {
                            Integer value = dataObj .
getJSONObject ( " LightStatus " ). getInteger ( " value " );
                            handlePropertySet ( " LightStatus
", new ValueWrapper . IntValueWrapper ( value ), true );
                        }
                    }
                }
            }
        }
    }
    public void onFailure ( ARequest aRequest ,
AError aError ) {
        ALog . d ( TAG , " onFailure () called with :
aRequest = [" + aRequest + "], aError = [" + aError + "]" );
    }
});

private void getDesiredProperty ( BaseInfo info , List <
String > properties , IConnectSessionListener listener ) {
    ALog . d ( TAG , " getDesiredProperty () called with :
info = [" + info + "], listener = [" + listener + "]" );
```

```

        if ( info != null && ! StringUtil.isEmptyString (
            info . productKey ) && ! StringUtil.isEmptyString ( info .
            deviceName ) ) {
            MqttPublishRequest request = new MqttPublish
            Request ();
            request . topic = DESIRED_PROPERTY_GET . replace ("{
            productKey }", info . productKey ). replace ("{ deviceName }",
            info . deviceName );
            request . replyTopic = DESIRED_PROPERTY_GET _REPLY
            . replace ("{ productKey }", info . productKey ). replace ("{
            deviceName }", info . deviceName );
            request . isRPC = true ;
            RequestModel < List < String >> model = new
            RequestModel <>();
            model . id = StringUtil.valueOf ( IDGeneratorUtils .
            getId () );
            model . method = METHOD_GET _DESIRED_PROPERTY ;
            model . params = properties ;
            model . version = " 1 . 0 ";
            request . payloadObj = model . toString ();
            ALog . d ( TAG , " getDesired Property : payloadObj =" +
            request . payloadObj );
            ConnectSDK . getInstance (). send ( request , listener
            );
        } else {
            ALog . w ( TAG , " getDesired Property failed ,
            baseInfo Empty .");
            if ( listener != null ) {
                AError error = new AError ();
                error . setMsg (" BaseInfoEmpty .");
                listener . onFailure ( null , error );
            }
        }
    }
}

```

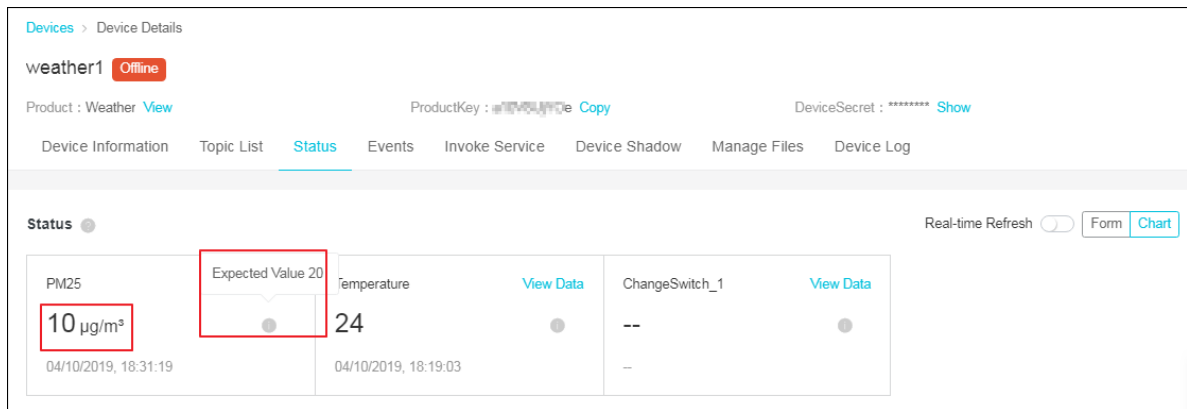
Verify the configuration

Verify that you can change the light bulb status by setting the desired value in IoT Platform regardless of whether the light bulb is online or not.

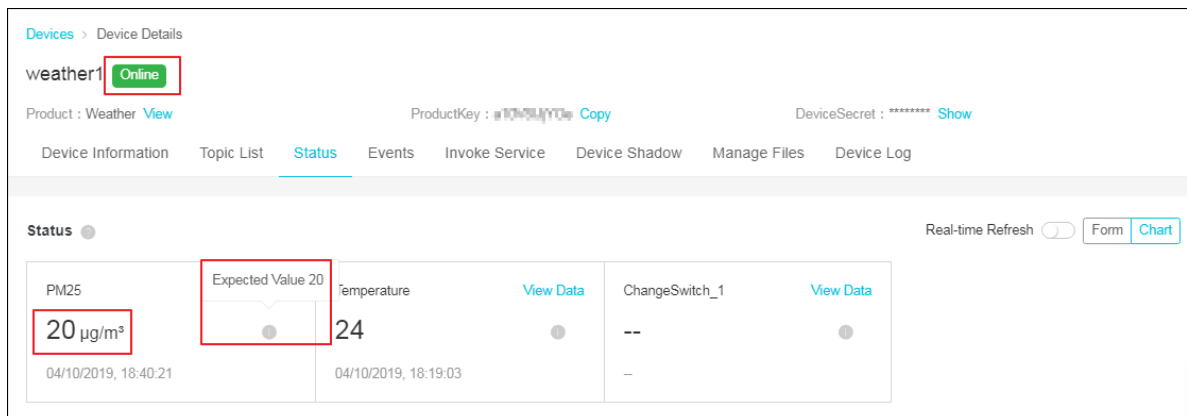
- When the light bulb is online, you change the light bulb status in IoT Platform and the light bulb responds to the change in real time.

The screenshot shows the 'Device Details' page for a device named 'weather1' in the IoT Platform. The 'Status' tab is active, displaying real-time data for three properties: PM25 (10 µg/m³), Temperature (24), and ChangeSwitch_1 (--). Red boxes highlight the 'Expected Value 10' for Temperature and the '10 µg/m³' value for PM25. The page includes a 'Real-time Refresh' toggle and buttons for 'Form' and 'Chart'.

- When the light bulb is offline, you change the light bulb status in IoT Platform. As a result, the desired value and the latest property value displayed in IoT Platform console are different.



- After the light bulb comes online again, the light bulb requests the desired property value. The latest property value will be immediately synchronized from the desired value.



Conclusion: To change the light bulb status, you can simply modify the desired property value. You can use the desired value to control the status of the light bulb that has no storage capability.

SDK demo code

