

Alibaba Cloud IoT Platform

User Guide

Issue: 20190710

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.
<code>Courier</code> 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
{} or {a b}	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 Create products and devices.....	1
1.1 Create a product.....	1
1.2 Create devices.....	4
1.2.1 Create multiple devices at a time.....	4
1.2.2 Create a device.....	6
1.2.3 Manage devices.....	8
1.3 TSL.....	10
1.3.1 Overview.....	10
1.3.2 Define features.....	11
1.3.3 Import Thing Specification Language (TSL).....	25
1.3.4 The TSL format.....	26
1.4 Data parsing.....	29
1.4.1 Data parsing.....	29
1.5 Topics.....	41
1.5.1 What is a topic?.....	41
1.5.2 Create a topic category.....	44
1.6 Tags.....	46
1.7 Gateways and sub-devices.....	50
1.7.1 Gateways and sub-devices.....	50
1.7.2 Sub-device management.....	52
1.8 Service Subscription.....	52
1.8.1 What is service subscription?.....	53
1.8.2 Development guide for Java HTTP/2 SDK.....	54
1.8.3 Development guide for .NET HTTP/2 SDK.....	62
1.8.4 Limits.....	70
1.8.5 Subscribe to device messages by using Message Service.....	71
1.9 Device group.....	74
1.10 Manage files.....	77
2 Rules.....	80
2.1 Data Forwarding.....	80
2.1.1 Overview.....	80
2.1.2 Compare data forwarding solutions.....	81
2.1.3 Create and configure a rule.....	86
2.1.4 SQL statements.....	94
2.1.5 Functions.....	98
2.1.6 Data forwarding procedure.....	102
2.1.7 Data format.....	103
2.1.8 Regions and zones.....	110

2.2 Data Forwarding Examples.....	111
2.2.1 Forward data to another topic.....	111
2.2.2 Forward data to Table Store.....	112
2.2.3 Forward data to ApsaraDB for RDS.....	116
2.2.4 Forward data to Message Service.....	120
2.2.5 Forward data to Function Compute.....	126
3 Monitoring and Maintenance.....	132
3.1 Real-time monitoring.....	132
3.1.1 Real-time monitoring.....	132
3.2 Online debug.....	138
3.2.1 Debug applications using Physical Devices.....	138
3.2.2 Debug applications using virtual devices.....	139
3.3 Device log.....	142
3.4 Firmware update.....	162
3.5 Remote configuration.....	169
4 Generic protocol SDK.....	176
4.1 Overview.....	176
4.2 Use the basic features.....	179
4.3 Use the advanced features.....	191
5 RRPC.....	198
5.1 What is RRPC?.....	198
5.2 System-defined topics.....	199
5.3 Custom topics.....	200
6 Device shadows.....	202
6.1 Device Shadow overview.....	202
6.2 Device shadow JSON format.....	204
6.3 Device shadow data stream.....	208
7 Configure the NTP service.....	216
8 Accounts and logon.....	218
8.1 Log on to the console using the primary account.....	218
8.2 Resource Access Management (RAM).....	219
8.2.1 RAM and STS.....	219
8.2.2 Custom permissions.....	222
8.2.3 API permissions.....	229
8.2.4 Use RAM users.....	234
8.2.5 Advanced guide to STS.....	237

1 Create products and devices

This topic describes how to create and manage products and devices in the console.

1.1 Create a product

The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

Context

This topic describes how to create products in the IoT Platform console.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Product, and then click Create Product.
3. Enter all the required information and then click OK.

The parameters are as follows.

Parameter	Description
Product Name	The name of the product that you want to create. The product name must be unique within the account. For example, you can enter the product model as the product name. A product name is 4 to 30 characters in length, and can contain Chinese characters, English letters, digits, and underscores. A Chinese character counts as two characters.

Parameter	Description
Node Type	· Device: Indicates that devices of this product cannot be mounted with sub-devices. This kind of device can connect to IoT Platform directly or as a sub-device of a gateway device. · Gateway: Indicates that devices of this product connect to IoT Platform directly and can be mounted with sub-devices . A gateway can manage sub-devices, maintain topological relationships with sub-devices, and synchronize topological relationships to IoT Platform. For more information about gateway devices and sub-devices, see Gateways and sub-devices.
Connect to Gateway  Note: This parameter appears if the node type is Device.	Indicates whether or not devices of this product can be connected to gateways as sub-devices. · Yes: Devices of this product can be connected to a gateway. If you select Yes here, you are required to select a gateway connection protocol under Network Connection and Data. · No: Devices of this product cannot be connected to a gateway. If you select No here, you are required to select a network connection method under Network Connection and Data.
Gateway Connection Protocol  Note: This parameter appears if you select Yes for Connect to Gateway .	Select a protocol for sub-device and gateway communication. · Custom: Indicates that you want to use another protocol as the connection protocol for sub-device and gateway communication. · Modbus: Indicates that the communication protocol between sub-devices and gateways is Modbus. · OPC UA: Indicates that the communication protocol between sub-devices and gateways is OPC UA. · ZigBee: indicates that the communication protocol between sub-devices and gateways is ZigBee. · BLE: indicates that the communication protocol between sub-devices and gateways is BLE.

Parameter	Description
Network Connection Method  Note: This parameter appears if you select No for Connect to Gateway.	Select a network connection method for the devices: · WiFi · Cellular (2g/3g/4G) · Ethernet · Other
Data Type	Select a format in which devices exchange data with IoT Platform. Options are ICA Standard Data Format (Alink JSON) and Do not parse/Custom. · ICA Standard Data Format (Alink JSON): The standard data format defined by IoT Platform for device and IoT Platform communication. · Do not parse/Custom: If you want to customize the serial data format, select Do not parse/Custom. Custom formatted data must be converted to Alink JSON script by Data parsing so that your devices can communicate with the IoT Platform.
Product Description	Describe the product information. You can enter up to 100 characters.

After the product is created successfully, you are automatically redirected to the Products page.

What's next

1. To configure features for a product (such as [Notifications](#), [TSL \(Define Feature\)](#), and [Service Subscription](#)), go to the product list, find the target product and then click its corresponding View button.
2. Register devices on IoT Platform.
3. Develop your physical devices by referring to [Developer Guide \(Devices\)](#).

4. To publish a product, go to the product details page and click Publish.

The screenshot shows the 'Product Details' page for a product named 'test2'. The left sidebar contains navigation options: Products, Devices, Product, Device, Group, Rules, Extended Services, Maintenance, and Documentation. The main content area has tabs for Product Information, Topic Categories, Define Feature, Service Subscription, Device Log, and Online Debugging. The 'Product Information' tab is active, displaying a table with the following data:

Product Information					
Product Name	test2	Node Type	Device	Created At	10/21/2018, 20:45:34
Product Version	Pro Edition	Category	Water Meter	Data Type	ICA Standard Data Format (Alink JSON)
Dynamic Registration	Disabled	ProductSecret	*****	Show	
Status	Developing	Connect to Gateway	No	Connection Protocol	WiFi
Product Description					

Below the table is the 'Tag Information' section, which shows 'Product Tag: No tags, Add'. A red box highlights the 'Publish' button in the top right corner of the page.

Note that before you publish a product, you must make sure that you have configured all the correct information for the product, have completed debugging the features, and have verified that it meets the criteria for being published.

When the product status is Published, you can view the product information but cannot modify or delete the product.

The screenshot shows the 'Products' page with a sub-tab 'My products (128)'. It features a 'Product List' section with a search bar and a 'Create Product' button. The table below lists three products:

Product Name	ProductKey	Node Type	Created At	Actions
donotparse	*****	Device	01/25/2019, 15:07:12	View
Hygrothermograph	*****	Device	01/23/2019, 16:09:16	View Delete
newproduct2	*****	Gateway	01/16/2019, 10:09:26	View Delete

A red box highlights the 'View' link in the Actions column for the 'donotparse' product.

To cancel the publishing of a product, click Cancel Publishing.

1.2 Create devices

1.2.1 Create multiple devices at a time

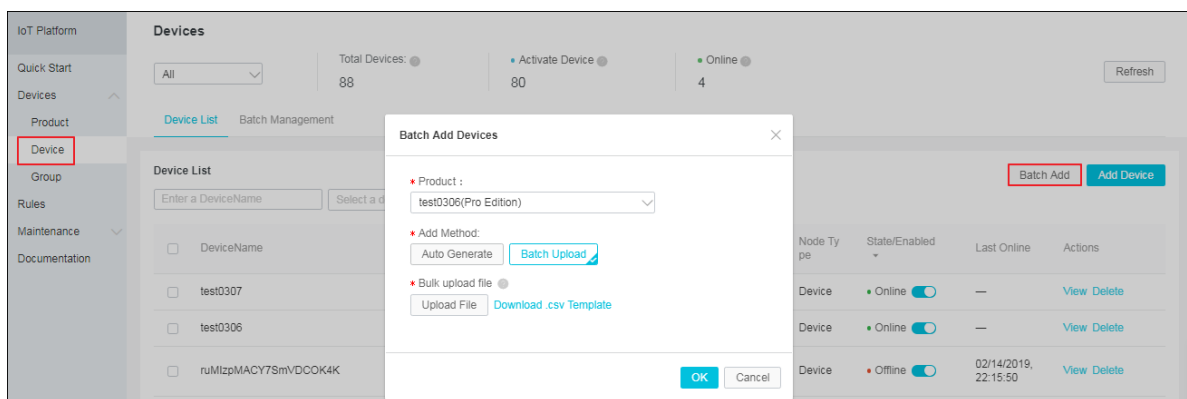
A product is a collection of devices. After you create products, you can create specific devices for the product models. You can create one device or multiple devices at a time. This topic explains how to create multiple devices at a time.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Device, and then click Batch Add.
3. Select a product that you have created. The devices to be created will be assigned with the features of the selected product.
4. Select how the devices are to be named. Two methods:
 - **Auto Generate:** You do not specify names for the devices that you want to create. You only specify the number of devices, and the system automatically generates names for the devices.
 - **Batch Upload:** You specify a name for each device you want to create. Under Upload File, click Download .csv Template to download the naming template. Enter device names in the template table and save the file. Then, click Upload File to upload the naming file.

**Note:**

- Device names must be 4-32 characters in length, and can contain English letters, digits, hyphens, underscores, @ symbols, dots, and colons.
- Each device name must be unique in the product.
- A file can include up to 1,000 names.
- The size of the file cannot exceed 2 MB.



5. Click OK to start batch device creation.
6. After the devices are successfully created, click Download Device Certificate to download the file containing the information of created devices.

Result

On the Batch Management tab page of Devices page, you can:

- Click View Details to view the detailed information of the devices.

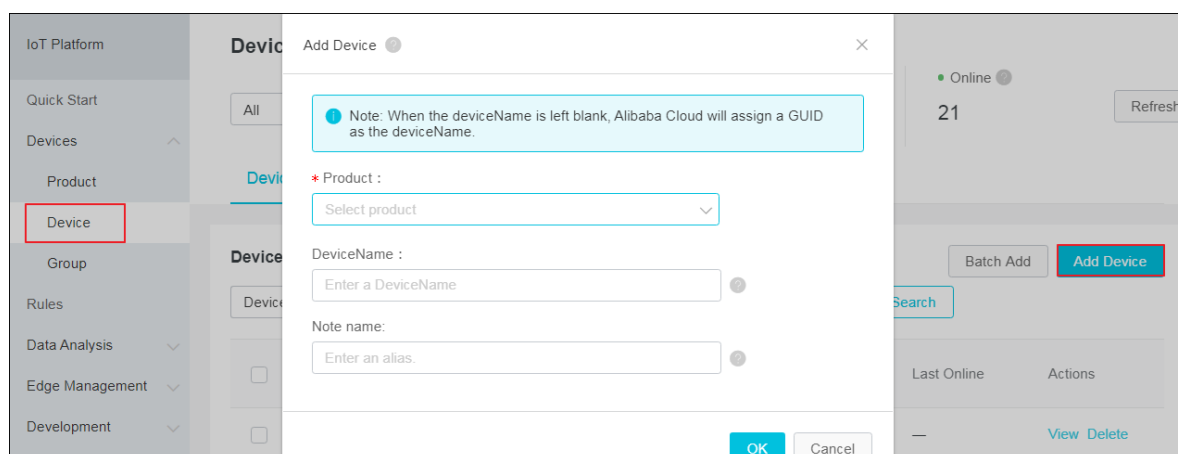
- Click Download CSV to download the certificates of the devices.

1.2.2 Create a device

A product is a collection of devices. After you have created a product, you must register devices under the product with IoT Platform. You can create devices individually or create multiple devices at one time. This topic describes how to create devices individually.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose Devices > Device, and then click Add Device.
3. In the Add Device dialog box, enter the device information and click OK.



Parameter	Description
Product	Select a product. The device to be created will be assigned the features and properties of the selected product.  Note: If the product is associated with another platform, make sure that your account has sufficient activation codes to create the device.

Parameter	Description
DeviceName	Set the device name. If you left this parameter empty, the system automatically generates a device name that contains numbers and letters. • The device name is unique within the product. • The device name must be 4 to 32 characters in length and can contain letters, numbers, and special characters. The supported special characters are hyphens (-), underscores (_), at signs (@), periods (.) , and colons (:).
Note name	Set the alias. The alias must be 4 to 64 characters in length and can contain Chinese characters, letters, numbers, and underscores (_). One Chinese character is counted as two characters.

Result

After the device is created, the View Device Certificate dialog box appears automatically. You can view and copy the device certificate information. A device certificate is the authentication certificate of a device when the device is communicating with IoT Platform. It contains *three key fields*: ProductKey, DeviceName, and DeviceSecret.

Parameter	Description
ProductKey	The key of the product to which the device belongs. It is the GUID that is issued by IoT Platform to the product.
DeviceName	The unique identifier of the device within the product. A device uses the DeviceName and the ProductKey as the device identifier to authenticate to and communicate with IoT Platform.
DeviceSecret	The device key issued by IoT Platform for device authentication and encryption. It must be used in pairs with the DeviceName.

You can also click View next to the newly created device on the Device List page. On the Device Details page, click the Device Information tab to view device information.

Devices > Device Details

test02 Inactive

Product : test_product [View](#) ProductKey : XXXXXXXXXX [Copy](#) DeviceSecret : ***** [Show](#)

[Device Information](#) [Topic List](#) [Status](#) [Events](#) [Invoke Service](#) [Device Shadow](#) [Manage Files](#) [Device Log](#)

Device Information

Product Name	test_product	ProductKey	XXXXXXXXXX Copy	Region	-
Node Type	Device	DeviceName	test02 Copy	DeviceSecret	***** Show
Alias ●	Edit	IP Address	-	Firmware Version	-
Created At	04/29/2019, 15:26:46	Activated At		Last Online	
Current Status	Inactive	Real-time Delay ●	Test		

More Device Information

SDK Language	-	Version	-	Module Manufacturer	-
Module Information	-				

Tag Information

Device Tag: No tags, [Add](#)

What's next

Follow instructions in [Device development documentation](#) to develop the device SDK.

1.2.3 Manage devices

After you create a device in IoT Platform, you can manage or view device information in the IoT Platform console.

Manage devices of an account

From the left-side navigation pane, choose Devices > Device. The Devices page appears.

Devices

All Total Devices: 150 Activate Device: 58 Online: 21 [Refresh](#)

[Device List](#) [Batch Management](#)

Device List

[Batch Add](#) [Add Device](#)

DeviceName... [Search](#)

☐	DeviceName/Alias	Product	Node Type	State/Enabled	Last Online	Actions
☐	abc9	aircleaner	Device	Inactive ●	—	View Delete
☐	abc4	aircleaner	Device	Inactive ●	—	View Delete
☐	abc8	aircleaner	Device	Inactive ●	—	View Delete
☐	abc6	aircleaner	Device	Inactive ●	—	View Delete
☐	abc11	aircleaner	Device	Inactive ●	—	View Delete
☐	abc2	aircleaner	Device	Inactive ●	—	View Delete
☐	abc10	aircleaner	Device	Inactive ●	—	View Delete
☐	abc1	aircleaner	Device	Inactive ●	—	View Delete

Task	Procedure
View devices under a specific product	Select a product in the upper-left corner of the page.
Search for a device	Enter a device name, note name, or device tag to search for a device. Fuzzy search is supported.
View detailed information about a device	Click View next to the corresponding device.
Delete a device	Click Delete next to the corresponding device.  Note: After a device is deleted, the device certificate becomes invalid and the data about this device in IoT Platform is deleted.

View detailed information about a device

In the device list, click View next to the corresponding device. The Device Details page appears.

Devices > Device Details

Xgateway1 Inactive

Product : Xgateway [View](#)

ProductKey :  a1CFMyM66xF [Copy](#)

DeviceSecret : ***** [Show](#)

[Device Information](#) [Topic List](#) [Status](#) [Events](#) [Invoke Service](#) [Device Shadow](#) [Manage Files](#) [Device Log](#) [Sub-device Management](#) [Sub-device Channels](#)

Device Information

Product Name	Xgateway	ProductKey	a1CFMyM66xF Copy	Region	-
Node Type	Gateway	DeviceName	Xgateway1 Copy	DeviceSecret	***** Show
Alias 	0419 Edit	IP Address	-	Firmware Version	-
Created At	04/19/2019, 10:18:16	Activated At		Last Online	
Current Status	Inactive	Real-time Delay 	Test		

More Device Information

SDK Language	-	Version	-	Module Manufacturer	-
Module Information	-				

Tag Information

Device Tag: No tags, [Add](#)

Task	Procedure
Activate the device	The Inactive status indicates that the device is not connected to IoT Platform. To develop the device and activate the device, see Download device SDKs .
View device information	View the basic information about the device, including device certificate information, firmware information , extended information , and tag information.

Task	Procedure
View device data	· On the Status tab page, view the latest values, data records, and desired values of properties. · On the Events tab page, view the records about device reported events. · On the Invoke Service tab page, view the service call records.
View device log	On the Device Log tab page, click Read Now to view the device log information. The information include device activities, upstream messages, downstream messages, TSL data, and QoS=1 message contents. For more information about device logs, see Device log .

1.3 TSL

1.3.1 Overview

Thing Specification Language (TSL) is a data model that digitizes a physical entity and constructs the entity data model in IoT Platform. In IoT Platform, a TSL model refers to a set of product features. After you have defined features for a product, the system automatically generates a TSL model of the product. A TSL model describes what a product is, what the product can do, and what services the product can provide.

A TSL model is a file in JSON format. TSL files are the digitized expressions of physical entities, such as sensors, vehicle-mounted devices, buildings and factories. A TSL file describes an entity in three dimensions: property (what the entity is), service (what the entity can do), and event (what event information the entity reports). Defining these three dimensions is to define the product features.

Therefore, the feature types of a product are Properties, Services and Events. You can define these three types of features in the console.

Feature type	Description
Property	Describes a running status of a device, such as the current temperature read by the environmental monitoring equipment . You can use GET and SET methods to send requests to get and set device properties.

Feature type	Description
Service	Indicates a feature or method of a device that can be used by a user. You can set input parameters and output parameters for a service. Compared with properties, services can implement more complex business logic, for example, a specific task.
Event	Indicates the notifications of a type of event occurred when a device is running. Events typically indicate notifications that require actions or attention, and they may contain multiple output parameters. For example, events can be notifications about the completion of tasks, system failures, or temperature alerts. You can subscribe to events or push events to a message receiving target.

Use TSL

1. In the IoT Platform console, [Define features](#) or [Import Thing Specification Language \(TSL\)](#).
2. Develop the SDK. See the documentations of [Link Kit SDK](#) for help information.
3. Connect the SDK to IoT Platform. Then, devices can report properties and events to IoT Platform, and in IoT Platform, you can set properties and call device services.

1.3.2 Define features

Defining features for products is to define Thing Specification Language (TSL), including defining properties, services, and events. This article describes how to define features in the IoT Platform console.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Product.
3. On the Products page, find the product for which you want to define features and click View.
4. Click Define Feature.

5. Add self-defined features. Click the Add Feature button corresponding to Self-defined Feature to add custom features for the product. You can define properties, services and events for the product.

The screenshot shows the 'Product Details' page for a product named 'testProduct13050'. The 'Define Feature' tab is selected. The 'Self-Defined Feature' section contains a table with three rows of properties. The 'Add Feature' button is highlighted in the top right corner of the table.