```
package    com . aliyun . alink . devicesdk . demo ;

import    com . alibaba . fastjson . JSON ;
import    com . alibaba . fastjson . JSONObject ;
import    com . aliyun . alink . apiclient . utils . StringUtil ;
import    com . aliyun . alink . dm . api . BaseInfo ;
import    com . aliyun . alink . dm . api . DeviceInfo ;
import    com . aliyun . alink . dm . api . InitResult ;
import    com . aliyun . alink . dm . model . RequestModel ;
import    com . aliyun . alink . dm . utils . IDGenerateUtils ;
import    com . aliyun . alink . linkkit . api . ILinkKitConnectListener ;
import    com . aliyun . alink . linkkit . api . IoTMQTTClientConfig ;
```

```

import com . aliyun . alink . linkkit . api . LinkKit ;
import com . aliyun . alink . linkkit . api . LinkKitInitParams ;
import com . aliyun . alink . linksdk . cmp . api . ConnectSDK ;
import com . aliyun . alink . linksdk . cmp . connect . channel .
MqttPublisherRequest ;
import com . aliyun . alink . linksdk . cmp . core . base .
ARequest ;
import com . aliyun . alink . linksdk . cmp . core . base .
AResponse ;
import com . aliyun . alink . linksdk . cmp . core . listener .
IConnectSessionListener ;
import com . aliyun . alink . linksdk . tmp . api . InputParameters ;
import com . aliyun . alink . linksdk . tmp . api . OutputParameters ;
;
import com . aliyun . alink . linksdk . tmp . device . payload .
ValueWrapper ;
import com . aliyun . alink . linksdk . tmp . devicemodel .
Service ;
import com . aliyun . alink . linksdk . tmp . listener .
IPublishResourceListener ;
import com . aliyun . alink . linksdk . tmp . listener .
ITResourceHandler ;
import com . aliyun . alink . linksdk . tmp . listener .
ITResourceCallback ;
import com . aliyun . alink . linksdk . tmp . utils . ErrorInfo ;
import com . aliyun . alink . linksdk . tools . AError ;
import com . aliyun . alink . linksdk . tools . ALog ;

import java . util . Arrays ;
import java . util . HashMap ;
import java . util . List ;
import java . util . Map ;

public class LampDemo {
    private static final String TAG = " LampDemo ";

    private final static String SERVICE_SET = " set ";
    private final static String SERVICE_GET = " get ";

    public static String DESIRED_PROPERTY_GET = "/ sys /{
productKey }/{ deviceName }/ thing / property / desired / get ";
    public static String DESIRED_PROPERTY_GET_REPLY =
"/ sys /{ productKey }/{ deviceName }/ thing / property / desired /
get_reply ";
    public static String METHOD_GET_DESIRED_PROPERTY = "
thing . property . desired . get ";

    public static void main ( String [] args ) {
        /**
         * ProductKey , DeviceName , and DeviceSecret
         */
        String productKey = "****";
        String deviceName = " Lamp ";
        String deviceSecret = "****";
        /**
         * MQTT connection information
         */
        String regionId = " cn - shanghai ";

        LampDemo manager = new LampDemo ();

        DeviceInfo deviceInfo = new DeviceInfo ();
        deviceInfo . productKey = productKey ;
        deviceInfo . deviceName = deviceName ;

```

```

        deviceInfo . deviceSecret = deviceSecret ;
        manager . init ( deviceInfo , regionId );
    }

    public void init ( final DeviceInfo deviceInfo , String
region ) {
        LinkKitInitParams params = new LinkKitInitParams
();
        /**
         * Set the MQTT initialization parameters
         */
        IoTMqttClientConfig config = new IoTMqttClient
entConfig ();
        config . productKey = deviceInfo . productKey ;
        config . deviceName = deviceInfo . deviceName ;
        config . deviceSecret = deviceInfo . deviceSecret ;
        config . channelHost = deviceInfo . productKey + ".iot
- as - mqtt ." + region + ". aliyuncs . com : 1883 ";
        /**
         * Indicates whether to receive offline messages
         * The cleanSession field corresponding to the
MQTT connection
         */
        config . receiveOfflineMsg = false ;
        params . mqttClientConfig = config ;

        /**
         * Pass in the device certificate information
for initialization
         */
        params . deviceInfo = deviceInfo ;

        LinkKit . getInstance (). init ( params , new
ILinkKitConnectListener () {
            public void onError ( AError aError ) {
                ALog . e ( TAG , " Init Error error =" + aError
);
            }
            public void onInitDone ( InitResult initResult ) {
                ALog . i ( TAG , " onInitDone result =" +
initResult );

                connectNotifListener ();

                // Obtain the latest desired property value
from IoT Platform
                getDesiredProperty ( deviceInfo , Arrays . asList
(" LightStatus " ), new IConnectSessionListener () {
                    public void onResponse ( ARequest aRequest
, AResponse aResponse ) {
                        if ( aRequest instanceof MqttPublish
Request && aResponse . data != null ) {
                            JSONObject jsonObject = JSONObject .
parseObject ( aResponse . data . toString ());
                            ALog . i ( TAG , " onResponse result ="
+ jsonObject );
                            JSONObject dataObj = jsonObject .
getJSONObject ( " data " );
                            if ( dataObj != null ) {
                                if ( dataObj . getJSONObject ( "
LightStatus " ) == null ) {
                                    // No desired value is
set

```

```

        } else {
            Integer value = dataObj .
getJSONObject (" LightStatus "). getInteger (" value ");
            handlePropertySet ("
LightStatus ", new ValueWrapper . IntValueWrapper ( value
), true );
        }
    }
}

public void onFailure ( ARequest aRequest
, AError aError ) {
    ALog . d ( TAG , " onFailure () called
with : aRequest = [" + aRequest + "], aError = [" + aError +
""] );
}
});
}
});
}