Feature Type	Feature Name	Identifier	Data Type	Data Definition	Actions
Property...	intPropertyName	intProperty	int32	Value Range:0 ~ 100	Edit Delete
Property...	floatPropertyName	floatProperty	float	Value Range:0 ~ 100	Edit Delete
Property...	doublePropertyName	doubleProperty	double	Value Range:0 ~ 100	Edit Delete

- Define a property. In the Add Self-defined Feature dialog box, select Properties as the feature type. Enter information for the property and then click OK.

* Feature Type:

Properties

Services

Events

* Feature Name:

* Identifier:

* Data Type:

* Value Range:

~

* Step :

Unit :

Read/Write Type:

☒ Read/Write ☐ Read-only

Description :

Enter a description

0/100

OK

Cancel

The parameters of properties are listed in the following table.

Parameter	Description
The function name	Property name, for example, Power Consumption. Each feature name must be unique in the product. A feature name must start with a Chinese character, an English letter, or a digit, can contain Chinese characters, English letters, digits, dashes(-) and underscores (_), and cannot exceed 30 characters in length.

Parameter	Description
Identifier	Identifies a property. It must be unique in the product. It is the parameter <code>identifier</code> in Alink JSON TSL, and is used as the key when a device is reporting data of this property. Specifically, IoT Platform uses this parameter to verify and determine whether or not to receive the data. An identifier can contain English letters, digits, and underscores (_), and cannot exceed 50 characters in length. For example, <code>PowerConsumption</code>.  Note: An identifier cannot be any one of the following words: <code>set</code>, <code>get</code>, <code>post</code>, <code>time</code>, and <code>value</code>, because they are system parameter names.

Parameter	Description
Data Type	- <code>int32</code> : 32-bit integer. If you select <code>int32</code>, you are required to define the value range, step, and unit. - <code>float</code> : Float. If you select <code>float</code>, you are required to define the value range, step, and unit. - <code>double</code> : Double float. If you select <code>double</code>, you are required to define the value range, step, and unit. - <code>enum</code> : Enumeration. You must specify enumeration items with values and descriptions. For example, 1 indicates heating mode and 2 indicates cooling mode. - <code>bool</code> : Boolean. You must specify the Boolean values. Values include 0 and 1. For example, you can use 0 to indicate disabled and 1 to indicate enabled. - <code>text</code> : Text string. You must specify the data length. The maximum value is 2048 bytes. - <code>date</code> : Timestamp. A UTC timestamp in string type, in milliseconds. - <code>struct</code> : A JSON structure. Define a JSON structure, and add new JSON parameters. For example, you can define that the color of a lamp is a structure composed of three parameters: red, green, and blue. Structure nesting is not supported. - <code>array</code> : Array. You must select a data type for the elements in the array from <code>int32</code>, <code>float</code>, <code>double</code>, <code>text</code> and <code>struct</code>. Make sure that the data type of elements in an array is the same and that the length of the array does not exceed 128 elements.  Note: When the gateway connection protocol is Modbus, you do not set this parameter.
Step	The smallest granularity of changes of properties, events, and input and output parameter values of services. If the data type is <code>int32</code> , <code>float</code> , or <code>double</code> , step is required.
Unit	You can select None or a unit suitable.

Parameter	Description
Read/Write Type	- Read / Write : GET and SET methods are supported for Read/Write requests. - Read - only : Only GET is supported for Read-only requests.  Note: When the gateway connection protocol is Modbus, you do not set this parameter.
Description	Enter a description or remarks about the property. You can enter up to 100 characters.

Parameter	Description
Extended Information	When the gateway connection protocol is Modbus or OPC UA, you can configure extended parameters. - When the gateway connection protocol is Modbus, configure the following parameters. ■ Operation Type: ■ Coil Status (read-only, 01) ■ Coil Status (read and write, 01-read, 05-write) ■ Coil Status (read and write, 01-read, 0F-write) ■ Discrete Input (read-only, 02) ■ Holding Registers (read-only, 03) ■ Holding Registers (read and write, 03-read, 06-write) ■ Holding Registers (read and write, 03-read, 10-write) ■ Input Registers (read-only, 04) ■ Register Address: Enter a hexadecimal address beginning with 0x. The range is 0x0 - 0xFFFF. For example, 0xFE. ■ Original Data Type: Multiple data types are supported, including int16, uint16, int32, uint32, int64, uint64, float, double, string, bool, and customized data (raw data). ■ Switch High Byte and Low Byte in Register: Swap the first 8 bits and the last 8 bits of the 16-bit data in the register. - Options: ■ true ■ false ■ Switch Register Bits Sequence: Swap the bits of the original 32-bit data. Options: ■ true ■ false ■ Zoom Factor: The zoom factor is set to 1 by default. It can be set to negative numbers, but cannot be set to 0. ■ Collection Interval: The time interval of data collection. It is in milliseconds and the value cannot be lower than 10. ■ Data Report: The trigger of data report. It can be either <code>At Specific Time</code> or <code>Report Changes</code>. - When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the property.

- Define a service. In the Add Self-defined Feature dialog box, select Services as the feature type. Enter information for the service and then click OK.



Note:

When the gateway connection protocol is Modbus, you cannot define any service for the product.

* Feature Type:

Properties **Services** Events

* Feature Name:

switch

* Identifier:

Switch

* Invoke Method::

☒ Asynchronous ☐ Synchronous

Input Parameters:

+ Add Parameter

Output Parameters:

+ Add Parameter

Description :

Enter a description

0/100

OK Cancel

The parameters of services are as follows.

Parameter	Description
The function name	Service name. A feature name must start with an English letter, Chinese character, or a number. It can contain English letters, Chinese characters, digits, dashes (-), and underscores (_), and cannot exceed 30 characters in length. If you have selected a category with feature template when you were creating the product, the system displays the standard services from the standard feature library for you to choose.  Note: When the gateway connection protocol is Modbus, you cannot define custom services for the product.
Identifier	Identifies a service. It must be unique within the product. The parameter <code>identifier</code> in Alink JSON TSL. It is used as the key when this service is called. An identifier can contain English letters, digits, and underscores (_), and cannot exceed 30 characters in length.  Note: Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value.
Invoke Method	- Asynchronous : For an asynchronous call, IoT Platform returns the result directly after the request is sent, and does not wait for a response from the device. - Synchronous : For a synchronous call, IoT Platform waits for a response from the device. If no response is received, the call times out.

Parameter	Description
Input Parameters	(Optional) Set input parameters for the service. Click Add Parameter, and add an input parameter in the dialog box that appears. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters.  Note: - Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value. - You can either use a property as an input parameter or define an input parameter. For example, you can specify the properties Sprinkling Interval and Sprinkling Amount as the input parameters of the Automatic Sprinkler service feature. Then, when Automatic Sprinkler is called, the sprinkler automatically starts irrigation according to the sprinkling interval and amount. - You can add up to 20 input parameters for a service.
Output Parameters	(Optional) Set output parameters for the service. Click Add Parameter, and add an output parameter in the dialog box that appears. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters.  Note: - Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value. - You can either use a property as an output parameter or define an output parameter. For example, you can specify the property SoilHumidity as an output parameter. Then, when the service Automatic Sprinkler is called, IoT Platform returns the data about soil humidity. - You can add up to 20 output parameters for a service.

Parameter	Description
Extended Information	When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the service.
Description	Enter a description or remarks about the service. You can enter up to 100 characters.

- Define an event. In the Add Self-defined Feature dialog box, select Events as the feature type. Enter information for the parameter and then click OK.



Note:

When the gateway connection protocol is Modbus, you cannot define any event for the product.

* Feature Type:

Properties Services **Events** ?

* Feature Name:

Alarm ?

* Identifier:

Alarm ?

* Event Type:

☒ Info ☐ Alert ☐ Error ?

Output Parameters:

☐ Parameter Name: current Edit Delete

+ Add Parameter

Description :

Enter a description

0/100

OK Cancel

The parameters of events are as follows.

Parameter	Description
The function name	Event name. A feature name must start with a Chinese character, an English letter, or a digit, can contain Chinese characters, English letters, digits, dashes(-) and underscores (_), and cannot exceed 30 characters in length.  Note: When the gateway connection protocol is Modbus, you cannot define events.

Parameter	Description
Identifier	Identifies an event. It must be unique in the product. It is the parameter <code>identifier</code> in Alink JSON TSL, and is used as the key when a device is reporting data of this event, for example, <code>ErrorCode</code>.  Note: Identifiers of input parameters cannot be any one of the following words: <code>set</code>, <code>get</code>, <code>post</code>, <code>time</code>, and <code>value</code>.
Event Type	- <code>Info</code> : Indicates general notifications reported by devices, such as the completion of a specific task. - <code>Alert</code> : Indicates alerts that are reported by devices when unexpected or abnormal events occur. It has a high priority. You can perform logic processing or analytics depending on the event type. - <code>Error</code> : Indicates errors that are reported by devices when unexpected or abnormal events occur. It has a high priority. You can perform logic processing or analytics depending on the event type.
Output Parameters	The output parameters of an event. Click Add Parameter, and add an output parameter in the dialog box that appears. You can either use a property as an output parameter or define an output parameter. For example, you can specify the property <code>Voltage</code> as an output parameter. Then, devices report errors with the current voltage value for further fault diagnosis. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters.  Note: - Identifiers of input parameters cannot be any one of the following words: <code>set</code>, <code>get</code>, <code>post</code>, <code>time</code>, and <code>value</code>. - You can add up to 50 output parameters for an event.
Extended Information	When the gateway collection protocol is OPC UA, set a node name. Each node name must be unique under the event.
Description	Enter a description or remarks about the event. You can enter up to 100 characters.

1.3.3 Import Thing Specification Language (TSL)

This article introduces how to import an existing TSL for a product.

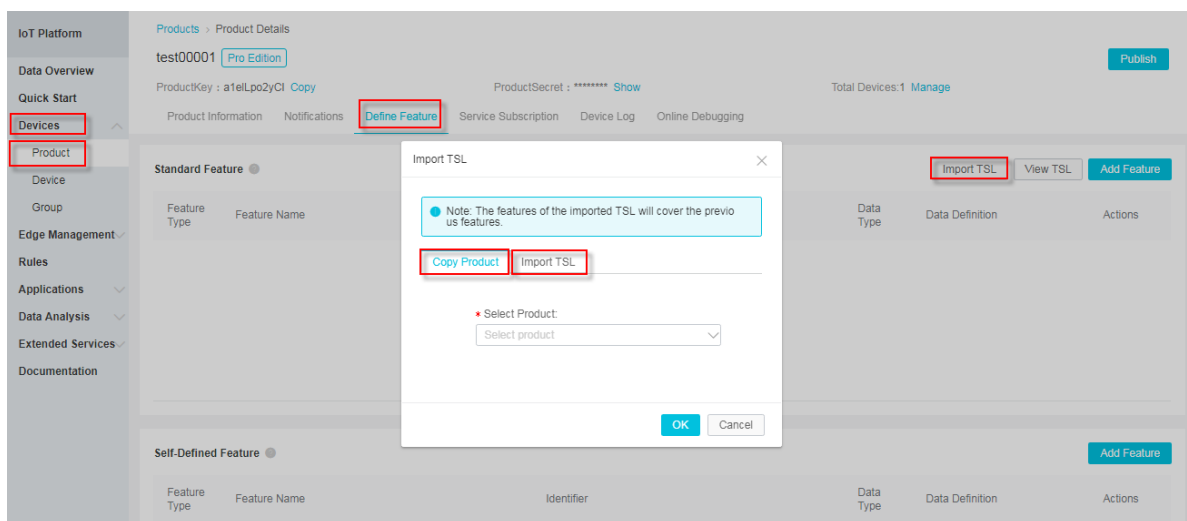
Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Product.
3. On the Products page, find the product for which you want to import TSL and click View.
4. Click Define Feature > Import TSL.



Note:

- The previously defined features of the product will be overwritten, once you have imported a new TSL for the product. Therefore, this function must be used with caution.
- You cannot import a TSL for a product whose gateway connection protocol is defined as Modbus.



You can import TSL in two ways:

- **Copy Product:** Copy the TSL of another product. Select an existing product and click OK to import the TSL of the selected product to this product.

If you want to modify some features, click Edit corresponding to the features on the Define Feature tab page.

- **Import TSL:** Paste your self-defined TSL script into the edit box and click OK.

The size of the imported file cannot exceed 64 KB.

1.3.4 The TSL format

The format of Thing Specification Language (TSL) is JSON. This article introduces the JSON fields of TSL.

In the Define Feature tab of your target product, click View TSL.

The following section details each JSON field.

```
{
  " schema ":" TSL schema of a thing ",
  " link ":" System - level URI in the cloud , used to
  invoke services and subscribe to events ",
  " profile ":{
    " productKey ":" Product ID ",
  },
  " properties ":[
    {
      " identifier ":" Identifies a property . It must
      be unique under a product ",
      " name ":" Property name ",
      " accessMode ":" Read / write type of properties ,
      including Read - Only and Read / Write ",
      " required ":" Determines whether a property
      that is required in the standard category is also
      required for a standard feature ",
      " dataType ":{
        " type ":" Data type : int ( original ), float (
        original ), double ( original ), text ( original ), date ( UTC
        string in millisecon ds ), bool ( integer , 0 or 1 ),
        enum ( integer ), struct ( supports int , float , double ,
        text , date , and bool ), array ( supports int , double ,
        float , and text )",
        " specs ":{
          " min ":" Minimum value , available only
          for the int , float , and double data types ",
          " max ":" Maximum value , available only
          for the int , float , and double data types ",
          " unit ":" Property unit ",
          " unitName ":" Unit name ",
          " size ":" Array size , up to 128
          elements , available only for the array data type ",
          " item ":{
            " type ":" Type of an array element "
          }
        }
      }
    }
  ],
  " events ":[
    {
      " identifier ":" Identifies an event that is
      unique under a product , where " post " are property
      events reported by default ",
      " name ":" Event name ",
      " desc ":" Event descriptio n ",
      " type ":" Event types , including info , alert ,
      and error ",
      " required ":" Whether the event is required for
      a standard feature ",
      " outputData ":[
```

```

        {
            " identifier ":" Uniquely identifies a
parameter ",
            " name ":" Parameter name ",
            " dataType ":{
                " type ":" Data type : int ( original ),
float ( original ), double ( original ), text ( original ),
date ( UTC string in millisecon ds ), bool ( integer , 0
or 1 ), enum ( integer ), struct ( supports int , float ,
double , text , date , and bool ), array ( supports int ,
double , float , and text )",
                " specs ":{
                    " min ":" Minimum value , available
only for the int , float , and double data types ",
                    " max ":" Maximum value , available
only for the int , float , and double data types ",
                    " unit ":" Property unit ",
                    " unitName ":" Unit name ",
                    " size ":" Array size , up to 128
elements , available only for the array data type ",
                    " item ":{
                        " type ":" Type of an array
element "
                    }
                }
            }
        },
        " method ":" Name of the method to invoke the
event , generated according to the identifier "
    ],
    " services ":[
        {
            " identifier ":" Identifies a service that is
unique under a product ( set and get are default
services generated according to the read / write type
of the property )",
            " name ":" Service name ",
            " desc ":" Service descriptio n ",
            " required ":" Whether the service is required
for a standard feature ",
            " inputData ":[
                {
                    " identifier ":" Uniquely identifies an
input parameter ",
                    " name ":" Name of an input parameter ",
                    " dataType ":{
                        " type ":" Data type : int ( original ),
float ( original ), double ( original ), text ( original ),
date ( UTC string in millisecon ds ), bool ( integer , 0
or 1 ), enum ( integer ), struct ( supports int , float ,
double , text , date , and bool ), array ( supports int ,
double , float , and text )",
                        " specs ":{
                            " min ":" Minimum value , available
only for the int , float , and double data types ",
                            " max ":" Maximum value , available
only for the int , float , and double data types ",
                            " unit ":" Property unit ",
                            " unitName ":" Unit name ",
                            " size ":" Array size , up to 128
elements , available only for the array data type ",
                            " item ":{

```

```

    " type ":" Type of an array
  element "
    }
  }
},
"outputData ":[
  {
    " identifier ":" Uniquely identifies an
    output parameter ",
    " name ":" Name of an output parameter ",
    " dataType ":{
      " type ":" Data type : int ( original ),
      float ( original ), double ( original ), text ( original ),
      date ( UTC string in millisecon ds ), bool ( integer , 0
      or 1 ), enum ( integer ), struct ( supports int , float ,
      double , text , date , and bool ), array ( supports int ,
      double , float , and text )",
      " specs ":{
        " min ":" Minimum value , available
        only for the int , float , and double data types ",
        " max ":" Maximum value , available
        only for the int , float , and double data types ",
        " unit ":" Property unit ",
        " unitName ":" Unit name ",
        " size ":" Array size , up to 128
        elements , available only for the array data type ",
        " item ":{
          " type ":" Type of an array
          element , available only for the array data type "
        }
      }
    }
  },
  " method ":" Name of the method to invoke
  the service , which is generated according to the
  identifier "
}
]
}

```

If the product is connected to a gateway as a sub-device and the connection protocol is Modbus or OPC UA, you can view the TSL extension configuration.

```

{
  " profile ": {
    " productKey ": " Product ID ",
  },
  " properties ": [
    {
      " identifier ": " Identifies a property . It must be unique
      under a product ",
      " operateType ": "( coilStatus / inputStatu s / holdingReg ister /
      inputRegis ter )",
      " registerAd dress ": " Register address ",
      " originalDa taType ": {
        " type ": " Data type : int16 , uint16 , int32 , uint32 , int64
        , uint64 , float , double , string , customized data ( returns
        hex data according to big - endian )",
        " specs ": {

```

```

" registerCount ": " The number of registers , available
only for string and customized data ",
" swap16 ": " swap the first 8 bits and the last 8
bits of the 16 bits of the register data ( for
example , byte1byte2 -> byte2byte1 0 ). Available for all
the other data types except string and customized
data ",
" reverseRegister ": " Ex : Swap the bits of the original
32 bits data ( for example , byte1byte2 byte3byte4 ->
byte3byte4 byte1byte2 ". Available for all the other data
types except string and customized data "
    }
},
" scaling ": " Scaling factor ",
" pollingTime ": " Polling interval . The unit is ms ",
" trigger ": " The trigger of data report . Currently ,
two types of triggering methods are supported : 1 :
report at the specified time ; 2 : report when changes
occurred "
    }
]
}

```

1.4 Data parsing

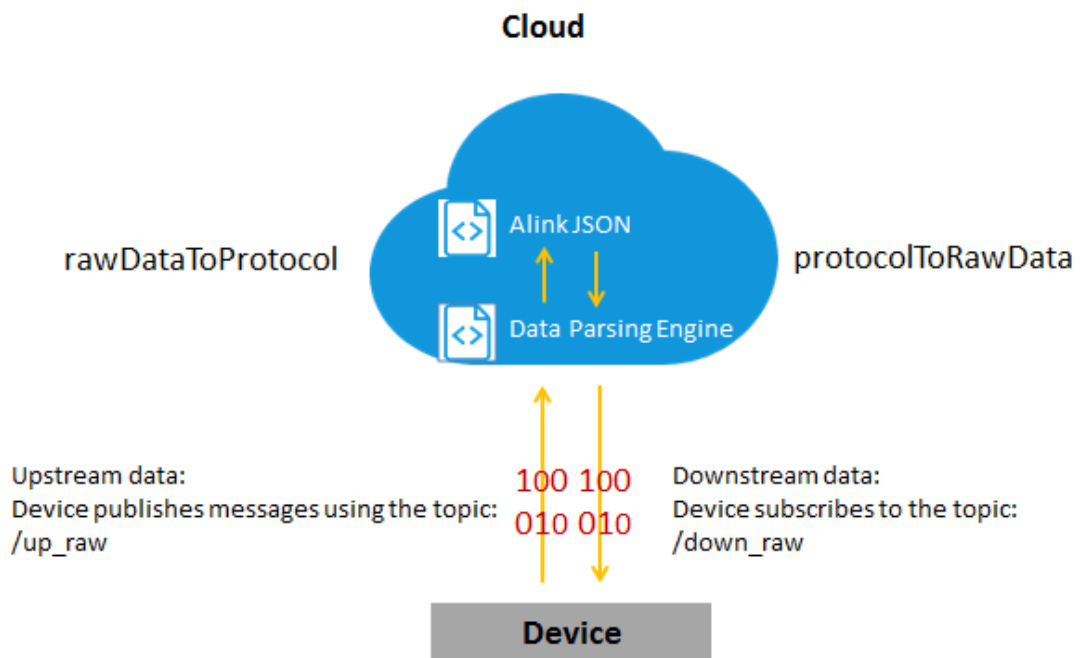
1.4.1 Data parsing

Devices with low configurations and limited resources or devices that have high requirements for network traffic can send raw data to IoT Platform. This prevents the devices from directly sending data to IoT Platform in Alink JSON format. You must write a data parsing script in the IoT Platform console to parse upstream and downstream data to be in standard Alink JSON format and the custom data format, respectively.