/**
 * When the device handles a property change , the
methods will be called in the following scenarios :
 * Scenario 1 . Pull mode : After the device is
connected to IoT Platform again , the device automatica
lly requests the latest desired property value from
IoT Platform .
 * Scenario 2 . Push mode : While the device is
online , it receives the desired property value that
IoT Platform pushes to the property .set topic .
 * @param identifier The property identifier
 * @param value The desired property value
 * @param needReport Indicates whether to subscribe
to the property .post topic to send the property
value to IoT Platform
 * In scenario 2 , the property report capability is
already integrated into the processing function and
the needReport parameter will be set to " false ".
 * @return
 */
private boolean handlePropertySet ( String identifier ,
ValueWrapper value , boolean needReport ) {
    ALog . d ( TAG , " The device handled property
changes = [" + identifier + "], value = [" + value + ""] );
    // Identify whether the property settings are
successful according to the response . In this example
, a success message is returned .
    boolean success = true ;
    if ( needReport ) {
        reportProperty ( identifier , value );
    }
    return success ;
}

private void reportProperty ( String identifier ,
ValueWrapper value ){
    if ( StringUtil . isEmptyStr ing ( identifier ) ||
value == null ) {
        return ;
    }

    ALog . d ( TAG , " Reported property identity =" +
identifier );

```

```

        Map < String , ValueWrapp er > reportData = new
HashMap <>();
        reportData . put ( identifier , value );
        LinkKit . getInstanc e (). getDeviceT hing (). thingPrope
rtyPost ( reportData , new IPublishRe sourceList ener () {

            public void onSuccess ( String s , Object o ) {
                // The property is reported
                ALog . d ( TAG , " Report success onSuccess ()
called with : s = [" + s + "], o = [" + o + "]" );
            }

            public void onError ( String s , AError aError )
{
                // The property fails to be reported
                ALog . d ( TAG , " Report failure onError
() called with : s = [" + s + "], aError = [" + JSON .
toJSONStri ng ( aError ) + "]" );
            }
        }
    );

    /**
     * Register a function to respond to service calls
     and property settings .
     * When IoT Platform calls a service from the
     device , the device must respond to the call and
     return a response .
     */
    public void connectNot ifyListene r () {
        List < Service > serviceLis t = LinkKit . getInstanc e
(). getDeviceT hing (). getService s ();
        for ( int i = 0 ; serviceLis t != null && i <
serviceLis t . size (); i ++ ) {
            Service service = serviceLis t . get ( i );
            LinkKit . getInstanc e (). getDeviceT hing ().
setService Handler ( service . getIdentif ier (), mCommonHan dler
);
        }
    }

    private IResReque stHandler mCommonHan dler = new
IResReque stHandler () {
        public void onProcess ( String serviceIde ntifier
, Object result , IResRespo nseCallbac k itResRespo
nseCallbac k ) {
            ALog . d ( TAG , " onProcess () called with : s = ["
+ serviceIde ntifier + "], " +
                " o = [" + result + "], itResRespo
nseCallbac k = [" + itResRespo nseCallbac k + "]" );
            ALog . d ( TAG , " Received an asynchrono us
service call from IoT Platform " + serviceIde ntifier );
            try {
                if ( SERVICE_SE T . equals ( serviceIde ntifier ))
{
                    Map < String , ValueWrapp er > data = ( Map <
String , ValueWrapp er > )( InputParam s ) result ). getData ();
                    ALog . d ( TAG , " Received asynchrono us
downstream data " + data );
                    // Set the property of the device and
                    then report the property value to IoT Platform .
                    boolean isSetPrope rtySuccess =

```

```

        handlePropertySet (" LightStatus ",
data . get (" LightStatus "), false );
        if ( isSetPropertySuccess ) {
            if ( result instanceof InputParams ) {
                // Return a response to IoT
Platform
                itResResponseCallback . onComplete (
serviceId ntifier , null , null );
            } else {
                itResResponseCallback . onComplete (
serviceId ntifier , null , null );
            }
        } else {
            AError error = new AError ();
            error . setCode ( 100 );
            error . setMsg (" setPropertyFailed .");
            itResResponseCallback . onComplete (
serviceId ntifier , new ErrorInfo ( error ), null );
        }
    } else if ( SERVICE_GET . equals ( serviceId
ntifier )) {
        } else {
            // The device operation varies by
service .
            ALog . d ( TAG , " The returned values
corresponding to the service .");
            OutputParams outputParams = new
OutputParams ();
            // outputParams . put (" op ", new
ValueWrapper . IntValueWrapper ( 20 ));
            itResResponseCallback . onComplete (
serviceId ntifier , null , outputParams );
        }
    } catch ( Exception e ) {
        e . printStackTrace ();
        ALog . d ( TAG , " The data returned from IoT
Platform is incorrectly formatted ");
    }
}

public void onSuccess ( Object o , OutputParams
outputParams ) {
    ALog . d ( TAG , " onSuccess () called with : o = ["
+ o + "]", outputParams = [" + outputParams + "]);
    ALog . d ( TAG , " The service was registered ");
}

public void onFail ( Object o , ErrorInfo errorInfo
) {
    ALog . d ( TAG , " onFail () called with : o = ["
+ o + "]", errorInfo = [" + errorInfo + "]);
    ALog . d ( TAG , " Service registration failed .");
}

private void getDesiredProperty ( BaseInfo info , List <
String > properties , IConnectSensorListener listener ) {
    ALog . d ( TAG , " getDesiredProperty () called with :
info = [" + info + "]", listener = [" + listener + "]);
    if ( info != null && ! StringUtil . isEmptyString (
info . productKey ) && ! StringUtil . isEmptyString ( info .
deviceName )) {
        MqttPublishRequest request = new MqttPublish
hRequest ();
    }
}

```

```

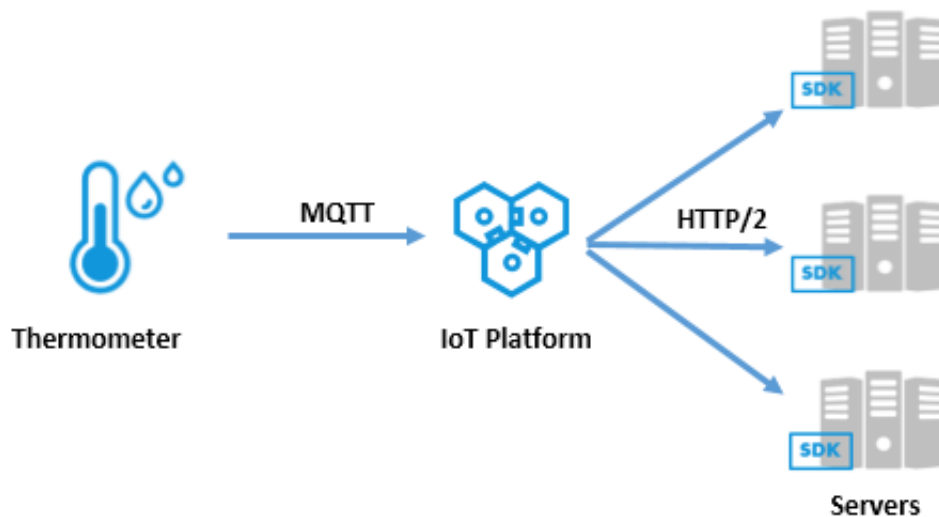
        request . topic = DESIRED_PR_OPERTY_GET . replace ("{
productKey }", info . productKey ). replace ("{ deviceName }", info
. deviceName );
        request . replyTopic = DESIRED_PR_OPERTY_GET _REPLY
. replace ("{ productKey }", info . productKey ). replace ("{
deviceName }", info . deviceName );
        request . isRPC = true ;
RequestMod el < List < String >> model = new
el <>();
model . id = String . valueOf ( IDGenerate rUtils .
getId ());
        model . method = METHOD_GET _DESIRED_P_RROPERTY ;
        model . params = properties ;
        model . version = " 1 . 0 ";
        request . payloadObj = model . toString ();
        Alog . d ( TAG , " getDesired Property : payloadObj ="
+ request . payloadObj );
        ConnectSDK . getInstanc e (). send ( request , listener
);
    } else {
        Alog . w ( TAG , " getDesired Property failed ,
baseInfo Empty .");
        if ( listener != null ) {
            AError error = new AError ();
            error . setMsg (" BaseInfoEm pty .");
            listener . onFailure ( null , error );
        }
    }
}
}
}

```

6 Configure service subscription

In some IoT scenarios, device status information and device collected data are expected to be directly received by business servers rather than be forwarded by cloud services. In such a case, you can use the [service subscription](#) function. You can push device messages to the business servers directly through the HTTP/2 channel, so that users can consume data based on their own business scenarios.

This topic describes how to use service subscription for data forwarding by using a hygrothermograph in an example to construct a client.



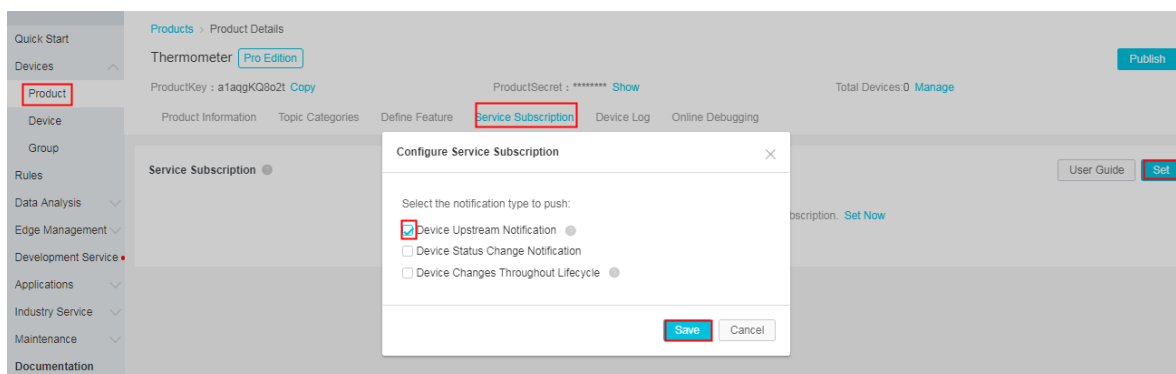
Prerequisites

Before you begin, create a product and a device in the IoT Platform console and connect the device to IoT Platform. For more information, see [Quick Start](#).

Configure service subscription

1. Log on to the [IoT Platform console](#), and choose Devices > Product from the left-side navigation pane.
2. In the product list, locate the product for which you want to configure service subscription and click View. The Product Details page appears.
3. Choose Service Subscription > Set.

4. In the Configure Service Subscription dialog box, select Device Upstream Notification, and click Save.



Use the service subscription SDK



Note:

IoT Platform supports Java 8 SDK.

1. Add the Java SDK dependencies as follows:

```
<!-- Aliyun core -->
<dependency>
  <groupId> com . aliyun </ groupId >
  <artifactId> aliyun - java - sdk - core </ artifactId >
  <version> 3 . 7 . 1 </ version >
</ dependency >

<!-- iot message client -->
<dependency>
  <groupId> com . aliyun . openservic es </ groupId >
  <artifactId> iot - client - message </ artifactId >
  <version> 1 . 1 . 2 </ version >
</ dependency >
```

2. Connect the SDK to IoT Platform using your Alibaba Cloud account information for identity authentication.