About data parsing

When receiving raw data from a device, IoT Platform runs the parsing script to convert the raw data to the Alink JSON data for business processing. When sending data to the device, IoT Platform also runs the parsing script to convert the Alink JSON data to the device custom formatted data.

Data parsing process:



For more information about sending data upstream and downstream, see "Devices report properties or events" and "Call device services or set device properties" in [Communications over Alink protocol](#).

Script format

```
/**
 * Convert data in Alink JSON format to data format
 * that can be identified by the device . This feature
 * is called when IoT Platform sends data to a device
 *
 * Input : jsonObj    Object    Required
 * Output : rawData   byte []   Array    Required
 *
 */
function protocolTo RawData ( jsonObj ) {
    return rawData ;
}

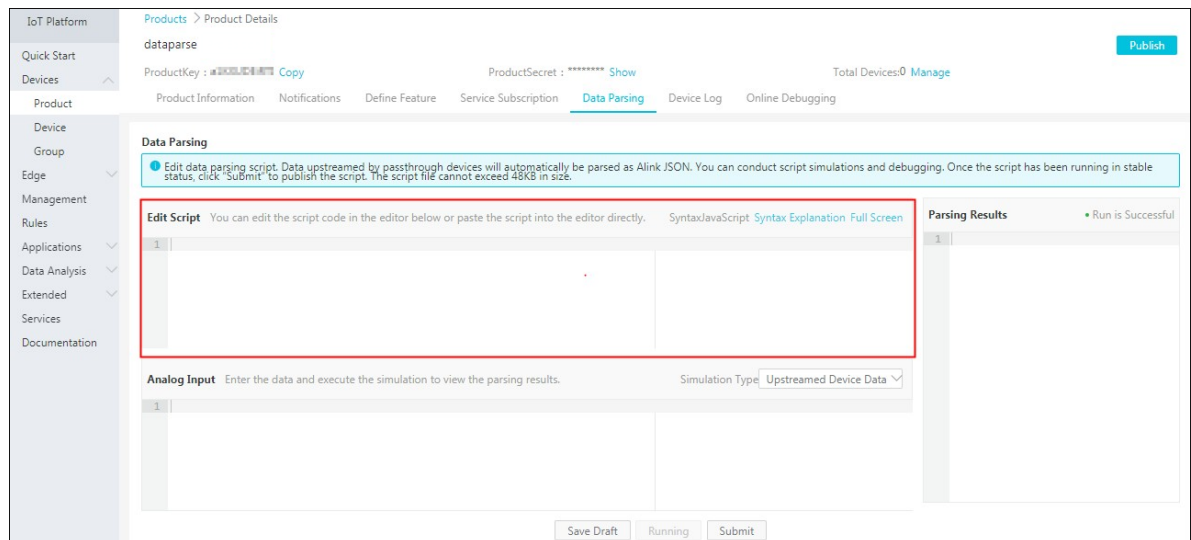
/**
 * Convert the custom formatted data to Alink JSON
 * data . This function is called when a device reports
 * data to IoT Platform .
 * Input : rawData   byte []   Array    Required
 * Output : jsonObj    Object    Required
 *
 */
function rawDataToP rotocol ( rawData ) {
    return jsonObj ;
}
```

```
}
```

Edit and verify scripts

Only JavaScript is supported to edit scripts. IoT Platform provides an online script editor that allows you to edit and submit scripts, and simulate data parsing for testing

1. Log on to the IoT Platform console.
2. From the left-side navigation pane, choose Devices > Product.
3. Click Create Product to create a product and set the data type to Do not parse/Custom. For more information, see [Create a product](#).
4. On the Product Details page, click the Data Parsing tab. Edit your data parsing script in the editor. Only JavaScript is supported. For more information, see [Example: Edit a script](#).

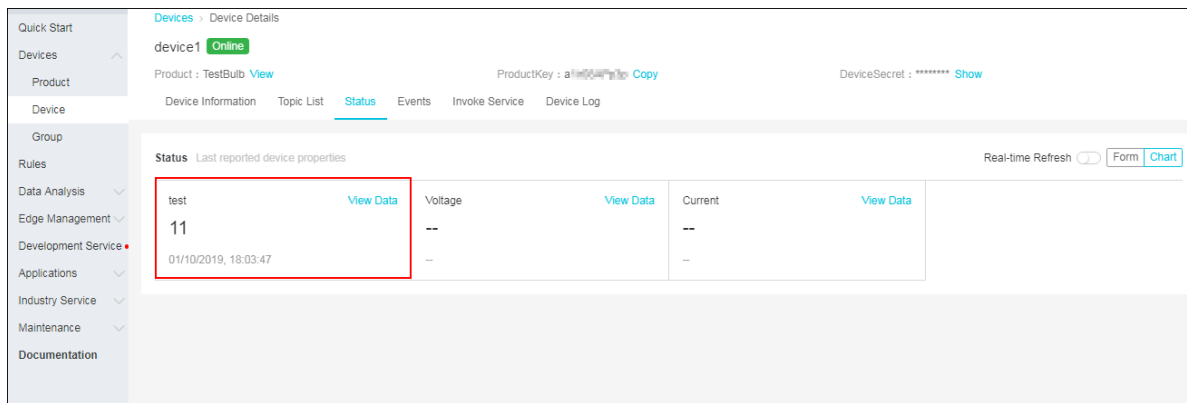


When you edit the script, you can perform the following operations:

- Click Full Screen to view or edit the script in full screen. Click Exit Full Screen to exit the full screen.
 - Click Save Draft at the bottom of the page to save the content that you have edited. The next time you access the Data Parsing page, you will be notified that you have a draft. You can then choose to restore edit or delete draft.
 - A saved draft script will not be published to the running platform and will not affect a published script.
 - A new draft will overwrite any previously saved draft.
5. After you finish editing the script, you can enter analog data in the Analog Input box. Click Run to test whether the script can be used to parse data correctly. For

more information about analog data and parsing results, see [Verify a data parsing script](#).

6. If you confirm that the script is correct and can parse data correctly, click Submit to submit the script to the running platform. When data is exchanged between IoT Platform and the device, the system will automatically call the corresponding function in the script to convert data.
7. Perform a test by sending data to IoT Platform from a real device.
 - a. Register a device, and develop the [device SDK](#)
 - b. The device connects to IoT Platform and reports data to IoT Platform.
 - c. In the IoT Platform console, go to the Device Details page of the device. Click the Status tab to view the device property data.



Example: Edit a script

The following describes the data parsing script format and content of a product. In this example, the device data is in hexadecimal notation, and the product has three properties: prop_float, prop_int16, and prop_bool.

1. Create a product and select Do not parse/Custom as the data type. Then, define the following properties. For more information, see [Define features](#).

Identifier	Type	Value range	Read/write
prop_float	float	-100 to 100	Read/write
prop_int16	int32	-100 to 100	Read/write
prop_bool	bool	0: Enabled. 1: Disabled.	Read/write

2. Define the communication protocol as follows:

Table 1-1: Upstream data request

Field	Number of bytes
Frame type	One
Request ID	Four
prop_int16	Two
prop_bool	One
prop_float	Four

Table 1-2: Upstream data response

Field	Number of bytes
Frame type	One
Request ID	Four
Result code	One

Table 1-3: Property setting request

Field	Number of bytes
Frame type	One
Request ID	Four
prop_int16	Two
prop_bool	One
prop_float	Four

Table 1-4: Property setting response

Field	Number of bytes
Frame type	One
Request ID	Four
Result code	One

3. Edit the script.

You must define the following methods in the script:

- `protocolTo RawData` : Convert Alink JSON formatted data to custom formatted data.
- `rawDataToP rotocol` : Convert custom formatted data to Alink JSON formatted data.

A script demo is as follows:

```
var  COMMAND_RE  PORT  = 0x00 ; // Devices  report  property
var  COMMAND_SE  T    = 0x01 ; // Set    property
var  COMMAND_RE  PORT_REPLY  = 0x02 ; // Respond  to  the
reported  data
var  COMMAND_SE  T_REPLY  = 0x03 ; // Respond  to  the
property  setting  request
var  COMMAD_UNK  OWN  = 0xff ;    // Other  command
var  ALINK_PROP  _REPORT_ME  THOD  = ' thing . event . property
.post ' ; // This  is  a  topic  for  devices  to  report
property  data  to  IoT  Platform .
var  ALINK_PROP  _SET_METHO  D  = ' thing . service . property .
set ' ; // This  is  a  topic  for  IoT  Platform  to
send  property  management  commands  to  devices .
var  ALINK_PROP  _SET_REPLY  _METHOD  = ' thing . service .
property . set ' ; // This  is  a  topic  for  devices  to
report  property  setting  results  to  IoT  Platform .
/*
Sample  data :
Upstream  data
Input ->
    0x00000000  0100320100  000000
Output ->
    {" method ":" thing . event . property . post "," id ":" 1 ","
params ":{\" prop_float \": 0 ,\" prop_int16 \": 50 ,\" prop_bool \": 1
},\" version \": \" 1 . 0 \"}

Property  setting  response
Input ->
    0x03002233  44c8
Output ->
    {" code ":" 200 "," data ":{} ,\" id ":" 2241348 "," version \": \" 1
. 0 \"}
*/
function  rawDataToP  rotocol ( bytes ) {
    var  uint8Array  = new  Uint8Array ( bytes . length );
    for ( var  i  = 0 ; i < bytes . length ; i ++ ) {
        uint8Array [ i ] = bytes [ i ] & 0xff ;
    }
    var  dataView  = new  DataView ( uint8Array . buffer , 0 );
    var  jsonMap  = new  Object ();
    var  fHead  = uint8Array [ 0 ]; // command
    if ( fHead == COMMAND_RE  PORT ) {
        jsonMap [ ' method ' ] = ALINK_PROP  _REPORT_ME  THOD ; //
The  Alink  JSON  formatted  data  topic  for  reporting
properties
        jsonMap [ ' version ' ] = ' 1 . 0 ' ; // The  fixed
protocol  version  field  in  the  Alink  JSON  format
```

```

        jsonMap [' id ' ] = ' ' + dataView . getInt32 ( 1 ); // The
request ID in Alink JSON format
        var params = {};
        params [' prop_int16 ' ] = dataView . getInt16 ( 5 ); //
The value of prop_int16
        params [' prop_bool ' ] = uint8Array [ 7 ]; // The value
of prop_bool
        params [' prop_float ' ] = dataView . getFloat32 ( 8 ); //
The value of prop_float
        jsonMap [' params ' ] = params ; // The value for
params in Alink JSON format
    } else if ( fHead == COMMAND_SE T_REPLY ) {
        jsonMap [' version ' ] = ' 1 . 0 ' ; // The fixed
protocol version field in the Alink JSON format
        jsonMap [' id ' ] = ' ' + dataView . getInt32 ( 1 ); // The
request ID value in Alink JSON format
        jsonMap [' code ' ] = ' ' + dataView . getUint8 ( 5 );
        jsonMap [' data ' ] = {};
    }

    return jsonMap ;
}
/*
Sample data :
Property setting
Input ->
{" method ":" thing . service . property . set "," id ":" 12345
"," version ":" 1 . 0 "," params :{" prop_float " : 123 . 452 , "
prop_int16 " : 333 , " prop_bool " : 1 }}
Output ->
0x01000030 39014d0142 f6e76d

Upstream data response
Input ->
{" method ":" thing . event . property . post "," id ":" 12345
"," version ":" 1 . 0 "," code " : 200 , " data " :{}}
Output ->
0x02000030 39c8
*/
function protocolTo RawData ( json ) {
    var method = json [' method ' ];
    var id = json [' id ' ];
    var version = json [' version ' ];
    var payloadArr ay = [];
    if ( method == ALINK_PROP _SET_METHO D ) // Set
properties
    {
        var params = json [' params ' ];
        var prop_float = params [' prop_float ' ];
        var prop_int16 = params [' prop_int16 ' ];
        var prop_bool = params [' prop_bool ' ];
        // Join raw data according to the custom
protocol format
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( COMMAND_SE T )); // The command field
        payloadArr ay = payloadArr ay . concat ( buffer_int 32
( parseInt ( id )); // The ID in Alink JSON format
        payloadArr ay = payloadArr ay . concat ( buffer_int 16
( prop_int16 )); // The value of prop_int16
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( prop_bool )); // The value of prop_bool
        payloadArr ay = payloadArr ay . concat ( buffer_flo
at32 ( prop_float )); // The value of prop_float
    }
}

```

```

    } else if ( method == ALINK_PROP_REPORT_ME_THOD ) { //
Response to device upstream data
        var id = json [' id '];
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( COMMAND_SE T )); // The command field
        payloadArr ay = payloadArr ay . concat ( buffer_int 32
( parseInt ( id ))); // The ID in Alink JSON format
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( code ));
    } else { // Other commands that will not be
processed
        var id = json [' id '];
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( COMMAND_SE T )); // The command field
        payloadArr ay = payloadArr ay . concat ( buffer_int 32
( parseInt ( id ))); // The ID in Alink JSON format
        payloadArr ay = payloadArr ay . concat ( buffer_uin t8
( code ));
    }
    return payloadArr ay ;
}
// The following lists some auxiliary functions :
function buffer_uin t8 ( value ) {
    var uint8Array = new Uint8Array ( 1 );
    var dv = new DataView ( uint8Array . buffer , 0 );
    dv . setUint8 ( 0 , value );
    return []. slice . call ( uint8Array );
}
function buffer_int 16 ( value ) {
    var uint8Array = new Uint8Array ( 2 );
    var dv = new DataView ( uint8Array . buffer , 0 );
    dv . setInt16 ( 0 , value );
    return []. slice . call ( uint8Array );
}
function buffer_int 32 ( value ) {
    var uint8Array = new Uint8Array ( 4 );
    var dv = new DataView ( uint8Array . buffer , 0 );
    dv . setInt32 ( 0 , value );
    return []. slice . call ( uint8Array );
}
function buffer_flo at32 ( value ) {
    var uint8Array = new Uint8Array ( 4 );
    var dv = new DataView ( uint8Array . buffer , 0 );
    dv . setFloat32 ( 0 , value );
    return []. slice . call ( uint8Array );
}

```

Verify a data parsing script

After you edit a sample script, you can verify the correctness of the script. Enter analog data in the Analog Input box, and click Run. The system will call this script to parse the analog data. The parsed result will be displayed in the Parsing Results box at the right side of the page.

- Parse the device-reported property data

Select **Upstreamed Device Data** as the simulation type, enter the following hexadecimal data, and then click **Run**.

```
0x00002233 441232013f a00000
```

The data parsing engine will convert the hexadecimal data to JSON data as defined in the script. The result will be displayed in the **Parsing Results** area.

```
{
  " method ": " thing . event . property . post ",
  " id ": " 2241348 ",
  " params ": {
    " prop_float ": 1 . 25 ,
    " prop_int16 ": 4658 ,
    " prop_bool ": 1
  },
  " version ": " 1 . 0 ",
}
```

- Parse downstream data from IoT Platform to the device.

Select **Received Device Data** as the simulation type, enter the following JSON data, and then click **Run**.

```
{
  " id ": " 12345 ",
  " version ": " 1 . 0 ",
  " code ": 200 ,
  " method ": " thing . event . property . post ",
  " data ": {}
}
```

The data parsing engine will convert the JSON data to the following hexadecimal data.

```
0x02000030 39c8
```

- Parse the property setting data from IoT Platform to devices.

Select **Received Device Data** as the simulation type, enter the following JSON data, and then click **Run**.

```
{
  " method ": " thing . service . property . set ",
  " id ": " 12345 ",
  " version ": " 1 . 0 ",
  " params ": {
    " prop_float ": 123 . 452 ,
    " prop_int16 ": 333 ,
    " prop_bool ": 1
  }
}
```

```
}
```

The data parsing engine converts JSON data to the following hexadecimal data.

```
0x01000030 39014d0142 f6e76d
```

- Parse property setting results returned by the device.

Select Upstreamed Device Data as the simulation type, enter the following hexadecimal data, and then click Run.

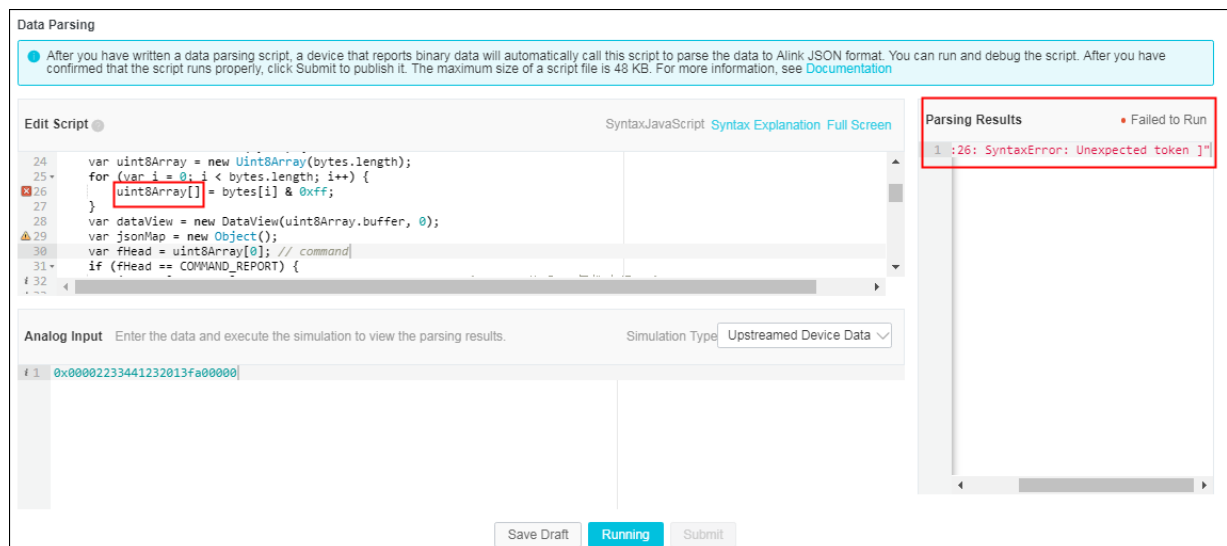
```
0x03002233 44c8
```

The data parsing engine will convert the hexadecimal data to the following JSON data.

```
{
  "code": " 200 ",
  "data": {},
  "id": " 2241348 ",
  "version": " 1 . 0 ",
}
```

If the script is incorrect, an error message is displayed in the Parsing Results area.

You must troubleshoot the error according to the error message and modify the script code accordingly.



Debug a data parsing script in a local computer

IoT Platform Data Parsing does not support debugging on the running platform. We recommend that you develop and debug the script locally and then paste the finished script into the online editor. You may use the following debugging method.

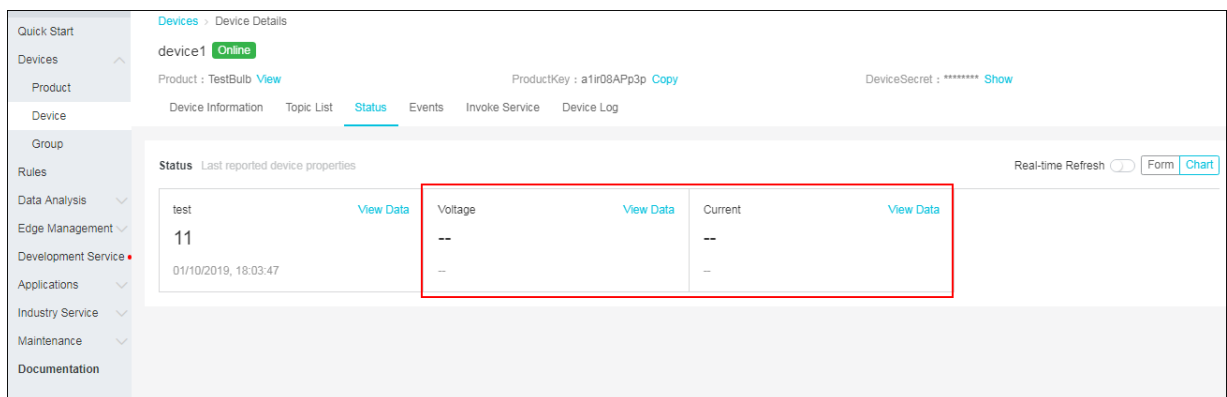
```
// Test Demo
```

```
function Test ()
{
    // 0x00123201 3fa00000
    var rawdata_re port_prop = new Buffer ([
        0x00 , // The fixed command header . A value of
0 indicates a property report message .
        0x00 , 0x22 , 0x33 , 0x44 , // The ID fields that
identify the request sequence .
        0x12 , 0x32 , // Two - byte value of prop_int16
        0x01 , // One - byte value of prop_bool
        0x3f , 0xa0 , 0x00 , 0x00 // Four - byte value of
prop_float
    ]);
    rawDataToP rotocol ( rawdata_re port_prop );
    var setString = new String ('{" method ":" thing . service
. property . set "," id ":" 12345 "," version ":" 1 . 0 "," params
":{" prop_float ": 123 . 452 , " prop_int16 ": 333 , " prop_bool ": 1
}}');
    protocolTo RawData ( JSON . parse ( setString ));
}
Test ();
```

Troubleshoot issues

After a device is connected to IoT Platform and reports data, the reported data can be displayed in the IoT console if data parsing functions correctly. To view the data, go to the Device Details page of the device and click the Status tab.

In some occasions, after the device reports data, no data is displayed on the page, as shown in the following figure:



To view device logs: From the left-side navigation pane, choose Maintenance > Device Log and select the corresponding product. On the Device Log page, click the TSL Data Analysis tab. You can view the communication log between the device and IoT Platform.

Use the following process to troubleshoot the issue:

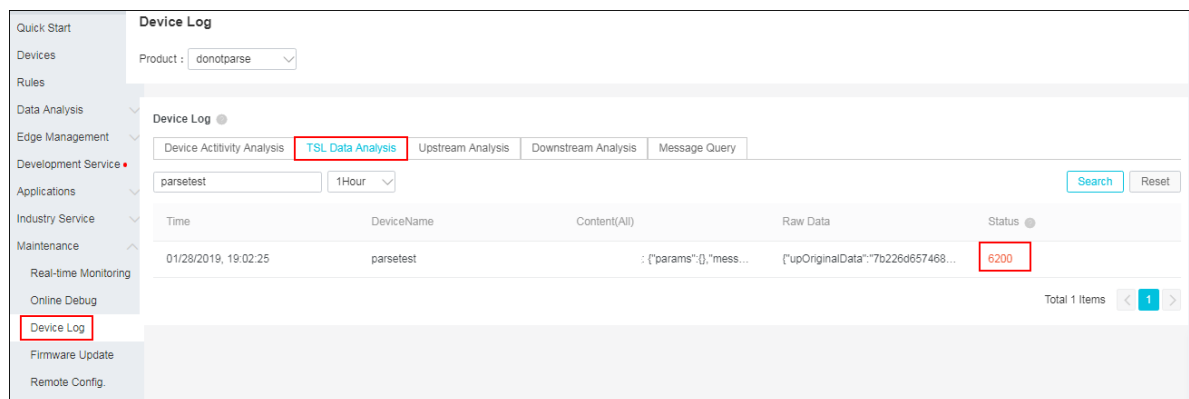
1. View the reported data on the Device Log page. Each log entry records the converted data and the original data.

2. Check the error codes according to the descriptions in [Device log](#).
3. Troubleshoot the issue based on the error code, the script, and the reported data.

The following lists some errors:

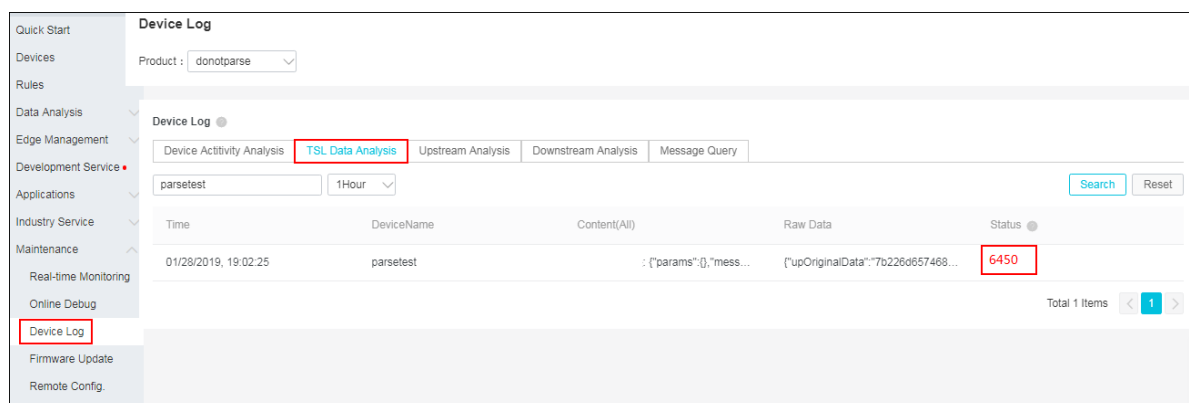
- The data parsing script is not found.

As shown in the following figure, the error code is 6200. To check the description of the error, see [Device log](#). The error code of 6200 indicates that no script was found. Check whether the data parsing script has been submitted in the console.



- Alink method does not exist.

The error code is 6450. This error code is described in [Device log](#) as follows: The method parameter is not found in Alink data. This error occurs if the method parameter is not found in the Alink data reported by the device or in the parsed result of Do not parse/Custom data.



You can check the raw data, for example:

```
17 : 54 : 19 . 064 , A7B02C6064 6B4D2E8744 F7AA7C3D95 67 ,
upstream - error - bizType = OTHER_MESS AGE , params ={" params
```

```
":{}}, result = code : 6450 , message : alink method not exist ,...
```

In the log, the error message is `alink method not exist` . If this error occurs, you must correct your script.

1.5 Topics

The cloud and devices communicate with each other in IoT Platform through topics. The device reports messages to a specified topic and subscribes to messages from the topic. IoT Platform sends commands to topics, and subscribes to specific topics to obtain device information.

1.5.1 What is a topic?

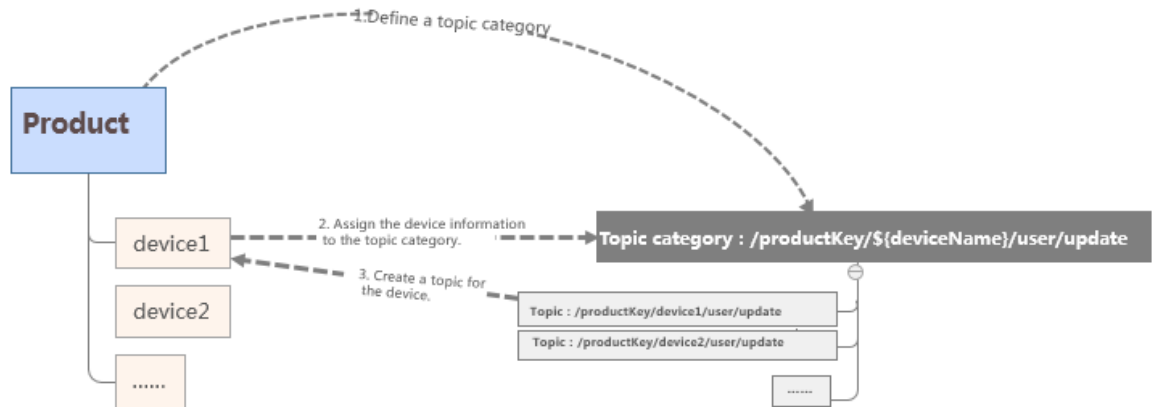
A server and a device communicate with each other in IoT Platform through topics. Topics are associated with devices, and topic categories are associated with products. A topic category of a product is automatically applied to all devices under the product to generate device-specific topics for message communication.

Topic category

To simplify authorization and facilitate communication between devices and IoT Platform, topic categories were introduced. A topic category is a set of topics within the same product. For example, topic category `/${ YourProduc tKey }/ ${ YourDevice Name }/ user / update` is a set that contains the following two topics: `/${ YourProduc tKey }/ device1 / user / update` and `/${ YourProduc tKey }/ device2 / user / update` .

After a device is created, all topic categories of the product are automatically applied to the device. You do not need to assign topics to each individual device.

Figure 1-1: Automatically create a topic



Descriptions for topic categories:

- A topic category uses a forward slash (/) to separate elements in different hierarchical levels. A topic category contains the following fixed elements: `${YourProductKey}` indicates the product identifier; `${YourDeviceName}` indicates the device name.
- Each element name can contain only letters, numbers, and underscores (_). An element in each level cannot be left empty.
- A device can have Pub and Sub permissions to a topic. Pub indicates that the device can publish messages to the topic. Sub indicates that the device can subscribe to the topic.

Topic

A topic category is used for topic definition rather than communication. Only topics can be used for communication.

- Topics use the same format as topic categories. The difference is that variable `${YourDeviceName}` in the topic category is replaced by a specific device name in the topic.
- A topic is automatically derived from the topic category of the product based on the corresponding device name. A topic contains the device name (`DeviceName`) and can be used for data communication only by the specified device. For example, topic `/${YourProductKey}/device1/user/update` belongs to the

device named device1. Only device1 can publish messages and subscribe to this topic. Other devices cannot use this topic.

Supported wildcards

To use the rules engine data forwarding function to forward device data, you must specify the source topic of the messages when writing an SQL statement. When you specify a topic in [setting a forwarding rule](#), you can use the following wildcards. One element can contain only one wildcard.

Wildcard	Description
#	Must be set as the last element in the topic. This wildcard can match any element in the current level and sub-levels. For example, in topic <code>/\${ YourProduc tKey }/ device1 / user /#</code> , wildcard <code>#</code> is added next to the <code>/ user</code> element to represent all elements after <code>/ user</code> . This topic can represent <code>/\${ YourProduc tKey }/ device1 / user / update</code> and <code>/\${ YourProduc tKey }/ device1 / user / update / error</code> .
+	Matches all elements in the current level. For example, in topic <code>/\${ YourProduc tKey }/+/ user / update</code> , the device name element is replaced by wildcard <code>+</code> to represent all devices under the product. This topic can represent <code>/\${ YourProduc tKey }/ device1 / user / update</code> and <code>/\${ YourProduc tKey }/ device2 / user / update</code> .

System topics and custom topics

IoT Platform supports the following types of topics:

Type	Description
System topics	The system-defined topics. System topics cannot be modified and deleted. System topics include topics used by IoT Platform functions, such as TSL model-related functions and firmware upgrade. For example, topics related to TSL models generally start with <code>/sys/</code>. Topics related to firmware upgrade start with <code>/ota/</code>. Topics for the device shadow function start with <code>/shadow/</code>.  Note: System topics are not completely displayed in the Topic Categories list and the Topic List. For more information about function-specific topics, see related function documentation.
Custom topics	You can customize a topic category on the Topic Categories tab page according to your business requirements. The topic categories you have customized for the product will be automatically applied to all devices under the product.

1.5.2 Create a topic category

This article introduces how to create a topic category for a product. Topic categories will be automatically assigned to devices of the product.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Product
3. On the Products page, find the product for which you want to create a topic category, and click View in the operation column.
4. On the Product Details page, click Topic Categories > Create Topic Category.

5. Define a topic category.

Create Topic Category

Use slashes (/) to delimit the category hierarchy. The first category is ProductKey. The second category is deviceName. The third category is used to identify custom topics in Pro Editions. For example, the /pk/\${deviceName}/update topic category includes topics /pk/mydevice/user/update and /pk/yourdevice/user/update

* Device Operation Authorizations:

Publish

* Topic Category:

/a107b8j9f0e/\${deviceName}/user/

Description :

0/100

OK

Cancel

- **Device Operation Authorizations:** Indicates the operations that devices can perform on the topics of this topic category. You can select from Publish, Subscribe, and Publish and Subscribe.
- **Topic Category:** Enter a custom topic category name according to the **Topic Rule** on the page.
- **Description:** Describes the topic category. You can leave this box empty.

6. Click OK.

Wildcard characters in topic categories

When you create topic categories, you can use wildcards. For more information about wildcards, see [What is a topic?](#) Supported wildcards:

- **#:** Includes the category level you enter and all lower levels in topics.

- **+**: Includes only one category level in topics, and not lower levels.

**Note:**

When you want to create topic categories with wildcards, note that:

- Only topics with Device Operation Authorizations as Subscription support wildcards.
- # can only be at the end of topics.
- For topics with wildcard characters, you cannot click Publish to publish messages on the Topic List tab page of devices.

1.6 Tags

A tag is a custom identifier you set for a product, a device, or a device group. You can use tags to flexibly manage your products, devices and groups.

IoT often involves the management of a huge number of products and devices.

How to distinguish various products and devices, and how to achieve centralized management become a challenge. Alibaba Cloud IoT Platform allows you to use tags to address these issues. The use of tags allows the centralized management of your various products, devices, and groups.

Therefore, we recommend that you create tags for your products, devices and device groups. The structure of a tag is `key : value`.

This article describes how to create product tags, device tags, and group tags in the console.

**Note:**

Each product, device, or group can have up to 100 tags.

Product tags

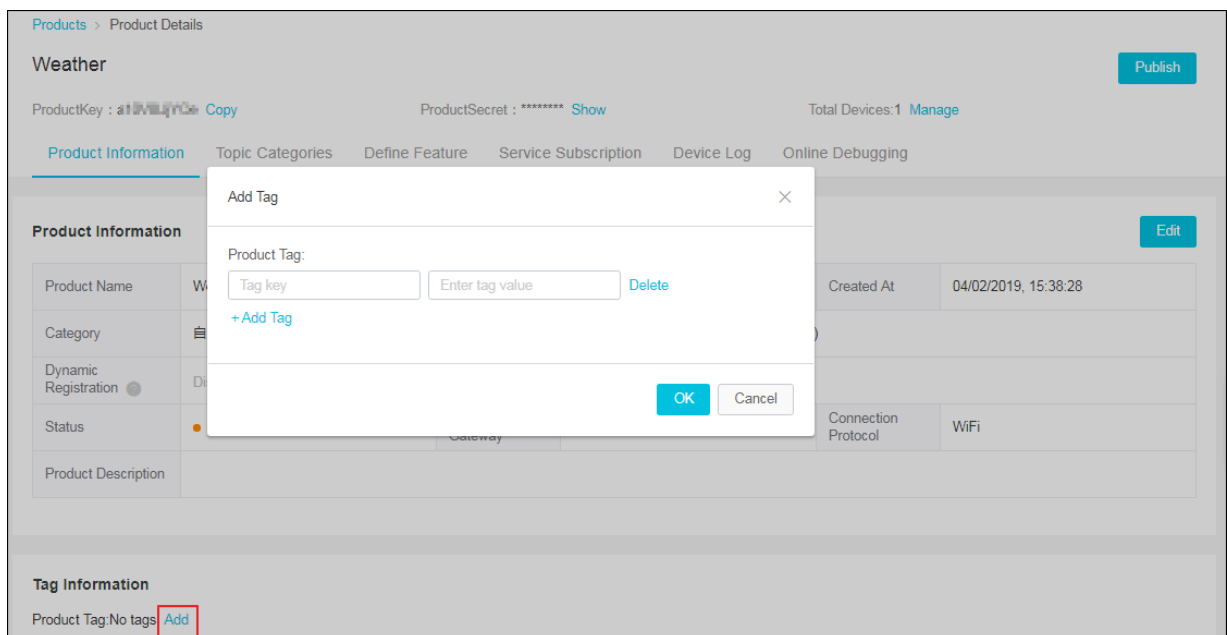
Product tags typically describe the information that is common to all devices of a product. For example, a tag can indicate a specific manufacturer, organization, physical size, or operating system. After a product has been created, you can create tags for it.

To create product tags in the console, follow these steps:

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

2. In the left-side navigation pane, click **Devices > Product**.
3. On the **Products** page, find the product for which you want to create tags and click **View**.
4. Click **Add** under **Tag Information**.
5. In the dialog box, enter values for **Tag Key** and **Tag Value**, and then click **OK**.

Parameter	Description
Tag Key	A tag key can contain English letters, digits and dots (.), and cannot exceed 30 characters.
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as two characters.



Device tags

You can facilitate device management by creating unique tags for devices. For example, you can use the device feature information as tags, such as `PowerMeter : room201` for the electricity meter of room 201.

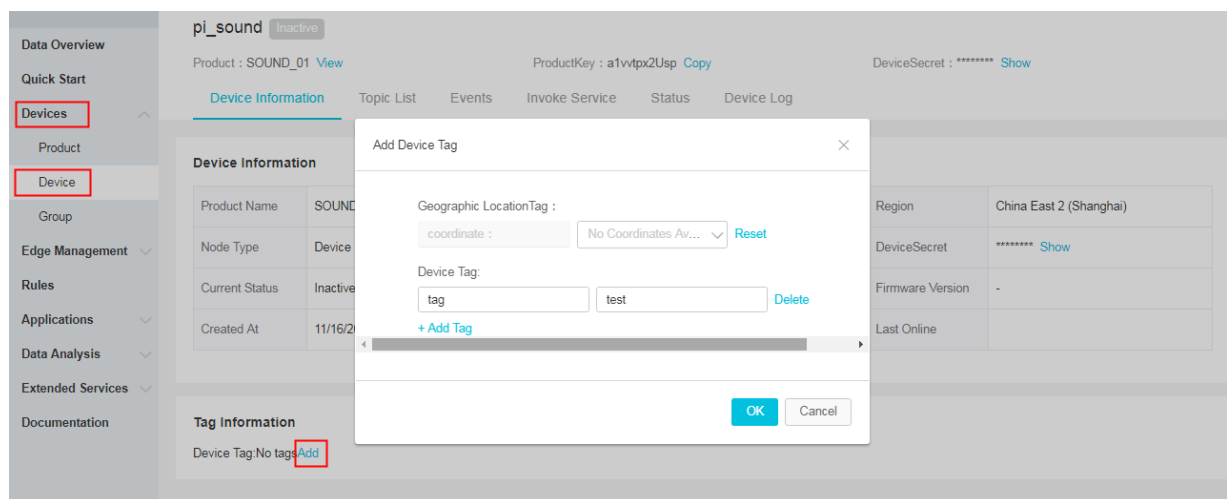
Device tags always follow the devices. You can include tag information in the messages reported to IoT Platform by devices. When you use the rules engine to

forward these messages to other Alibaba Cloud services, the tag information is also forwarded to the targets.

To create device tags in the console, follow these steps:

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click **Devices > Device**.
3. On the Devices page, find the device for which you want to create tags, click **View** to go to the Device Details page.
4. Click **Add** under Tag Information.
5. In the dialog box, enter values for **Tag Key** and **Tag Value**, and then click **OK**.

Parameter	Description
Tag Key	A tag key can contain English letters, digits, and dots (.), and can be 2-30 characters in length.
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as 2 characters.



Group tags

You can manage devices across products by grouping your devices. A group tag typically describe the general information of devices in the group and the sub-groups

. For example, you can use region information as a group tag. After you have created a group, you can create tags for it.

To create group tags, follow these steps:

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click **Devices > Group**.
3. On the Group Management page, find the group for which you want to create tags and click **View**.
4. Click **Add** under Tag Information.
5. In the dialog box, enter values for **Tag Key** and **Tag Value**, and then click **OK**.

Parameter	Description
Tag Key	A tag key can contain English letters, digits, and dots (.), and can be 2-30 characters in length.
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as 2 characters.

The screenshot displays the IoT Platform console interface. On the left, the navigation pane shows 'IoT Platform' at the top, followed by 'Data Overview', 'Quick Start', 'Devices' (highlighted with a red box), 'Product', 'Device', 'Group' (highlighted with a red box), 'Edge Management', 'Rules', 'Applications', 'Data Analysis', 'Extended Services', and 'Documentation'. The main area is titled 'Group Management > Group Details' and shows the group 'test11'. It includes fields for 'Group Level: Group/test11', 'Group ID: Z0EIGF5aqc0thBtW' (with a 'Copy' link), 'Total Devices: 1', 'Activate Devices: 1', and 'Online Devices: 1'. Below this, there are tabs for 'Group Information', 'Device List', and 'Subgroups'. The 'Group Information' tab is active, showing a table with group details. At the bottom, the 'Tag Information' section shows 'Group Tag: No tags.' and an 'Add' button (highlighted with a red box).

Group Information					
Group Name	test11	Group Level	Group/test11	Group ID	Z0EIGF5aqc0thBtW Copy
Total Devices	1	Activate Device	1	Online	1
Created At	10/24/2018, 17:47:57				
Group Description	betested				

Tag Information

Group Tag: No tags. [Add](#)

Manage tags in batch

In the console, you only can create, modify, and delete tags one by one. IoT Platform provides APIs for managing tags in batch. In addition, IoT Platform provides APIs for

querying products, devices, and groups based on tags. For more information about tag related APIs, see the documents in API reference.