```
import java . net . UnknownHostException ;
import java . util . concurrent . ExecutionException ;

import com . aliyun . openservic es . iot . api . Profile ;
import com . aliyun . openservic es . iot . api . message .
MessageClientFactory ;
import com . aliyun . openservic es . iot . api . message . api .
MessageClient ;
import com . aliyun . openservic es . iot . api . message .
callback . MessageCallback ;
import com . aliyun . openservic es . iot . api . message .
entity . Message ;

public class H2Client {

    public static void main ( String [] args ) throws
UnknownHostException ,
```

```

ExecutionException , InterruptedException
dException {
    // Identity information
    String accessKey = "Your AccessKey ID ";
    String accessSecret = "Your AccessKey Secret >";
    String regionId = "cn-shanghai";
    String uid = "Your Alibaba Cloud UID ";
    String endPoint = "https://" + uid + ".iot-as-http2." + regionId + ".aliyuncs.com";
    // Configure connection parameters
    Profile profile = Profile.getAccessKeyProfile (
        endPoint, regionId, accessKey, accessSecret);

    // Construct a client
    MessageClient client = MessageClientFactory.
        messageClient (profile);
    // Receive data
    client.connect (messageToken -> {
        Message m = messageToken.getMessage ();
        System.out.println ("\\ntopic =" + m.getTopic ());
        System.out.println ("payload =" + new String (
            m.getPayload ());
        System.out.println ("generateTime =" + m.
            getGenerateTime ());
        // CommitSuccess indicates that the current
        message has been consumed and will be deleted by
        IoT Platform. If the message is not marked with
        CommitSuccess, the message will be retained until
        it expires.
        return MessageCallback.Action.CommitSuccess;
    });
}

```

3. The device reports the message to IoT Platform and the client receives the device message.

Data forwarding example

- Sample data of devices without TLS models

The device comes online, publishes a temperature and humidity message, and goes offline.

```

{
  "temperature": 27,
  "humidity": 26
}

```

The business server receives the following messages:

```

# Device connection message
topic = /as/mqtt/status/a1wLOMQOK4K/temperature-Monitor
payload = {
  "lastTime": "2018-09-04 14:59:59.782",
  "utcLastTime": "2018-09-04T06:59:59.782Z",
  "clientId": "42.120.75.152",
  "utcTime": "2018-09-04T06:59:59.794Z",
  "time": "2018-09-04 14:59:59.794",
}

```

```

    " productKey ": " a1wLOMQOK4 K ",
    " deviceName ": " temperatur e - Monitor ",
    " status ": " online "
  }
  generateTime = 1536044399 797

# Data message reported by the device
topic = / a1wLOMQOK4 K / temperatur e - Monitor / temperatur
eData
payload = {
  " temperatur e ": 27 ,
  " humidity ": 26
}
generateTime = 1536044404 762

# Device disconnection message
topic = / as / mqtt / status / a1wLOMQOK4 K / temperatur e -
Monitor
payload = {
  " lastTime ": " 2018 - 09 - 04 15 : 00 : 09 . 761 ",
  " utcLastTime ": " 2018 - 09 - 04T07 : 00 : 09 . 761Z ",
  " utcTime ": " 2018 - 09 - 04T07 : 00 : 13 . 337Z ",
  " time ": " 2018 - 09 - 04 15 : 00 : 13 . 337 ",
  " productKey ": " a1wLOMQOK4 K ",
  " deviceName ": " temperatur e - Monitor ",
  " status ": " offline "
}
generateTime = 1536044413 339

```

**Note:**

The service subscription function is based on the UID of your Alibaba Cloud account. All messages of the products that are created by this account will be received by the HTTP/2 client. You can differentiate service consumption among business servers by the ProductKey and DeviceName information that is contained in the topic.

- Sample data of devices using TSL models

The device comes online, publishes a temperature and humidity message, and goes offline.

```

{
  " id ": 1536045963 829 ,
  " params ": {
    " temperatur e ": 20 ,
    " humidity ": 22
  },
  " method ": " thing . event . property . post "
}

```

The business server receives the following messages:

```

# Device connection message
topic = / as / mqtt / status / a1gMuCoK4m 2 / nuwr5r9hf6 1
payload = {
  " lastTime ": " 2018 - 09 - 04 15 : 25 : 53 . 771 ",

```

```

    " utcLastTime ": " 2018 - 09 - 04T07 : 25 : 53 . 771Z ",
    " clientIp ": " 42 . 120 . 75 . 152 ",
    " utcTime ": " 2018 - 09 - 04T07 : 25 : 53 . 787Z ",
    " time ": " 2018 - 09 - 04 15 : 25 : 53 . 787 ",
    " productKey ": " algMuCoK4m 2 ",
    " deviceName ": " nuwr5r9hf6 1 ",
    " status ": " online "
  }
}
generateTime = 1536045953 787

# Device - reported data message that is parsed based
on the TSL model
topic = / algMuCoK4m 2 / nuwr5r9hf6 1 / thing / event / property /
post
payload = {
  " deviceType ": " None ",
  " iotId ": " hutRN9iuB v05Q0clH4f 0010885100 ",
  " productKey ": " algMuCoK4m 2 ",
  " gmtCreate ": 1536045958 765 ,
  " deviceName ": " nuwr5r9hf6 1 ",
  " items ": {
    " temperature ": {
      " value ": 20 ,
      " time ": 1536045958 770
    },
    " humidity ": {
      " value ": 22 ,
      " time ": 1536045958 770
    }
  }
}
generateTime = 1536045958 770

# Device disconnect ion message
topic = / as / mqtt / status / algMuCoK4m 2 / nuwr5r9hf6 1
payload = {
  " lastTime ": " 2018 - 09 - 04 15 : 26 : 03 . 749 ",
  " utcLastTime ": " 2018 - 09 - 04T07 : 26 : 03 . 749Z ",
  " utcTime ": " 2018 - 09 - 04T07 : 26 : 06 . 586Z ",
  " time ": " 2018 - 09 - 04 15 : 26 : 06 . 586 ",
  " productKey ": " algMuCoK4m 2 ",
  " deviceName ": " nuwr5r9hf6 1 ",
  " status ": " offline "
}
generateTime = 1536045966 588

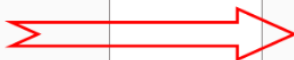
```



Note:

The messages of devices using TSL are not in the original format. The data will be parsed based on TSL models.

```
{
  "id": 1536045963829,
  "params": {
    "temperature": 20,
    "humidity": 22
  },
  "method": "thing.event.property.post"
}
```



```
topic=/algMuCoK4m2/nuwr5r9hf6l/thing/event/property/post
payload={
  "deviceType": "None",
  "iotId": "hutrRN9iuBvO5QOc1H4f0010885100",
  "productKey": "algMuCoK4m2",
  "gmtCreate": 1536045958765,
  "deviceName": "nuwr5r9hf6l",
  "items": {
    "temperature": {
      "value": 20,
      "time": 1536045958770
    },
    "humidity": {
      "value": 22,
      "time": 1536045958770
    }
  }
}
generateTime=1536045958770
```

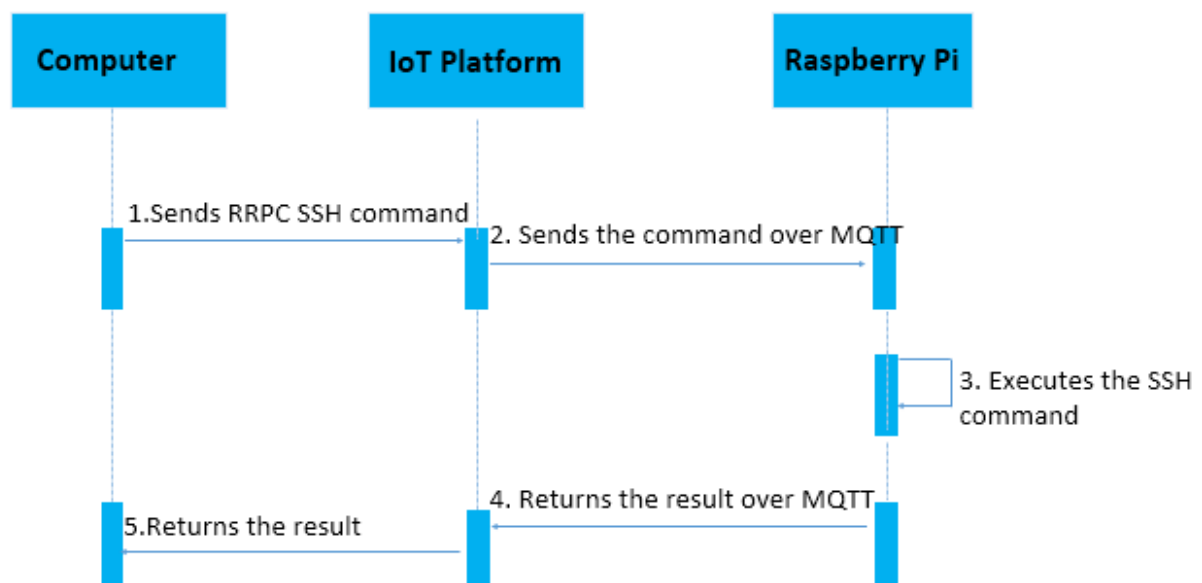
7 Control Raspberry Pi servers from IoT Platform

Alibaba Cloud IoT Platform allows you to remotely control Raspberry Pi servers without public IP addresses. This topic describes how to control Raspberry Pi servers from IoT Platform, and provides sample development code.

Background information

Assume that you use Raspberry Pi to build a server at your company or home to run some simple tasks, such as starting a script or downloading files. However, if the Raspberry Pi server does not have a public IP address, you cannot control the server when you are not in the company or at home. If you use other NAT traversal tools to control the server, disconnection may occur frequently. To resolve these issues, you can combine the IoT Platform [RRPC](#) (remote synchronous process call) function with [JSch](#) to control the Raspberry Pi server from IoT Platform.

Process



Use the following process to control a Raspberry Pi server from IoT Platform:

- Call the IoT Platform [RRPC method](#) from your computer to send an SSH command.
- After receiving the SSH command, IoT Platform sends the SSH command to the Raspberry Pi server by using MQTT.
- The server runs the SSH command.

- The server encapsulates the command output into an RRPC response and reports the response to IoT Platform by using MQTT.
- IoT Platform returns the RRPC response to the computer.

**Note:**

The RRPC call timeout period is 5 seconds. If IoT Platform does not receive any reply from the Raspberry Pi server within 5 seconds, a timeout error will be returned. If the command you have sent takes a relatively long time to process, you can ignore the timeout error.

Download SDKs and demos

To remotely control Raspberry Pi servers, you must first develop the server SDK and device SDK.

- Install the IoT Platform [server SDK](#) on the computer. To develop the server, you can use the [server Java SDK demo](#).
- Install the IoT Platform [device SDK](#) on the Raspberry Pi server. To develop the device, use the [device Java SDK Demo](#).

The following sections provide examples about how to develop the server SDK and device SDK.

**Note:**

In the examples, only simple Linux commands are supported in the code, such as `uname`, `touch`, and `mv`. Complex commands such as file editing are not supported. To implement complex commands, you must write code as needed.

Develop the device SDK

After you have downloaded and installed the device SDK and SDK demo, add project dependencies and the following Java files.

The project can be exported as a JAR package and run on the Raspberry Pi server.

1. Add dependencies to the `pom.xml` file.

```
<!-- Device SDK -->
<dependency>
  <groupId>com.aliyun.alink.linksdk</groupId>
  <artifactId>iot-linkkit-java</artifactId>
  <version>1.1.0</version>
  <scope>compile</scope>
</dependency>
```

```

< dependency >
  < groupId > com . google . code . gson </ groupId >
  < artifactId > gson </ artifactId >
  < version > 2 . 8 . 1 </ version >
  < scope > compile </ scope >
</ dependency >
< dependency >
  < groupId > com . alibaba </ groupId >
  < artifactId > fastjson </ artifactId >
  < version > 1 . 2 . 40 </ version >
  < scope > compile </ scope >
</ dependency >

<!-- SSH client -->
<!-- https://mvnrepository.com/artifact/com.jcraft/jsch -->
< dependency >
  < groupId > com . jcraft </ groupId >
  < artifactId > jsch </ artifactId >
  < version > 0 . 1 . 55 </ version >
</ dependency >

```

2. Add the `SSHShell . java` file that is used to run SSH commands.

```

public class SSHShell {

    private String host ;

    private String username ;

    private String password ;

    private int port ;

    private Vector < String > stdout ;

    public SSHShell ( final String ipAddress , final
String username , final String password , final int
port ) {
        this . host = ipAddress ;
        this . username = username ;
        this . password = password ;
        this . port = port ;
        this . stdout = new Vector < String > ();
    }

    public int execute ( final String command ) {

        System . out . println ( " ssh command : " + command );

        int returnCode = 0 ;
        JSch jsch = new JSch ();
        SSHUserInfo userInfo = new SSHUserInfo ();

        try {
            Session session = jsch . getSession ( username ,
host , port );
            session . setPassword ( password );
            session . setUserInfo ( userInfo );
            session . connect ();

            Channel channel = session . openChannel ( " exec
");

```

```

        (( ChannelExec c ) channel ). setCommand ( command );

        channel . setInputStream ( null );
        BufferedReader input = new BufferedReader (
new InputStreamReader ( channel . getInputStream () ));

        channel . connect ();

        String line = null ;
        while (( line = input . readLine () ) != null ) {
            stdout . add ( line );
        }
        input . close ();

        if ( channel . isClosed () ) {
            returnCode = channel . getExitStatus ();
        }

        channel . disconnect ();
        session . disconnect ();
    } catch ( JSchException e ) {
        e . printStackTrace ();
    } catch ( Exception e ) {
        e . printStackTrace ();
    }

    return returnCode ;
}

public Vector < String > getStdout () {
    return stdout ;
}
}

```

3. Add the *SSHUserInfo.java* file that is used to verify the SSH account and password.

```

public class SSHUserInfo implements UserInfo {

    @ Override
    public String getPassphrase () {
        return null ;
    }

    @ Override
    public String getPassword () {
        return null ;
    }

    @ Override
    public boolean promptPassphrase ( final String arg0 )
    {
        return false ;
    }

    @ Override
    public boolean promptPassword ( final String arg0 ) {
        return false ;
    }

    @ Override

```

```

        public boolean promptYesNo ( final String arg0 ) {
            if ( arg0 . contains ( " The authenticity of host
")) {
                return true ;
            }
            return false ;
        }

        @ Override
        public void showMessage ( final String arg0 ) {
        }
    }
}

```

4. Add the *Device . java* file that is used to establish an MQTT connection.

```

public class Device {

    /**
     * Establish a connection
     *
     * @ param productKey The product key
     * @ param deviceName The device name
     * @ param deviceSecret The device secret
     * @ throws InterruptedException
     */
    public static void connect ( String productKey
, String deviceName , String deviceSecret ) throws
InterruptedException {

        // Initialization parameters
        LinkKitInitParams params = new LinkKitInitParams
();

        // Set MQTT initialization parameters
        IoTMqttClientConfig config = new IoTMqttClient
entConfig ();
        config . productKey = productKey ;
        config . deviceName = deviceName ;
        config . deviceSecret = deviceSecret ;
        params . mqttClientConfig = config ;

        // Set device certificate information for
initialization and import the certificate information
n
        DeviceInfo deviceInfo = new DeviceInfo ();
        deviceInfo . productKey = productKey ;
        deviceInfo . deviceName = deviceName ;
        deviceInfo . deviceSecret = deviceSecret ;
        params . deviceInfo = deviceInfo ;

        // Initialize the SDK
        LinkKit . getInstance (). init ( params , new
ILinkKitConnector () {
            public void onError ( AError aError ) {
                System . out . println ( " init failed !! code
=" + aError . getCode () + " , msg =" + aError . getMsg () + " ,
subCode ="
                + aError . getSubCode () + " , subMsg =" +
aError . getSubMsg () );
            }
        }
    }
}

```

```

        public void onInitDone ( InitResult  initResult )
        {
            System . out . println ( " init  success  !!") ;
        }
    });

    // Perform the subsequent operations only after
the initialization is complete . You can increase
the sleep time as needed .
    Thread . sleep ( 2000 );
}

/**
 * Publish a message
 *
 * @ param topic The topic to which the message
is published
 * @ param payload The message payload
 */
public static void publish ( String topic , String
payload ) {
    MqttPublis hRequest request = new MqttPublis
hRequest ();
    request . topic = topic ;
    request . payloadObj = payload ;
    request . qos = 0 ;
    LinkKit . getInstanc e (). getMqttCli ent (). publish (
request , new IConnectSe ndListener () {
        @ Override
        public void onResponse ( ARequest aRequest ,
AResponse aResponse ) {
        }

        @ Override
        public void onFailure ( ARequest aRequest ,
AError aError ) {
        }
    });
}

/**
 * Subscribe to a topic
 *
 * @ param topic The topic of messages to
subscribe to
 */
public static void subscribe ( String topic ) {
    MqttSubscr ibeRequest request = new MqttSubscr
ibeRequest ();
    request . topic = topic ;
    request . isSubscrib e = true ;
    LinkKit . getInstanc e (). getMqttCli ent (). subscribe (
request , new IConnectSu bscribeLis tener () {
        @ Override
        public void onSuccess () {
        }

        @ Override
        public void onFailure ( AError aError ) {
        }
    });
}

/**

```

```

    * Unsubscribe from a topic
    *
    * @param topic The topic of messages to
unsubscribe from
    */
    public static void unsubscribe ( String topic ) {
        MqttSubscribeRequest request = new MqttSubscribeRequest ();
        request . topic = topic ;
        request . isSubscribe = false ;
        LinkKit . getInstance (). getMqttClient (). unsubscribe
e ( request , new IConnectUnsubscribeListener () {
            @Override
            public void onSuccess () {

            }

            @Override
            public void onFailure ( AError aError ) {

            }
        });
    }

    /**
    * Terminate the connection
    */
    public static void disconnect () {
        // Perform the deinitialization
        LinkKit . getInstance (). deinit ();
    }
}

```

5. Add the `SSHDevice . java` file. The `SSHDevice . java` file includes the main method. You can call the main method to receive RRPC commands, call `SSHShell` to run SSH commands, and return RRPC responses. In the `SSHDevice . java` file, you must enter the device certificate information, including `ProductKey`, `DeviceName`, and `DeviceSecret`, and the SSH account and password.

```

public class SSHDevice {

    //===== Start to Enter Required Parameters
    //=====
    // ProductKey
    private static String productKey = "";
    //
    private static String deviceName = "";
    // DeviceSecret
    private static String deviceSecret = "";
    // The message communication topic . You can
directly use the topic without creating or defining
the topic .
    private static String rrpcTopic = "/" + sys + "/" +
productKey + "/" + deviceName + "/" + rrpc + "/request +";
    // The domain name or IP address that you want
to access by using SSH
    private static String host = " 127 . 0 . 0 . 1 ";
    // The SSH username
    private static String username = "";
    // The SSH password
    private static String password = "";
}

```

```

// The SSH port
private static int port = 22 ;
//===================================================== End =====

public static void main ( String [] args ) throws
InterruptedException {

    // Listen to downstream data
    registerNotifyListener ();

    // Establish the connection
    Device . connect ( productKey , deviceName , deviceSecret );

    // Subscribe to a topic
    Device . subscribe ( rpcTopic );
}

public static void registerNotifyListener () {
    LinkKit . getInstance (). registerOn NotifyListener (
new IConnectNotifyListener () {
        @Override
        public boolean shouldHandle ( String connectId
, String topic ) {
            // Only process messages of the specified
            topic
            if ( topic . contains ("/ rpc / request /")) {
                return true ;
            } else {
                return false ;
            }
        }

        @Override
        public void onNotify ( String connectId , String
topic , AMessage aMessage ) {
            // Receive RPC requests and return RPC
            responses
            try {
                // Run the SSH command
                String payload = new String (( byte [])
aMessage . getData (), " UTF - 8 ");
                SSHShell sshExecutor = new SSHShell (
host , username , password , port );
                sshExecutor . execute ( payload );

                // Obtain the command output
                StringBuffer sb = new StringBuffer
();
                Vector < String > stdout = sshExecutor .
getStdout ();
                for ( String str : stdout ) {
                    sb . append ( str );
                    sb . append ("\ n ");
                }

                // Return command output to the
                server
                String response = topic . replace ("/
request /", "/ response /");
                Device . publish ( response , sb . toString
());
            } catch ( UnsupportedEncodingException e ) {
                e . printStackTrace ();
            }
        }
    }
}

```

```

        }

        @Override
        public void onConnectStateChange ( String
connectId , ConnectState connectState ) {
        }
    });
}
}

```

Develop the server SDK

After you have downloaded and installed the server SDK and SDK demo, add project dependencies and the following Java files.

1. Add dependencies to the `pom.xml` file.

```

<!-- Server SDK -->
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-iot</artifactId>
    <version>6.5.0</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.5.1</version>
</dependency>

<!-- commons - codec -->
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.8</version>
</dependency>

```

2. Add the `OpenApiClient.java` file that is used to call the IoT Platform API.

```

public class OpenApiClient {

    private static DefaultAcsClient client = null;

    public static DefaultAcsClient getClient ( String
accessKeyId , String accessKeySecret ) {

        if ( client != null ) {
            return client;
        }

        try {
            IClientProfile profile = DefaultProfile
.getProfile ( "cn-shanghai", accessKeyId , accessKeySecret );
            DefaultProfile.addEndpoint ( "cn-shanghai", "
cn-shanghai", "Iot", "iot.cn-shanghai.aliyuncs.com" );
            client = new DefaultAcsClient ( profile );
        } catch ( Exception e ) {

```

```

        System.out.println(" create Open API Client
failed !! exception:" + e.getMessage());
    }

    return client ;
}
}

```

3. Add the `SSHCommand Sender . java` file. The `SSHCommand Sender . java` file contains the main method that is used to send SSH commands and receive responses to SSH commands. In the `SSHCommand Sender . java` file, you must enter your Alibaba Cloud account information including AccessKey ID and AccessKey Secret, and device certificate information including ProductKey and DeviceName, and SSH command.

```

public class SSHCommand Sender {

    //===== Start to Enter Required Parameters
    //=====

    // AccessKey ID of your Alibaba Cloud account
    private static String accessKeyI D = "";
    // AccessKey Secret of your Alibaba Cloud account
    private static String accessKeyS ecret = "";
    // ProductKey
    private static String productKey = "";
    // DeviceName
    private static String deviceName = "";
    //===== End =====

    public static void main ( String [] args ) throws
ServerExce ption , ClientExce ption , Unsupporte dEncodingE
xception {

        // The Linux command
        String payload = " uname - a ";

        // Construct an RRPC request
        RRpcReques t request = new RRpcReques t ();
        request . setProduct Key ( productKey );
        request . setDeviceN ame ( deviceName );
    }
}

```

```

        request . setRequest Base64Byte ( Base64 . encodeBase
64String ( payload . getBytes ());

        request . setTimeout ( 5000 );

        // Obtain informatio n about the client connected
to the Raspberry Pi server

        DefaultAcs Client client = OpenApiCli ent . getClient
( accessKeyI D , accessKeyS ecret );

        // Initiate an RRPC request

        RRpcRespon se response = ( RRpcRespon se ) client .
getAcsResp onse ( request );

        // Process an RRPC response

        //" response . getSuccess ()" only indicates that the
RRPC request has been sent . It does not indicate
that the device has received the RRPC request and
has returned a response .

        // Identify whether the message has been
received and a response has been returned according
to the RrpcCode value . For more informatio n , see
the document of RRPC API https :// www . alibabaclo ud .
com / help / doc - detail / 69797 . htm .

        if ( response != null && " SUCCESS ". equals (
response . getRrpcCod e ())) {

            // Output the response

            System . out . println ( new String ( Base64 .
decodeBase 64 ( response . getPayload Base64Byte ()), " UTF - 8
"));

        } else {

            // An error occurred while outputting the
response and an RRPC code is displayed .

            System . out . println ( response . getRrpcCod e ());

        }

    }

}

```