1.7 Gateways and sub-devices

1.7.1 Gateways and sub-devices

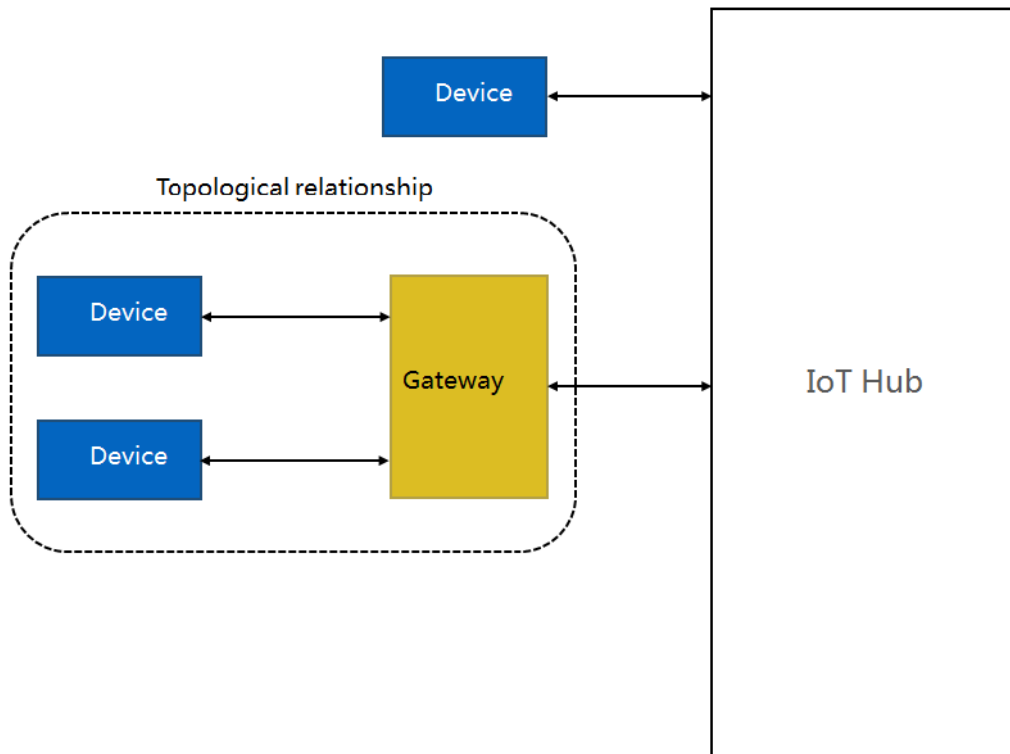
IoT Platform allows devices to connect to it directly, or be mounted as sub-devices to gateways that connect to IoT Platform.

Gateways and devices

When you create a product, you must select a node type for the devices of the product. Currently, IoT Platform supports two node types, `Device` and `Gateway`.

- **Device:** Devices of this node type cannot be mounted with sub-devices, but can be connected directly to the IoT Platform or be mounted as sub-devices to gateways.
- **Gateway:** Devices of this node type can connect to IoT Platform directly and can be mounted with sub-devices. Gateways are then used to manage sub-devices, maintain topological relationships with sub-devices, and synchronize these topological relationships to IoT Platform.

The topological relationship between a gateway and its sub-devices is shown in the following figure:



Connect gateways and sub-devices to IoT Platform

Once a gateway has been connected to IoT Platform, the gateway will synchronize its topological relationships with its sub-devices to IoT Platform. A gateway supports device authentication, message reporting, instruction receiving, and other communications with IoT Platform for all its sub-devices. That is, sub-devices are managed by their corresponding gateway.

1. For more information about how to connect gateways to IoT Platform, see [Link Kit SDK](#).
2. You can connect sub-devices to IoT Platform using either of the following two methods:
 - The [Unique-certificate-per-device authentication](#) method. This method requires you to install the device certificates (namely, the ProductKey, DeviceName, and DeviceSecret) in the physical sub-devices, and then connect the sub-devices to IoT Platform.
 - The [Unique-certificate-per-product authentication](#) method. This method requires you to enable Dynamic Registration on the product details page and register devices in the IoT Platform console. Then, when a physical sub-device is being connected, the gateway will initiate a connection request to IoT Platform for the sub-device. IoT Platform then verifies the sub-device information. If

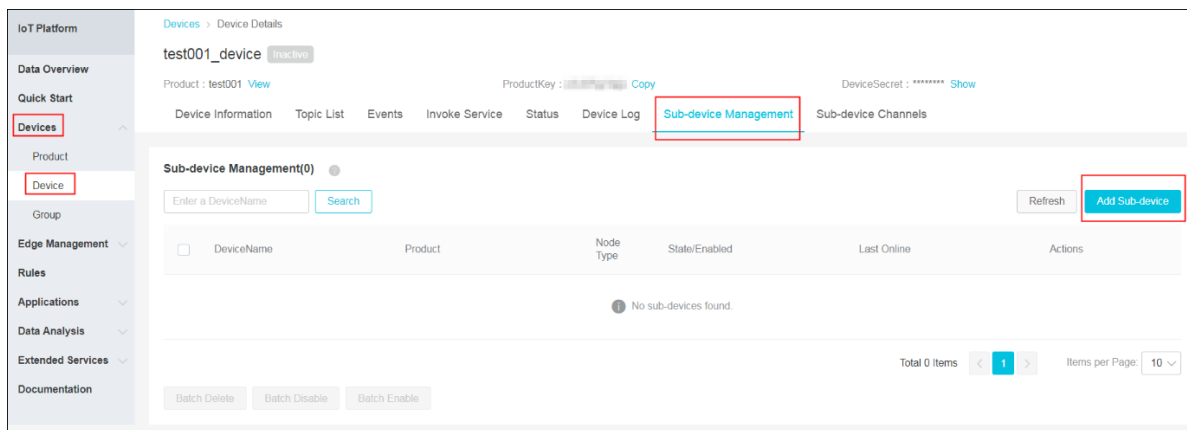
the verification passes, IoT Platform will assign the DeviceSecret to the sub-device. The sub-device then receives all the required information (namely, the ProductKey, DeviceName, and DeviceSecret) to successfully connect to IoT Platform.

1.7.2 Sub-device management

You can add sub-devices to a gateway device, and send the TSL and the extended service information of the sub-devices to the gateway.

Procedure

1. In the left-side navigation pane, click **Devices > Device**.
2. On the Devices page, find the gateway device for which you want to add sub-devices and click **View** corresponding to it. You are directed to the Device Details page.
3. Click **Sub-device Management > Add Sub-device**.



4. Enter the information of the sub-device in the dialog box.

Parameter	Description
Product	Select the name of the product for which the sub-device belongs.
Device	Select the name of the device that you want to add as a sub-device.

What's next

The topological relationship between the gateway and the sub-device has been built. On the details page of the sub-device, you can view the gateway device information.

1.8 Service Subscription

1.8.1 What is service subscription?

A server can directly subscribe to messages under a product: device upstream notifications, device status change notifications, notifications of sub-devices reported by gateway devices, device lifecycle change notifications, and topological relationship change notifications. After you configure the Service Subscription function, IoT Platform forwards the subscribed messages from all devices under the product to your server. Two subscription methods are supported. One is to forward data through HTTP/2 channels to your servers and the other is to push message to your Message Service instances.

Scenarios

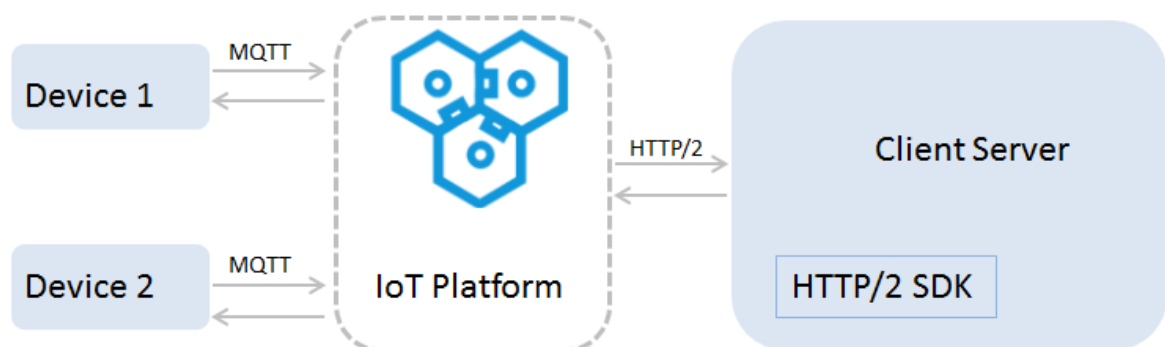
Service Subscription is applicable to scenarios where only data receiving is involved. The following conditions must also be met:

- The server must receive subscribed data from all devices under the product.
- Device data is transmitted at a rate of up to 5,000 messages per second.

HTTP/2-based message subscription

The new version of IoT Platform can push messages over HTTP/2 channels. After you configure HTTP/2-based message subscription for a product, IoT Platform will push the subscribed messages of all devices under the product to your server through the HTTP/2 channel.

Data forwarding workflow for HTTP/2-based subscription:



The server can receive messages directly from IoT Platform by connecting the HTTP/2 SDK to IoT Platform. The HTTP/2 SDK provides identity authentication, topic subscription, and message sending and message receiving capabilities.

- The HTTP/2 SDK on the server is used to transfer a large number of messages between IoT Platform and the server.
- The HTTP/2 SDK on the device is used to transfer messages between devices and IoT Platform.



Note:

Only Java and .NET SDKs are supported.

For information about how to configure HTTP/2 channels and configuration examples, see:

- [Limits](#)
- [Development guide for Java HTTP/2 SDK](#)
- [Development guide for .NET HTTP/2 SDK](#)

For information about comparisons between service subscription-based and rules engine-based data forwarding, see [Compare data forwarding solutions](#).

Push messages to Message Service

IoT Platform pushes subscribed messages to Message Service. Your server applications listen to queues in Message Service to receive device messages.

For more information about how to use Message Service to subscribe to device messages, see [Use Message Service to subscribe to device messages](#).



Note:

Message Service charges fees for receiving messages pushed by IoT Platform. For more information about the billing and usage of Message Service, see [Message Service documentation](#).

1.8.2 Development guide for Java HTTP/2 SDK

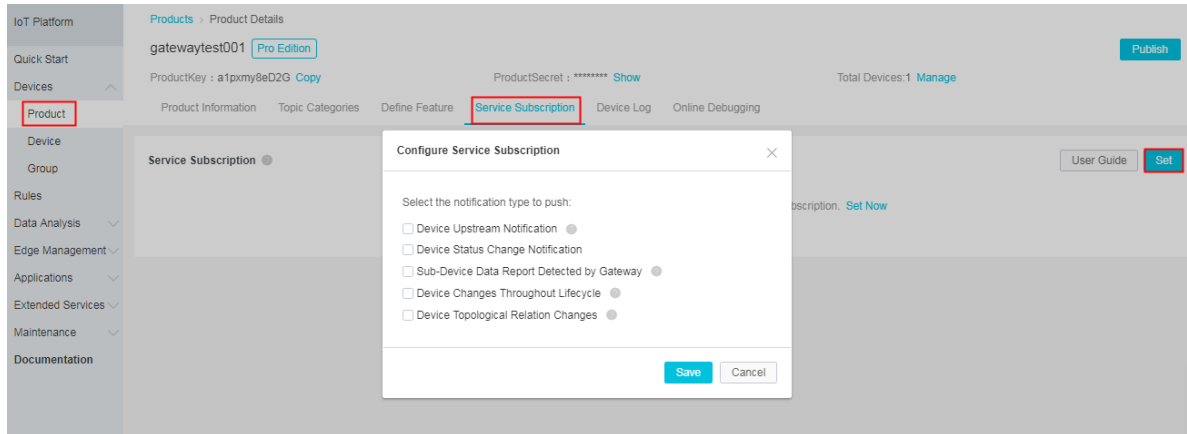
This article introduces how to configure the service subscription, connect to the HTTP/2 SDK, authenticate identity, and configure the message-receiving interface.

Specifically, this section details the development process of the service subscription. Download the [server side Java HTTP/2 SDK demo](#).

Configure service subscription

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, click Devices > Product.

3. In the product list, find the product for which you want to configure the service subscription and click View. You are directed to the Product Details page.
4. Click Service Subscription > Set Now.
5. Select the types of notifications that you want to push to the SDK.



- **Device Upstream Notification:** Indicates the messages of the topics to which devices are allowed to publish messages. If this notification type is selected, the HTTP/2 SDK can receive messages reported by devices.

Devices report custom data and TSL data of properties, events, responses to property setting requests, and responses to service calling requests.

For example, a product has three topic categories:

- `/ ${YourProductKey} / ${YourDeviceName} / user / get` , devices can subscribe to messages.
- `/ ${YourProductKey} / ${YourDeviceName} / user / update` , devices can publish messages.
- `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` , devices can publish messages.

Service Subscription can push messages of the topics `/ ${YourProductKey} / ${YourDeviceName} / user / update` and `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` , to which devices can then publish messages. Additionally, the messages of `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` are processed by the system before being pushed.

- **Device Status Change Notification:** Indicates the notifications that are sent when the statuses of devices change, for example, notifications for when devices

go online or go offline. The topic `/as/mqtt/status/${YourProductKey}/${YourDeviceName}` has device status change messages. After this notification type is selected, the HTTP/2 SDK can receive the device status change notifications.

- *Sub-Device Data Report Detected by Gateway*: Gateways can report the information of sub-devices that are discovered locally. To use this feature, make sure that the applications on the gateway support this feature.
- *Device Topological Relation Changes*: Includes notifications about creation and removal of the topological relation between a gateway and its sub-devices.
- *Device Changes Throughout Lifecycle*: Includes notifications about device creation, deletion, disabling, and enabling.



Note:

For messages of device properties, events, and services, Device Status Change Notification, Sub-Device Data Report Detected by Gateway, Device Topological Relation Changes, and Device Changes Throughout Lifecycle, the QoS is 0 by default. For other Device Upstream Notification messages (except messages of device properties, events, and services), you can set the QoS is 0 or 1 on your device SDK.

Connect to the SDK

Add the maven dependency to the project to connect to the SDK.

```
< dependency >
  < groupId > com . aliyun . openservice </ groupId >
  < artifactId > iot - client - message </ artifactId >
  < version > 1 . 1 . 3 </ version >
</ dependency >

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

Identity authentication

Use the AccessKey information of your account for identity authentication and to build the connection between the SDK and IoT Platform.

Example:

```
// Your account accessKeyId
String accessKey = "xxxxxxxxxx xxxxx ";
// Your account AccessKeySecret
```

```

String accessSecret = "xxxxxxxxxx xxxxx ";
// regionId
String regionId = "cn-shanghai";
// Your account ID
String uid = "xxxxxxxxxx xx";
// endPoint: https://{uid}.iot-as-http2.{region}.aliyuncs.com
String endPoint = "https://" + uid + ".iot-as-http2." + regionId + ".aliyuncs.com";

// Connection configuration
Profile profile = Profile.getAccessKeyProfile (
endPoint, regionId, accessKey, accessSecret);

// Construct the client
MessageClient client = MessageClientFactory.
messageClient (profile);

// Receive data
client.connect (messageToken -> {
    Message m = messageToken.getMessage();
    System.out.println ("receive message from " +
m);
    return MessageCallback.Action.CommitSuccess;
});

```

The value of `accessKey` is the AccessKeyID of your account, and the value of `accessSecret` is the AccessKeySecret corresponding to the AccessKeyID. Log on to the [Alibaba Cloud console](#), hover the mouse over your account image, and click **AccessKey** to view your AccessKeyID and AccessKeySecret. You can also click **Security Settings** to view your account ID.

The value of `regionId` is the region ID of your IoT Platform service.

Configure the message receiving interface

Once the connection is established, the server immediately pushes the subscribed messages to the SDK. Therefore, when you are configuring the connection, you also configure the message-receiving interface, which is used to receive the messages for which callback has not been configured. We recommend that you call `setMessageListener` to configure a callback before you `connect` the SDK to IoT Platform.

Use the `consume` method of `MessageCallback` interface and call the `setMessageListener()` of `messageClient` to configure the message receiving interface.

The returned result of `consume` determines whether the SDK sends an ACK.

The method for configuring the message receiving interface is as follows:

```

MessageCallback messageCallback = new MessageCallback ()
{

```

```

    @ Override
    public Action consume ( MessageToken messageToken ) {
        Message m = messageToken . getMessage ();
        log . info (" receive : " + new String ( messageToken .
        getMessage (). getPayload ());
        return MessageCallback . Action . CommitSuccess ;
    }
};
messageClient . setMessage Listener ("/${ YourProductKey }/#",
messageCallback );

```

The parameters are as follows:

- `MessageToken` indicates the body of the returned message. Use `MessageToken . getMessage ()` to get the message body. `MessageToken` is required when you send ACKs manually.

A message body example is as follows:

```

public class Message {
    // Message body
    private byte [] payload ;
    // Topic
    private String topic ;
    // Message ID
    private String messageId ;
    // QoS
    private int qos ;
}

```

- For more information, see [Message body format](#) .
- `messageClient . setMessage Listener ("/${ YourProductKey }/#", messageCallback );` is a method to specify topics for callbacks.

You can specify topics for callbacks, or you can use the generic callback.

- Callbacks with specified topics

Callbacks with specified topics have higher priority than the generic callback . When a message matches with multiple topics, the callback with the topic whose elements rank higher in the lexicographical order is called and only one callback is performed.

When you are configuring a callback, you can specify the topics with wildcards, for example, `/${ YourProductKey }/${ YourDevice Name }/#`.

Example:

```

messageClient . setMessage Listener ("/ alEddfaXXX X /
device1 /#", messageCallback );

```

```
// When the received message matches with the
// specified topic , for example , "/ alEddfaXXX X / device1 /
// update ", the callback with this topic is called .
```

- **Generic callback**

If you do not specify any topic for callbacks, the generic callback is called.

The method for configuring the generic callback is as follows:

```
messageCli ent . setMessage Listener ( messageCal lback );
// If the received message does not match with any
// specified topics which are configured for callbacks
// , the generic callback is called .
```

- **Configure ACK reply**

After a message with QOS>0 is consumed, an ACK must be sent as the reply. SDKs support sending ACKs as replies both automatically and manually. The default setting is to reply with ACKs automatically. In this example, no ACK reply setting is configured, so the system replies with ACKs automatically.

- **Reply ACKs automatically:** If the returned value of `MessageCal lback . consume` is `true` , the SDK will reply an ACK automatically; If the returned value is `false` or an exception occurs, the SDK will not reply with any ACK. If no ACK is replied for the messages with QOS>0, the server will send the message again.
- **Reply ACKs manually:** Use `MessageCli ent . setManualA cks` to configure for replying ACKs manually.

Call `MessageCli ent . ack ( )` to reply ACKs manually, and the parameter `MessageTok en` is required. You can obtain the value of `MessageTok en` from the received message.

The method to manually reply ACKs is as follows:

```
messageCli ent . ack ( messageTok en );
```

Message body format

- **Device status notification:**

```
{
  " status ":" online | offline ",
  " productKey ":" 1234556556 9 ",
  " deviceName ":" deviceName 1234 ",
  " time ":" 2018 - 08 - 31 15 : 32 : 28 . 205 ",
  " utcTime ":" 2018 - 08 - 31T07 : 32 : 28 . 205Z ",
  " lastTime ":" 2018 - 08 - 31 15 : 32 : 28 . 195 ",
```

```
" utcLastTime ":" 2018 - 08 - 31T07 : 32 : 28 . 195Z ",
" clientIp ":" 123 . 123 . 123 . 123 "
}
```

Parameter	Type	Description
status	String	Device status: online or offline.
productKey	String	The unique identifier of the product to which the device belongs.
deviceName	String	The name of the device.
time	String	The time when the notification is sent.
utcTime	String	The UTC time when the notification is sent.
lastTime	String	The time when the last communication occurred before this status change.
utcLastTime	String	The UTC time when the last communication occurred before this status change.
clientIp	String	The Internet IP address for the device.

**Note:**

We recommend that you maintain your device status according to the value of the parameter `lastTime`.

- Device lifecycle change:

```
{
  " action " : " create | delete | enable | disable ",
  " iotId " : " 4z819VQHk6 VSLmmBJfrf 00107ee201 ",
  " productKey " : " 1234556556 9 ",
  " deviceName " : " deviceName 1234 ",
  " deviceSecret " : "",
  " messageCreateTime ": 1510292739 881
}
```

Parameter	Type	Description
action	String	- create: Create devices. - delete: Delete devices. - enable: Enable devices. - disable: Disable devices.
iotId	String	The unique identifier of the device within IoT Platform.
productKey	String	The ProductKey of the product.
deviceName	String	The name of the device.

Parameter	Type	Description
deviceSecret	String	The device secret. This parameter is included only when the value of action is create.
messageCreateTime	Long	The timestamp when the message is generated, in milliseconds.

- Device topological relationship change:

```
{
  "action": "add | remove | enable | disable",
  "gwIotId": "4z819VQHk6 VSLmmBJfrf 00107ee200",
  "gwProductKey": "1234556554",
  "gwDeviceName": "deviceName 1234",
  "devices": [
    {
      "iotId": "4z819VQHk6 VSLmmBJfrf 00107ee201",
      "productKey": "1234556556 9",
      "deviceName": "deviceName 1234"
    }
  ],
  "messageCreateTime": 1510292739881
}
```

Parameter	Type	Description
action	String	- add: Add topological relationships. - remove: Delete topological relationships. - enable: Enable topological relationships. - disable: Disable topological relationships.
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The ProductKey of the product to which the gateway device belongs.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices whose topological relationship with the gateway will be changed.
iotId	String	The unique identifier of the sub-device.
productKey	String	The ProductKey of the product to which the sub-device belongs.
deviceName	String	The name of the sub-device.
messageCreateTime	Long	The timestamp when the messages is generated, in milliseconds.

- A gateway detects and reports sub-devices:

```
{
  " gwIotId ":" 4z819VQHk6  VSLmmBJfrf  00107ee200 ",
  " gwProductKey ":" 1234556554 ",
  " gwDeviceName ":" deviceName  1234 ",
  " devices ":[
    {
      " iotId ":" 4z819VQHk6  VSLmmBJfrf  00107ee201 ",
      " productKey ":" 1234556556  9 ",
      " deviceName ":" deviceName  1234 "
    }
  ]
}
```

Parameter	Type	Description
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The unique identifier of the gateway product.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices detected by the gateway.
iotId	String	The unique identifier of the sub-device.
productKey	String	The ProductKey of the product that the sub-device belongs to.
deviceName	String	The name of the sub-device.

1.8.3 Development guide for .NET HTTP/2 SDK

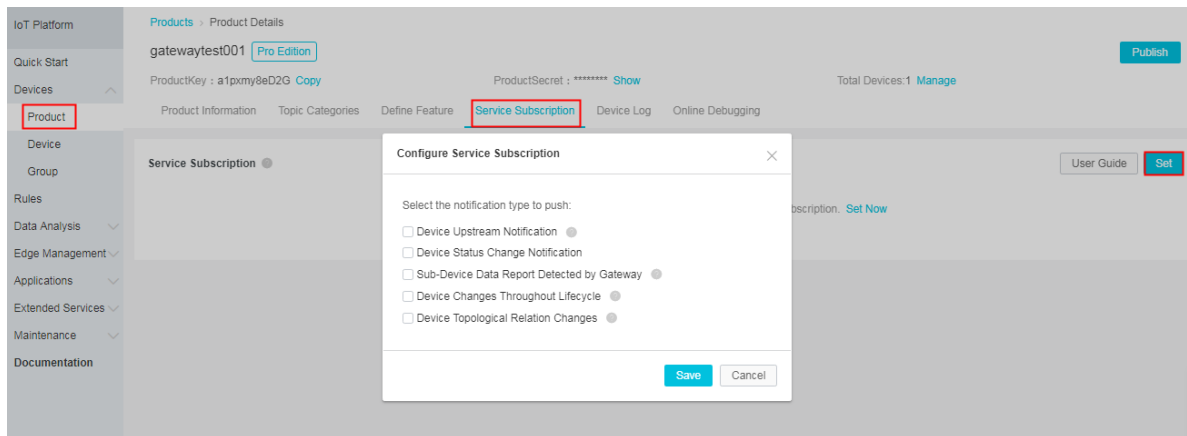
This topic describes how to configure service subscription, connect a .NET SDK of the HTTP/2 server to IoT Platform, perform identity authentication, and set the message receiving interface.

The following process describes how to develop service subscription. Download the [server side .NET SDK demo](#).

Configure service subscription

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

5. Select the types of notifications that you want to push.



- **Device Upstream Notification:** Indicates the messages in the topics to which devices are allowed to publish messages. If this notification type is selected, the HTTP/2 SDK can receive messages reported by devices.

Devices can report both custom data and TSL data of properties, events, responses to property settings, and responses to service callings .

For example, a product has three topic categories:

- `/ ${YourProductKey} / ${YourDeviceName} / user / get` , to which devices can subscribe.
- `/ ${YourProductKey} / ${YourDeviceName} / user / update` , to which devices can publish messages.
- `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` , to which devices can publish messages.

Service subscription can push messages in the topics `/ ${YourProductKey} / ${YourDeviceName} / user / update` and `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` , to which devices can publish messages. Additionally, the messages in the topics `/ sys / ${YourProductKey} / ${YourDeviceName} / thing / event / property / post` are processed by the system.

- **Device Status Change Notification:** Indicates the notifications that are sent when the statuses of devices change, for example, the device connection and device disconnection notifications. The topic which is used to send device status change notifications: `/ as / mqtt / status / ${YourProductKey} / ${YourDeviceName}`

`YourDevice Name }`. After this notification type is selected, the HTTP/2 SDK can receive the device status change notifications.

- *Sub-Device Data Report Detected by Gateway:* A gateway can report the information about sub-devices that are discovered locally. Make sure that the gateway has an application that can discover and report sub-device information.
- *Device Topological Relation Changes:* It includes notifications about device topological relation change.
- *Device Changes Throughout Lifecycle:* It includes notifications about device creation, deletion, disabling, and enabling.



Note:

For messages of device properties, events, and services, Device Status Change Notification, Sub-Device Data Report Detected by Gateway, Device Topological Relation Changes, and Device Changes Throughout Lifecycle, the QoS is 0 by default. For other Device Upstream Notification messages (except messages of device properties, events, and services), you can set the QoS is 0 or 1 on your device SDK.

Connect the HTTP/2 SDK to IoT Platform

Add dependency package [iotx-as-http2-net-sdk.dll](#) to a project.

Authenticate identity

To use the service subscription feature, you must use the AccessKey information of your account for identity authentication and establish a connection between the HTTP/2 SDK and IoT Platform.

Example:

```
// The AccessKey ID of your Alibaba Cloud account
string accessKey = "xxxxxxxxxx xxxxx ";
// The AccessKey Secret of your Alibaba Cloud account
string accessSecret = "xxxxxxxxxx xxxxx ";
// The region ID of your IoT Platform service
string regionId = "cn-shanghai ";
// The UID of your Alibaba Cloud account
string uid = "xxxxxxxxxx xxxxx ";
// The domain name
string domain = ".aliyuncs.com ";
// The endpoint
string endpoint = "https://" + uid + ".iot-as-http2." +
regionId + domain ;

// Configure connection parameters
Profile profile = new Profile ();
profile.AccessKey = accessKey ;
profile.AccessSecret = accessSecret ;
```

```

profile . RegionId = regionId ;
profile . Domain = domain ;
profile . Url = endpoint ;
// Clear accumulated messages
profile . CleanSession = true ;
profile . GetAccessKeyAuthParams ();

// Construct the client
IMessageClient client = new MessageClient ( profile );

// Connect to the HTTP / 2 server to receive messages
client . DoConnect ( new DefaultHttp2MessageCallback () );

// Configure a specified topic for callback
client . SetMessageListener ("/${ YourProductKey }/#", new
CustomHttp2MessageCallback () );

```

The value of `accessKey` is the AccessKey ID of your account, and the value of `accessSecret` is the AccessKey Secret corresponding to the AccessKey ID. Log on to the [Alibaba Cloud console](#), hover over your account avatar, and click AccessKey to view your AccessKey ID and AccessKey Secret. You can also click Security Settings to view your account ID.

The value of `regionId` is the region ID of your IoT Platform service.

Configure the message receiving interface

After the connection is established, IoT Platform immediately pushes the subscribed messages to the HTTP/2 SDK. Therefore, you must configure the message receiving interface.

The message receiving interface is as follows:

```

public interface IHttp2MessageCallback
{
    ConsumeAction Consume ( Http2ConsumeMessage http2ConsumeMessage );
}

```

You must use the `consume` method of `IHttp2MessageCallback` to set the message receiving interface.

Configure the message receiving interface as follows:

```

public class DefaultHttp2MessageCallback : IHttp2MessageCallback
{
    public DefaultHttp2MessageCallback ()
    {
    }

    public ConsumeAction Consume ( Http2ConsumeMessage http2ConsumeMessage )
    {
    }
}

```

```

        Console.WriteLine("receive : " + http2ConsumerMessage.MessageId);
        // Automatically return an ACK
        return ConsumeAction.CommitSuccess;
    }
}

```

The parameters are as follows:

- `Http2ConsumerMessage` indicates the body of the returned message.

A message body contains the following information:

```

public class Http2ConsumerMessage
{
    // The message body
    public byte[] Payload { get; set; }
    // The topic
    public string Topic { get; set; }
    // The message ID
    public string MessageId { get; set; }
    // QoS
    public int Qos { get; set; }
    // The connection parameter
    public Http2Connection Connection { get; set; }
}

```

- For more information, see [Message body format](#).
- `messageClient.SetMessageListener("/${YourProductKey}/#", messageCallback);` is a method to configure a callback. In this example, a topic is configured for the callback.

You can specify topics for callbacks, or you can use the generic callback.

- Topic-specific callbacks

A topic-specific callback has higher priority than the generic callback. When a message matches multiple topics, the callback with the topic whose elements rank higher in the lexicographical order is called and only one callback is performed.

When you configure a callback, you can specify a topic with wildcards, for example, `/${YourProductKey}/${YourDeviceName}/#`.

Example:

```

client.SetMessageListener("/alEddfaXXX / device1 /#",
messageCallback);

```

```
// When the received message matches the specified
// topic , for example , "/ alEddfaXXX X / device1 / update ",
// the callback with this topic is called .
```

- **Generic callback**

If you do not specify any topic for a callback, the generic callback is called.

Configure the generic callback as follows:

```
new DefaultHttp2MessageCallback ()
```

- **Configure ACK replies.**

After a message with QoS>0 is consumed, an ACK must be sent as the reply. The HTTP/2 SDK supports sending ACKs as replies both automatically and manually. By default, ACKs are returned automatically. In this example, no ACK reply setting is configured, so the system returns ACKs automatically.

- **Return ACKs automatically:** If the returned value of `Http2MessageCallback.consume` is `ConsumeAction.CommitSuccess`, the HTTP/2 SDK will return an ACK automatically. If the returned value is `ConsumeAction.CommitFailure` or an exception occurs, the HTTP/2 SDK will not return any ACK. If no ACK is returned for a message with QoS>0, IoT Platform will send the message again.
- **Return ACKs manually:** Use `ConsumeAction.CommitFailure` to manually return an ACK.

Call the `MessageClient.doAck()` method to return ACKs manually. The method contains the following parameters: `topic`, `messageId`, and the connection parameter. You can obtain these parameter values from the received message.

Manually return an ACK as follows:

```
client.doAck ( connection , topic , messageId , delegate );
```

Message body format

- **Device status notification:**

```
{
  " status ":" online | offline ",
  " productKey ":" 1234556556 9 ",
  " deviceName ":" deviceName 1234 ",
  " time ":" 2018 - 08 - 31 15 : 32 : 28 . 205 ",
  " utcTime ":" 2018 - 08 - 31T07 : 32 : 28 . 205Z ",
  " lastTime ":" 2018 - 08 - 31 15 : 32 : 28 . 195 ",
```

```

    " utcLastTime " : " 2018 - 08 - 31T07 : 32 : 28 . 195Z ",
    " clientIp " : " 123 . 123 . 123 . 123 "
  }

```

Parameter	Data type	Description
status	String	The device status: online or offline.
productKey	String	The unique identifier of the product to which the device belongs.
deviceName	String	The name of the device.
time	String	The time when the notification was sent.
utcTime	String	The UTC time when the notification was sent.
lastTime	String	The time when the last communication occurred before this status change.
utcLastTime	String	The UTC time when the last communication occurred before this status change.
clientIp	String	The public outbound IP address of the device.

**Note:**

We recommend that you maintain your device status according to the value of the lastTime parameter.

- Device lifecycle change notification:

```

{
  " action " : " create | delete | enable | disable ",
  " iotId " : " 4z819VQHk6 VSLmmBJfrf 00107ee201 ",
  " productKey " : " 1234556556 9 ",
  " deviceName " : " deviceName 1234 ",
  " deviceSecret " : "",
  " messageCreateTime " : 1510292739 881
}

```

Parameter	Data type	Description
action	String	- create: Create a device. - delete: Delete a device. - enable: Enable a device. - disable: Disable a device.
iotId	String	The unique identifier of the device in IoT Platform.
productKey	String	The unique identifier of the product to which the device belongs.

Parameter	Data type	Description
deviceName	String	The name of the device.
deviceSecret	String	The device key. This parameter is included only when the value of action is create.
messageCreateTime	Long	The timestamp when the message was generated , in milliseconds.

- Device topological relationship change notification:

```
{
  " action " : " add | remove | enable | disable ",
  " gwIotId ": " 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
  " gwProductKey ": " 1234556554 ",
  " gwDeviceName ": " deviceName 1234 ",
  " devices ": [
    {
      " iotId ": " 4z819VQHk6 VSLmmBJfrf 00107ee201 ",
      " productKey ": " 1234556556 9 ",
      " deviceName ": " deviceName 1234 "
    }
  ],
  " messageCreateTime ": 1510292739 881
}
```

Parameter	Data type	Description
action	String	- add: Build topological relationships. - remove: Delete topological relationships. - enable: Enable topological relationships. - disable: Disable topological relationships.
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The unique identifier of the product to which the gateway device belongs.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices whose topological relationships with the gateway will be changed.
iotId	String	The unique identifier of the sub-device.
productKey	String	The unique identifier of the product to which the sub-device belongs.
deviceName	String	The name of the sub-device.
messageCreateTime	Long	The timestamp when the message was generated , in milliseconds.

- A gateway detects and reports sub-devices:

```
{
  " gwIotId ":" 4z819VQHk6  VSLmmBJfrf  00107ee200 ",
  " gwProductKey ":" 1234556554 ",
  " gwDeviceName ":" deviceName  1234 ",
  " devices ":[
    {
      " iotId ":" 4z819VQHk6  VSLmmBJfrf  00107ee201 ",
      " productKey ":" 1234556556  9 ",
      " deviceName ":" deviceName  1234 "
    }
  ]
}
```

Parameter	Data type	Description
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The unique identifier of the product to which the gateway device belongs.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices discovered by the gateway.
iotId	String	The unique identifier of the sub-device.
productKey	String	The unique identifier of the product to which the sub-device belongs.
deviceName	String	The name of the sub-device.

1.8.4 Limits

Service Subscription has the following limits.

Item	Limit description
JDK version	Only JDK 8 is supported.
Authentication timeout	Once the connection is established, an authentication request is sent immediately. If the authentication is not successful within 15 seconds, the server will close the connection.

Item	Limit description
Receiving data timeout	After the connection is established, the client sends ping packets regularly to maintain the connection. You can set the interval for sending ping packets on your clients. The default value is 30 seconds. The maximum value is 60 seconds. If no Ping packet or data is sent in 60 seconds, the server will close the connection. If the client has not received any pong packets in the specified time period, the SDK will close the connection and then try to connect again later. The default interval is 60 seconds.
Pushing message timeout	The server pushes again 10 failed messages in bulk each time . If the server does not receive an ACK from the client after 10 seconds, the message push times out.
Repush policy for failed messages	The stacked messages (due to client being offline, slow message consumption, or other reasons) are repushed every 60 seconds.
Message storage time	Messages with QoS 0 are saved for one day, and messages with QoS 1 are saved for seven days.
Number of SDK instances	Each account can enable up to 64 SDK instances.
Message limit for each tenant	The maximum number of messages sent each second for a single tenant is 1,000 QPS. If your business requires more, you can open a ticket and make a request.

1.8.5 Subscribe to device messages by using Message Service

IoT Platform allows cloud applications to receive device messages by listening to queues in Message Service (MNS). This topic describes how to subscribe to device messages by using Message Service.

Procedure

1. In the IoT Platform console, configure service subscription for a product. IoT Platform can automatically forward messages to queues in Message Service.
 - a) From the left-side navigation pane, choose Devices > Product. On the Products page, select a product, and click View in the Actions column.
 - b) Click the Service Subscription tab.
 - c) On the Service Subscription tab page, click Set corresponding to User Service Client(Push MNS). Then in the dialog box that appears, select the types of messages that you want to push to MNS.

After the subscription is complete, IoT Platform automatically creates a message queue in MNS . Details about the message queue are displayed on the Service Subscription tab page.

The screenshot shows the 'Products > Product Details' page for a product named 'Weather'. The 'ProductKey' is 'a104v5UjY0a' and the 'ProductSecret' is '*****'. The 'Service' tab is selected, showing the 'User Service Client(Push MNS)' section. Below this, a table displays subscription details.

Subscribed	
Region	cn-shanghai
Queue	aliyun-iot-a104v5UjY0a
Role Name	AliyunIOTAccessingMNSRole Role Details

2. Receive device messages by listening to the message queue.

In this example, enter the following information to use the Java SDK of Message Service.

For more information, see [Message Service documentation](#).

- In the pom.xml file, add the following dependencies:

```
< dependency >
  < groupId > com . aliyun . mns </ groupId >
  < artifactId > aliyun - sdk - mns </ artifactId >
  < version > 1 . 1 . 8 </ version >
  < classifier > jar - with - dependenci es </ classifier >
</ dependency >
```

- When you configure message receiving, enter the following information:

```
CloudAccou nt account = new CloudAccou nt ($ AccessKeyI
d , $ AccessKeyS ecret , $ AccountEnd point );
```

- Replace \$ AccessKeyI d and \$ AccessKeyS ecret with your AccessKey ID and AccessKey Secret values. These values are required for you to access APIs. You can find these values in your profile by clicking the Alibaba Cloud account avatar.
- Replace \$ AccountEnd point with the actual endpoint value. You can obtain this value from the Message Service console.
- Enter the logic for receiving device messages:

```
MNSClient client = account . getMNSClie nt ();
CloudQueue queue = client . getQueueRe f (" aliyun - iot -
alxxxxxx8o 9 "); // Enter the name of the queue that
has been automatica lly created

while ( true ) {
  // Get messages
  Message popMsg = queue . popMessage ( 10 ); // The
timeout for long polling is 10 seconds
  if ( popMsg != null ) {
    System . out . println (" PopMessage Body : "+ popMsg
. getMessage BodyAsRawS tring ()); // Get raw messages
    queue . deleteMess age ( popMsg . getReceipt Handle
()); // Delete messages from the queue
  } else {
    System . out . println (" Continuing ");
  }
}
```

- Run the program to listen to the MNS queue.

3. Start a device and send a message from the device to IoT Platform.

To view the content of the reported message, see [SDK reference](#).

4. Check if the cloud applications can listen to the subscribed messages. If they successfully listen to the messages, they will receive messages like the following:

```
{
  "messageid ":" ", // The message ID
  "messagetype ":" upload ",
  "topic ":"// The topic from which the message comes
  "payload ":" // Base64 - encoded data
  "timestamp ":" // The timestamp
}
```

Parameter	Description
messageid	Message ID generated by IoT Platform .
messagetype	The message type. · status: Indicates device status change notifications. · upload: Indicates device upstream notifications. · device_lifecycle: Indicates device lifecycle change messages. · topo_lifecycle: Indicates topological relationship change notifications. · topo_listfound: Indicates messages of sub-devices reported by gateway devices.
topic	The topic from which the message comes.
payload	Base64-encoded data of message payload.
timestamp	The timestamp in the format of Epoch.

1.9 Device group

IoT Platform supports device groups. You can assign devices from different products to the same group. This article introduces how to create and manage device groups in the IoT Platform console.

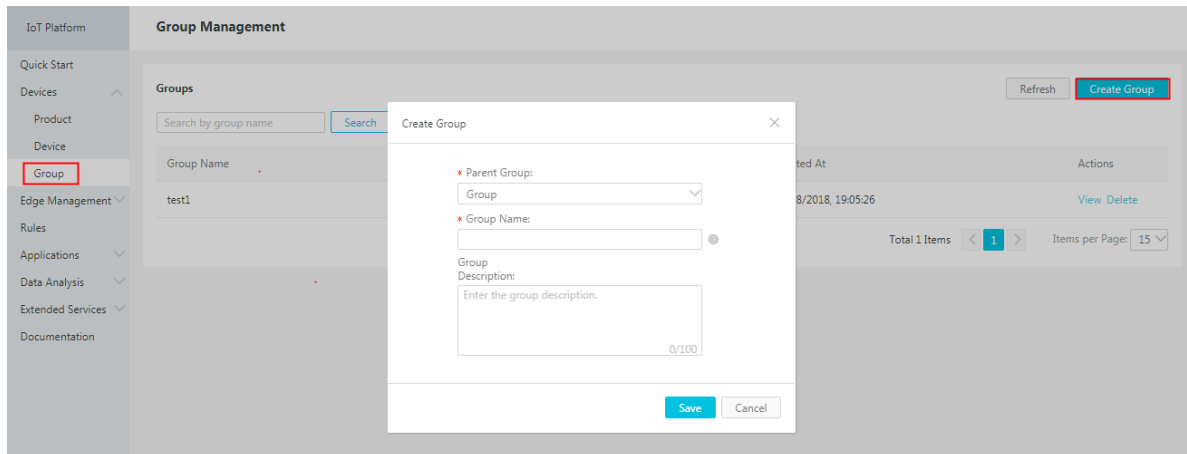
Procedure

1. Log on to the [IoT Platform console](#).
2. Click Devices > Group.
3. On the group management page, click Create Group, enter group information, and then click Save.



Note:

You can create up to 1,000 groups (including parent groups and subgroups) .



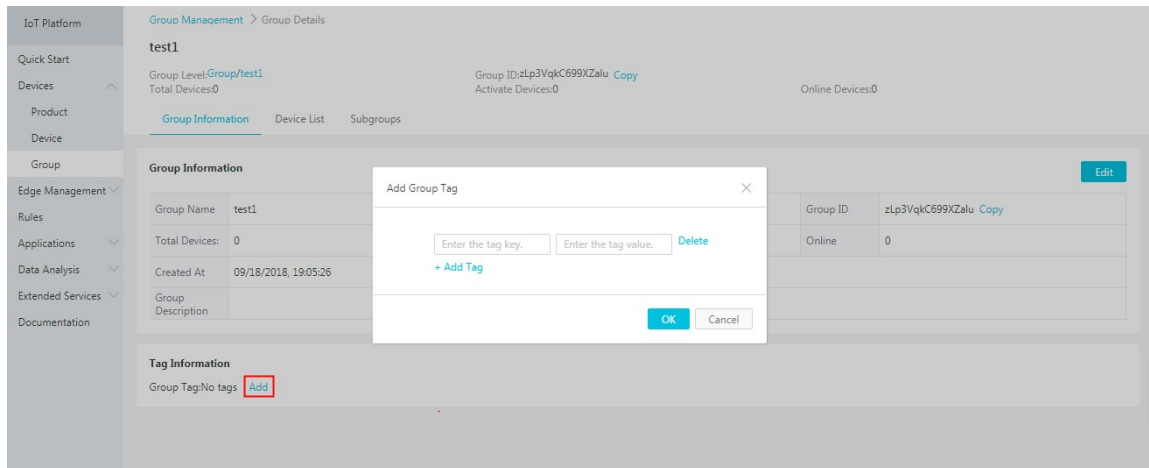
The parameters are as follows:

- **Parent Group:** Select a group type.
 - **Group:** Indicates that the group to be created is a parent group.
 - **Select an existing group:** Specifies a group as the parent group and creates a subgroup for it.
 - **Group Name:** Enter a name for the group. A group name can be 4 to 30 characters in length and can include Chinese characters, English letters, digits and underscores (_) . The group name must be unique among the groups for an account, and cannot be modified once the group has been created.
 - **Group Description:** Describes the group. Can be left empty.
4. On the Group Management page, click View to view the Group Details page of the corresponding group.
 5. (Optional) Add tags for the group. Tags can be used as group identifiers when you manage your groups.
 - a) Click Add under Tag Information, and then enter keys and values of tags.
 - b) Click OK to create all the entered tags.



Note:

You can add up to 100 tags for a group.

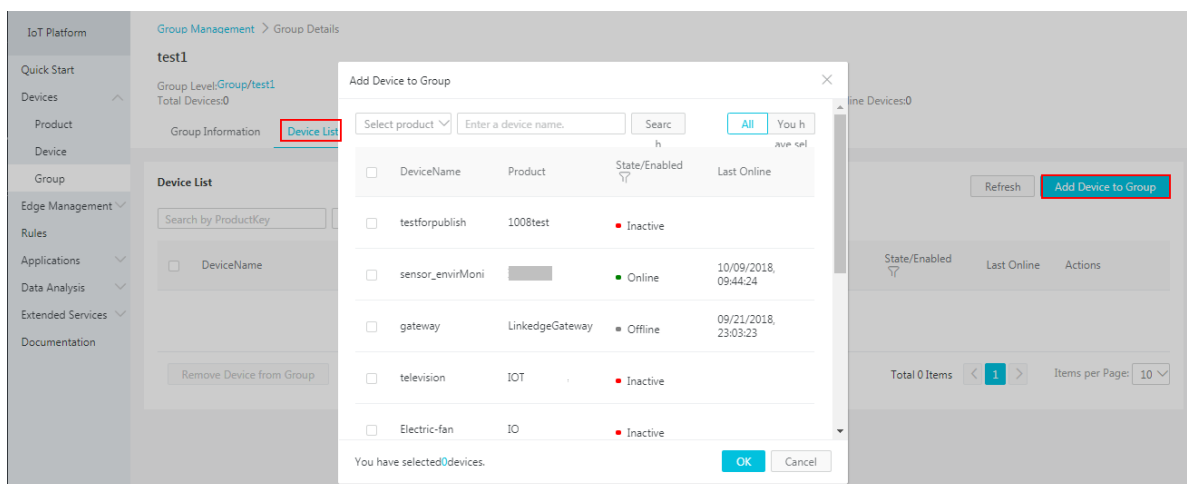


6. Click Device List > Add Device to Group. Select the devices that you want to add to the group.



Note:

- You can add up to 1,000 devices at a time. You can add up to 20,000 devices for a group in total.
- A device can be included in a maximum of 10 groups.



There are two buttons at the upper-right corner of the Add Device to Group page:.

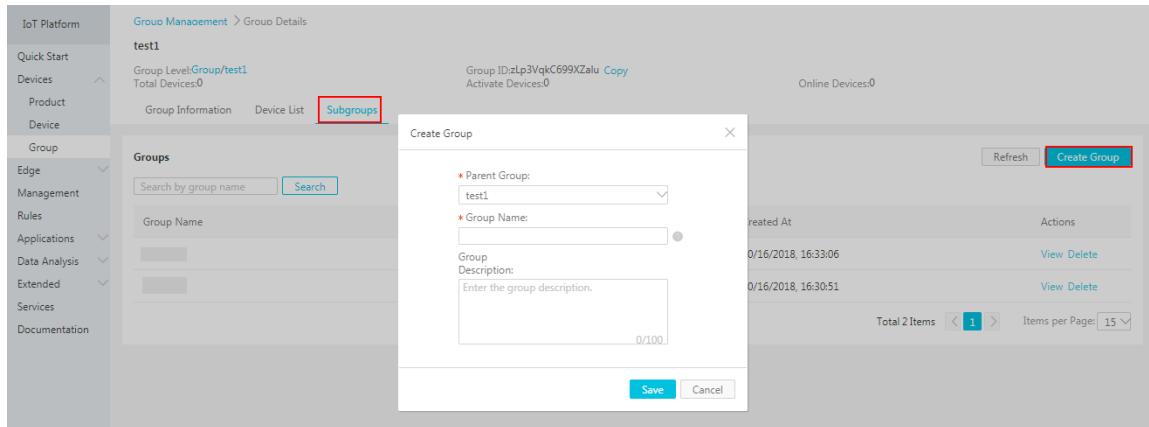
- Click All to display all the devices.
- Click You have selected to display the devices you have selected.

7. (Optional) Click Subgroups > Create Group to add a subgroup for the group.

Subgroups are used to manage devices in a more specific manner. For example, you can create subgroups such as "SmartKitchen" and "SmartBedroom" for a

parent group "SmartHome", and then you can manage your kitchen devices and bedroom devices separately. The procedure is as follows:

a) Select the parent group, enter a group name and description, and click Save.



b) On the Subgroups page of the parent group, click View to view the corresponding Group Details page.

c) Click Device List > Add Device to Group, and then add devices for the subgroup. After creating the subgroup and adding devices for it, you can then manage it. You can also create sub-subgroups within the subgroup.



Note:

- A group can include up to 100 subgroups.
- Only three layers of groups are supported: parent group>subgroup>sub-subgroup.
- A group can only be a subgroup of one parent group.
- You can not change the relationships between a parent group and its subgroups once they have been created. If you want to change the relationships, delete the existing subgroups and create new ones.
- You cannot delete a group that has subgroups. You must delete all its subgroups before deleting the parent group.

1.10 Manage files

IoT Platform allows devices to upload files over HTTP/2 channels to the Alibaba Cloud IoT Platform server for storage. After a file is uploaded, you can download and delete the file in the IoT Platform console.

Prerequisites

- The device is connected to IoT Platform. For more information about device SDK development, see [Link Kit SDK documentation](#).
- The HTTP/2 file upload function is compiled and configured on the device.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose Devices > Device, and then click View next to the corresponding device.

The screenshot displays the IoT Platform console interface. On the left, a navigation menu lists various sections, with 'Device' under the 'Devices' category highlighted by a red box. The main content area is titled 'Devices' and includes summary statistics: 'Total Devices: 151', 'Activate Device: 63', and 'Online: 23'. Below these, there are tabs for 'Device List' and 'Batch Management'. The 'Device List' tab is active, showing a table of devices. The table has columns for 'DeviceName/Alias', 'Product', 'Node Type', 'State/Enabled', 'Last Online', and 'Actions'. Three devices are listed: 'weather1' (Weather product, Online state), 'test0401' (test0401 product, Online state), and 'device1' (20190401 product, Offline state). In the 'Actions' column for the 'weather1' device, the 'View' link is highlighted with a red box.

DeviceName/Alias	Product	Node Type	State/Enabled	Last Online	Actions
weather1	Weather	Device	Online	04/10/2019, 18:39:57	View Delete
test0401	test0401	Device	Online	04/01/2019, 17:27:12	View Delete
device1	20190401	Device	Offline	04/01/2019, 14:54:32	View Delete

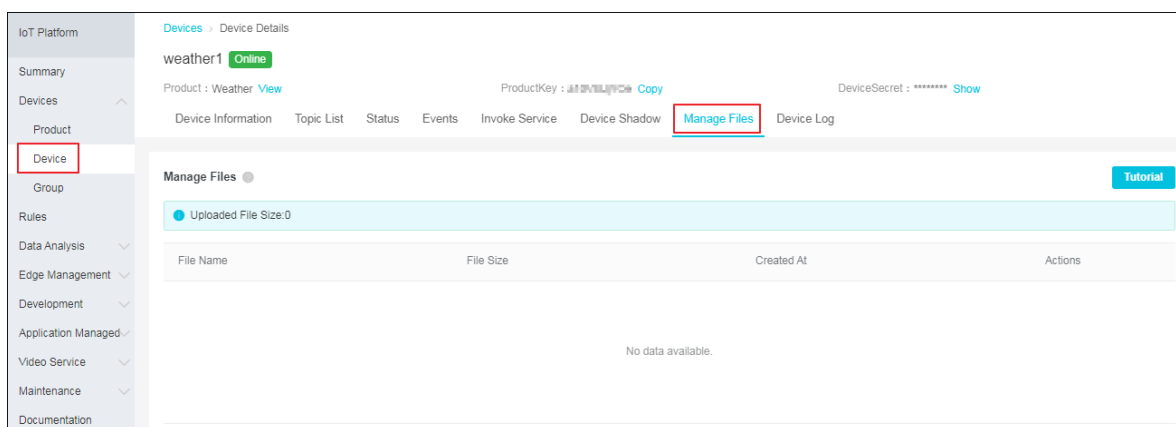
3. On the Device Details page, click the Manage Files tab.

On the Manage Files tab page, you can view the files that were uploaded by the device through the HTTP/2 channel.



Note:

The maximum file size that can be stored on the IoT Platform server for each Alibaba Cloud account is 1 GB. The maximum number of files that can be stored for each device is 1,000.



You can perform the following operations on an uploaded file:

Operation	Description
Download	Download the file to your local device.
Delete	Delete the file.

In addition to file management in the console, you can also query or delete files by calling the following cloud API operations: [QueryDeviceFileList](#), [QueryDeviceFile](#), and [DeleteDeviceFile](#).

2 Rules

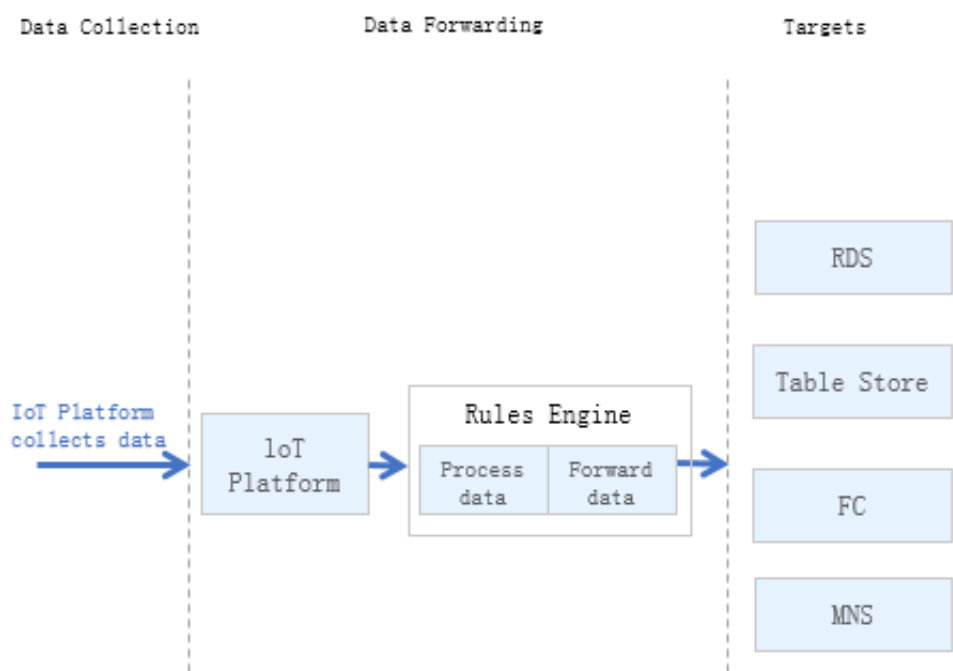
2.1 Data Forwarding

2.1.1 Overview

When your devices communicate using [topics](#), you can use the rule engine and write SQL expressions to process data in topics. You can also configure forwarding rules to send the processed data to other Alibaba Cloud services. For example:

- You can forward the processed data to [RDS](#), and [Table Store](#) for storage.
- You can forward the processed data to [Function Compute](#) for event-driven computing.
- You can forward the processed data to another topic to achieve M2M communication.
- You can forward the processed data to [Message Service](#) to ensure reliable use of data.

By using the rule engine, you will be provided with a complete range of services including data collection, computing, and storage without purchasing a distributed server deployment architecture.



**Note:**

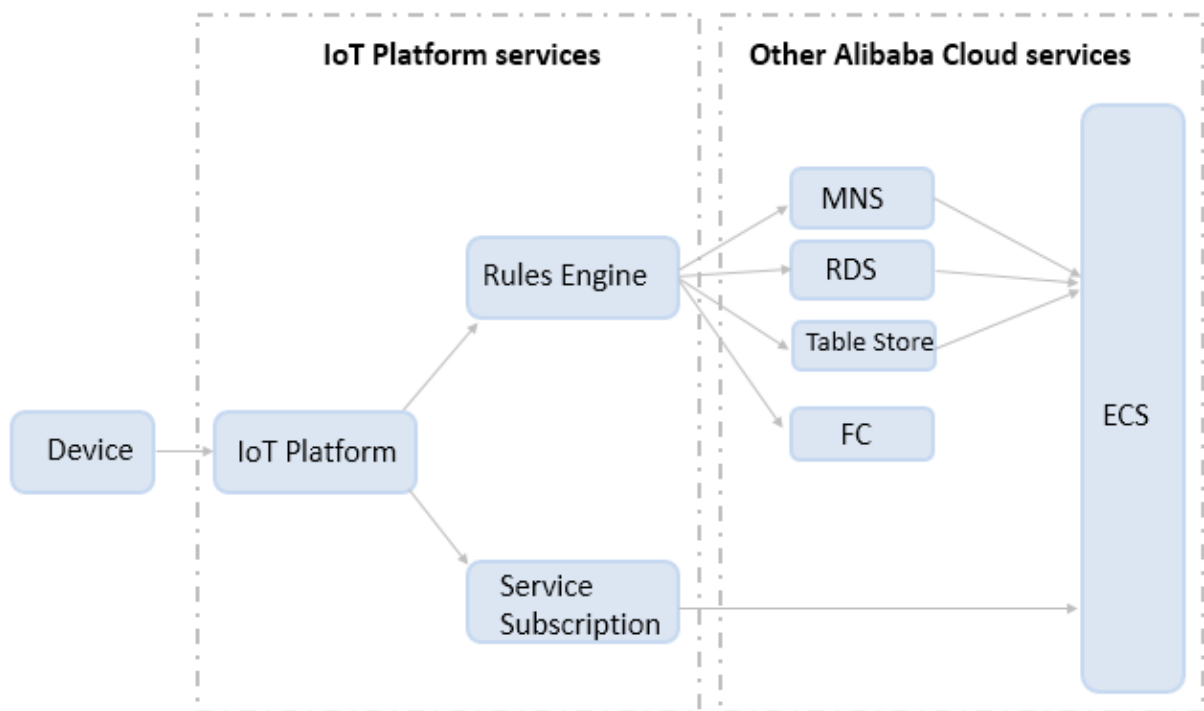
When using the rule engine, you need to pay attention to the following points:

- The rule engine processes data based on topics. You can use the rule engine to process device data only when devices are communicating with each other by using topics.
- The rule engine processes the data in topics using SQL.
- SQL subqueries and the use of the LIKE operator are currently not supported.
- Some functions are supported. For example, you can use `deviceName ()` to obtain the name of the current device. For more information about the supported functions, see Function list.

2.1.2 Compare data forwarding solutions

In many scenarios, you must process the data that is reported by devices or use the data for business applications. You can forward device data by either using the IoT Platform service subscription or the rules engine data forwarding function. This topic compares the various data forwarding solutions and application scenarios that are supported by IoT Platform to help you select a forwarding solution that best suits your needs.

Data forwarding solutions



IoT Platform supports the following functions for data forwarding:

- **Rules engine data forwarding:** Provides basic data filtering and processing capabilities. You can configure forwarding rules to filter and process device data and then forward the data to other Alibaba Cloud services.
- **Service subscription:** Obtains device data directly from HTTP/2 clients. You can quickly obtain device data without being filtered and processed. This function is simple, easy to use, and efficient.

Compare rules engine data forwarding and service subscription

Forwarding function	Scenarios	Advantages and disadvantages	Restrictions
Rules engine data forwarding	• Complex scenarios • High-throughput scenarios	Advantages: • Full fledged. • Allows you to modify forwarding rules while the rules are running. • Supports data filtering and processing. • Allows you to forward data to other Alibaba Cloud services. The following table "Rules engine-based solutions" briefly compares solutions that use the rules engine for forwarding data to different Alibaba Cloud services. Disadvantages: • Complex to use. Users must write SQL expressions and configure forwarding rules.	See Limits for data forwarding .

Forwarding function	Scenarios	Advantages and disadvantages	Restrictions
Service subscription	- Scenarios that simply involves data receiving. - Scenarios that meet the following requirements: - IoT Platform receives all device data. - The device SDK is developed based on the Java or .NET language. - The devices forwards data at a maximum rate of 5,000 queries per second (QPS).	Advantages: - Easy to use. Disadvantages: - Lack of the filtering capability. - Limited language support for the SDK.	See Limits for service subscription .

Table 2-1: Rules engine-based solutions

Forwarding destination	Scenarios	Advantages	Disadvantages
Message Service (MNS)	Device data requires complex or refined processing. Scenarios where the transmit rate is slower than 1,000 QPS.	- Uses the HTTPS protocol. - Allows IoT Platform to forward data on the Internet with high performance.	Provides performance slightly lower than MQ for RocketMQ.
ApsaraDB for RDS	Data storage scenarios.	Writes data directly to databases.	N/A
Table Store	Data storage scenarios.	Writes data directly into Table Store instances.	N/A

Forwarding destination	Scenarios	Advantages	Disadvantages
Function Compute	Scenarios where the device development process must be simplified and device data must be processed in a flexible way.	• Great flexibility in data processing. • Multiple functions. • Do not require deployment.	Higher costs.

Service subscription

Business servers can subscribe to all types of messages by using the SDK.

Restrictions	Guidelines	References
• Only the SDK in Java 8 or later and the .NET SDK are supported. • Service subscription does not support filtering messages. It receives all subscribed messages from the devices. • The transmit rate is up to 1,000 QPS. If your business requires a higher QPS, open a ticket and describe your requirements. For more information about service subscription restrictions, see Limits.	• Scenarios where the maximum transmit rate is no higher than 5,000 QPS. • Make sure that you are fully aware of any impacts of data loss and data delay on your business. Protect important data in the business layer. • Service subscription does not apply to scenarios where data filtering and fine-grained processing are required. We recommend that you use the rules engine data forwarding for these scenarios.	• What is service subscription • Development guide for the Java SDK • Development guide for the .NET SDK • Best practices

Forward data to Message Service

The rules engine enables IoT Platform to forward messages in specific topics to the topics in Message Service. Message Service can then receive these messages by using the Message Service SDK. Message Service allows access from the public network but it provides a lower performance than RocketMQ. We recommend that you use Message Service for scenarios where the transmit rate is lower than 1,000 QPS.

Restrictions	Guidelines	References
	When a message fails to be forwarded by using the rules engine after making the maximum retries, the message will be dropped. Message-oriented services may have delay issues. Make sure that you are fully aware of the impacts of data loss or delay on your business.	• Create and configure a rule • Forward data to Message Service • Message Service documentation

Forward data to Function Compute

The rules engine enables IoT Platform to forward messages in specific topics to Function Compute. Developers can then further process the messages. Function Compute does not require deployment, which simplifies business development.

Restrictions	Guidelines	References
See Function Compute limits .	• Applicable to scenarios where users can customize data processing or are required to simplify the development and operation processes. • When a message fails to be forwarded by using the rules engine after making the maximum retries, the message will be dropped. Make sure that you are fully aware of any impacts of data loss or delay on your business.	• Create and configure a rule • Forward data to Function Compute • Function Compute documentation

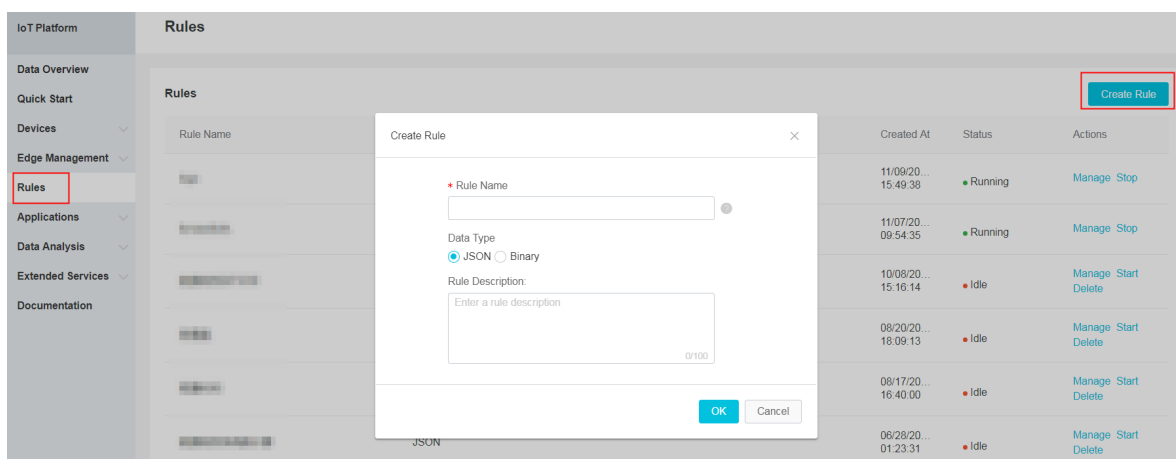
2.1.3 Create and configure a rule

Using the data forwarding feature of the rules engine, IoT Platform can forward specified messages of topics to other IoT Platform topics and other Alibaba Cloud services. This topic describes how to create and configure a rule. The process is to create a rule, write a SQL statement for data processing, configure data forwarding destinations, and configure a forwarding destination for error messages.

Procedure

1. In the left-side navigation pane of the IoT Platform console, click Rules.
2. On the Data Forwarding Rules tab, click Create Data Forwarding Rule.

3. Enter a Rule Name ,select a Data Type , and then click OK.



Parameter	Description
Rule Name	Enter a unique rule name, which is used to identify the rule. A rule name can contain Chinese characters, English letters, digits , underscores (_) and hyphens (-), and must be 1 - 30 characters in length. A Chinese character counts as two characters.
Data Type	Select a data type for the data that this rule processes. Options: JSON and Binary.  Note: • The rules engine processes data based on topics. Therefore , you must select the format of the data in the topic that you want to process. • If the data type is Binary, the rule cannot process data from system-defined topics, and cannot forward data to Table Store or RDS instances.
Rule description	The description of the rule.

4. After the rule has been successfully created, you are directed to the Data Forwarding Rule Details page. On this page, you must edit a SQL statement to

process data, configure data forwarding destinations, and configure a destination for error messages.

The screenshot shows the 'Data Forwarding Rule Details' page for a rule named 'test03'. The page is divided into three main sections: 'Process Data', 'Data Forwarding', and 'Forward Error Data'. Each section has a header bar with a title and an 'Add' button. The 'Process Data' section has a 'Write SQL' button. The 'Data Forwarding' section has an 'Add Operation' button. The 'Forward Error Data' section has an 'Add Misoperation' button. The 'Process Data' section also has a 'SQL Syntax' button. The 'Data Forwarding' section has a 'Data Destination' table with an 'Actions' column. The 'Forward Error Data' section has a 'Data Destination' table with an 'Actions' column. The 'Process Data' section has a message: 'You have not specified a SQL statement for handling data. Write SQL'. The 'Data Forwarding' section has a message: 'No data forwarded , Add Operation'. The 'Forward Error Data' section has a message: 'No data forwarded , Add Misoperation'.

a) Click Write SQL, and then edit a SQL statement for data processing.

In the following example, the statements can retrieve the contents of the `deviceName` field from the messages of the custom topics of all the devices under product `test0306`.



Note:

You can use `to_base64 (*)` to convert binary data to a base64 string. Built-in functions and conditions are also supported.

Write SQL

Rule Query Expression:

Copy statement

SELECT `deviceName()` as deviceName
FROM `"/a1#1u#0000/+/user/#"`
WHERE

Field:

deviceName() as deviceName

Topic :

/a1#1u#0000/+/user/#

Custom

test0306

All equipment (+)

user/#

Condition: (optional)

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

OK

Cancel

The parameters to be configured are as follows. For more information, see [SQL statements](#) and [Functions](#).

Parameter	Description
Rule Query Expression	The system will display the complete SQL statement here according to the values of <code>Field</code> , <code>Topic</code> ,and <code>Condition</code> .

Parameter	Description
Field	Specify the message fields that this rule will retrieve from the message contents. For example, if you enter <code>deviceName ()</code> as <code>deviceName</code>, the rule will retrieve the device names from the messages. For message content data, see Data Format.
Topic	Select the topics whose messages are to be processed by this rule. Topic types: • Custom: The messages are from custom topics. Wildcards <code>+</code> and <code>#</code> are supported when you specify custom topics. To learn how to use wildcards in topics, see Custom topics. • System: Only when the data type is JSON, are system topics available. The messages are from system-defined topics, including messages of reporting properties and events, device lifecycle change, topological relationship change, and gateways reporting sub-devices. For message contents, see Data format. • Device Status: Only when the data type is JSON, can you use a rule to process device status messages, which are messages about devices connecting to and disconnecting from IoT Platform. For message contents, see Data format.
Condition	The condition for triggering the rule.

- b) Click Add Operation next to Data Forwarding. Configure a destination to which you want to forward the processed data. For more information about data forwarding examples, see the documents in Examples.



Note:

A rule can have up to 10 data forwarding destinations.

Currently, if data forwarding fails due to exceptions in the target Alibaba Cloud services, the rules engine retries three times: after one second, after three seconds, and after ten seconds. If all the retries fail, the message will be discarded. If you do not want to miss the forwarding failed messages, you can proceed to the next step: Add Misoperation. You can then add a destination for error messages.

- c) Click Add Misoperation next to Forward Error Data and then create an action to forward error messages about data forwarding failures to a specified target.



Note:

- Error messages and device data cannot be forwarded to the same Alibaba Cloud service. For example, you cannot configure Table Store as the destination for both error messages and device data.

- Rules engine retries three times if data fails to be forwarded to the specified destinations. If all the retries fail, an error message is forwarded according to this configuration.
- If the error message fails to be forwarded, the rules engine does not retry sending the message.
- Here, the term "error messages" refers only to messages that relate to errors resulting from exceptions in the target Alibaba Cloud instance.
- You can add only one destination for error message forwarding.
- Error message format:

```
{
  " ruleName ":"",
  " topic ":"",
  " productKey ":"",
  " deviceName ":"",
  " messageId ":"",
  " base64OriginalPayload ":"",
  " failures ":[
    {
      " actionType ":" OTS ",
      " actionRegion ":" cn - shanghai ",
      " actionResource ":" table1 ",
      " errorMessage ":""
    },
    {
      " actionType ":" RDS ",
      " actionRegion ":" cn - shanghai ",
      " actionResource ":" instance1 / table1 ",
      " errorMessage ":""
    }
  ]
}
```

Parameters in error messages:

Parameter	Description
ruleName	The name of the data forwarding rule.
topic	The source topic of the message.
productKey	The unique identifier of the product that the device belongs to.
deviceName	The device name.
messageId	The message ID that is generated by IoT Platform for this message.
base64OriginalPayload	The original data that has been Base64 encoded .

Parameter	Description
failures	Detailed messages about the failure. There may be multiple error messages if the rule forwards data to multiple destinations.
actionType	The target Alibaba Cloud service to which data fails to be forwarded.
actionRegion	The region of the target Alibaba Cloud service.
actionResource	The target resource.
ErrorMessage	Error message.

5. After you complete all the configurations, go back to the Data Forwarding Rules tab of Rules page, and click Start corresponding to the rule to start this rule. Data will then be forwarded following this rule.

Rules					
Data flow					
Data Forwarding Rules					Create Data Forwarding Rule
Rule Name	Data Type	Rule Description	Created At	Status	Actions
test03	JSON	-	03/07/2019, 11:30:52	Idle	Start View Delete
1111	JSON	<span style="c...	07/27/2018, 15:21:38	Idle	Start View Delete
1112	JSON	</script>	07/27/2018, 15:20:40	Idle	Start View Delete

You can also perform the following operations:

- Click View, and then modify the rule configurations on the Data Forwarding Rule Details page.
- Click Delete to delete this rule.



Note:

Rules that are in a running state cannot be deleted.

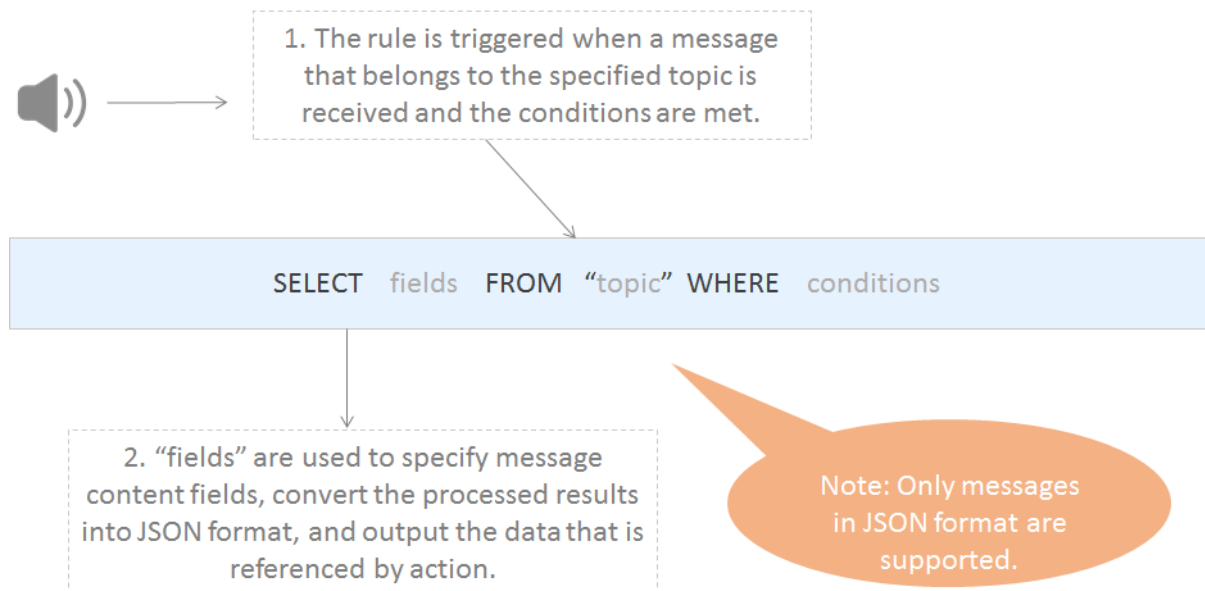
- Click Stop to disable this rule.

2.1.4 SQL statements

You can write SQL statements to parse and process data when you create data forwarding rules. Binary data will not be parsed, but directly passed through to targets. This topic describes SQL statements.

SQL statements

JSON data can be mapped to a virtual table. Keys in a JSON data record correspond to the column names. Values in a JSON data record correspond to the column values. After being mapped to a virtual table, a JSON data record can be processed using SQL. The following example demonstrates how to represent a data forwarding rule as a SQL statement.



For example, an environmental sensor that is typically used for fire detection and collecting temperature, humidity, and atmospheric pressure data, reports the following data:

```
{
  "temperature": 25.1
  "humidity": 65
  "pressure": 101.5
  "location": "xxx, xxx"
}
```

Assume that you need to set an alarm that is triggered when the temperature is higher than 38 °C and the humidity is lower than 40 %, write the following SQL statement as a rule:

```
SELECT temperature as t, deviceName() as deviceName,
location FROM /ProductA /+ update WHERE temperature > 38
and humidity < 40
```

If the reported data meets the rule parameters, the rule is triggered and the temperature data is

```
parsed to obtain the information about temperature ,
device name , and location for further processing .
```

FROM clause

You can enter a topic in the FROM clause. You can enter a wildcard character `+` that includes all topics on the current category level to match the topic whose device messages need to be processed. When a message that matches the specified topic is received, only the message payload that is in the JSON format can be parsed and then processed by the SQL statement that you have defined. Invalid messages are ignored. You can use the `topic ()` function to reference a specific topic.

In this example, the "FROM /ProductA/+/update" clause indicates that only messages that match the /ProductA/+/update format are processed. For more information about matching rules, see [Topic](#).

SELECT statement

· JSON data

In the SELECT statement, you can use the result of parsing the payload of the reported message that represents the keys and values in the JSON data. You can also use built-in functions in the SQL statement, such as `deviceName ()`.

You can combine `*` with functions. SQL subqueries are not supported.

The reported JSON data can be an array or nested JSON data. You can also use a JSONPath expression to obtain values in the reported data record. For example, for a payload `{ a : { key1 : v1 , key2 : v2 } }`, you can obtain the value `v2` by specifying `a . key2` as the JSON path. When specifying variables in SQL statements, note the difference between single quotation marks (`'`) and double quotation marks (`"`). Constants are enclosed with single quotation marks (`'`). Variables are enclosed with double quotation marks (`"`). Variables may also be written without being enclosed by quotation marks. For example, `a . key2` represents a constant whose value is `a . key2`.

For more information about built-in functions, see [Functions](#).

```
In the statement " SELECT temperature as t ,
deviceName () as deviceName , location " that is provided
in the previous example , temperature and location
```

are the fields in the reported message , and `deviceName ()` is a built-in function .

- Binary data

- Enter `*` to pass through binary data directly. You cannot add a function after `*`.
- You can use built-in functions. The `to_base64 (*)` function converts the payload that is binary data to a base64 string. The `deviceName ()` function extracts the name information of a device.



Note:

Each SELECT statement can contain up to fifty fields.

WHERE clause

- JSON data

The WHERE clause is used as the condition for triggering the rule. SQL subqueries are not supported. The fields that can be used in the WHERE clause are the same as those that can be used in the SELECT statement. When a message of the corresponding topic is received, the results obtained using the WHERE clause will be used to determine whether a rule will be triggered. For more information about conditional expressions, see the following table: Supported conditional expressions.

In the previous example , " WHERE temperatur e > 38 and humidity < 40 " indicates that the rule is triggered when the temperatur e is higher than 38 ° C and the humidity is lower than 40 %.

- Binary data

If the reported message is composed of binary data, you can only use built-in functions and conditional expressions in the WHERE clause. You cannot use the fields in the payload of the reported message.

SQL results

The SQL result returned after the SQL statement is executed will be forwarded. If an error occurs while parsing the payload of the reported message, the rule execution fails. In the expression used for data forwarding, you must use `${ expression }` to specify the data that you want to forward.

In the previous example , when configurin g the data forwarding action , you can use `${ t }`, `${ deviceName }`, and `${ loaction }` to reference the SQL result . For example

, if you want to forward the SQL result to Table Store, you can use `${ t }`, `${ deviceName }`, and `${ loaction }`.

Notes on arrays

Array expressions are enclosed with double quotation marks ("). Use `$.` to obtain a JSONObject. `$.` can be omitted. Use `.` to obtain a JSONArray.

If the device message is `{" a ": [{" v ": 0 }, {" v ": 1 }, {" v ": 2 } ]}`, results of different expressions are as follows:

- The result of `" a [ 0 ]"` is `{" v ": 0 }`
- The result of `"$. a [ 0 ]"` is `{" v ": 0 }`
- The result of `". a [ 0 ]"` is `[ {" v ": 0 } ]`
- The result of `" a [ 1 ]. v "` is `1`
- The result of `"$. a [ 1 ]. v "` is `1`
- The result of `". a [ 1 ]. v "` is `[ 1 ]`

Supported WHERE expressions

Operator	Description	Example
=	Equal to	color = 'red'
<>	Not equal to	color <> 'red'
AND	Logic AND	color = 'red' AND siren = 'on'
OR	Logic OR	color = 'red' OR siren = 'on'
()	Conditions that are enclosed with parentheses () are considered as a whole.	color = 'red' AND (siren = 'on' OR isTest)
+	Addition	4 + 5
-	Subtraction	5-4
/	Division	20 / 4
*	Multiplication	5 * 4
%	Return the remainder	20% 6
<	Less than	5 < 6
<=	Less than or equal to	5 <= 6

>	Greater than	6 > 5
>=	Greater than or equal to	6 >= 5
Function call	For more information about supported functions, see Functions .	deviceId()
Attributes expressed in the JSON format	You can extract attributes from the message payload and express them in the JSON format.	state.desired.color,a.b.c[0].d
CASE ... WHEN ... THEN ... ELSE ... END	CASE expression. Nested expressions are not supported.	CASE col WHEN 1 THEN 'Y' WHEN 0 THEN 'N' ELSE ' ' END as flag
IN	Only listing is supported. Subqueries are not supported.	For example, you can use WHERE a IN(1, 2, 3). However, you cannot use WHERE a IN(select xxx).
LIKE	This operator is used to match a specific character. When you use a LIKE operator, you can only use the % wildcard character to represent a character string.	For example, you can use the LIKE operator in WHERE c1 LIKE '%abc' and WHERE c1 not LIKE '%def%' .

2.1.5 Functions

The rules engine provides functions that allow you to handle data when writing a SQL script.

Call functions

In SQL statement, you can use functions to get or handle data.

For example, in the following example, the functions: `deviceName()`, `abs(number)`, and `topic(number)` are used.

```
SELECT case flag when 1 then ' Light On ' when 2
then ' Light Off ' else '' end flag , deviceName (), abs (
temperatur e ) tmr FROM "/" topic /#" WHERE temperatur e > 10
and topic ( 2 )=' 123 '
```



Note:

When you use functions, note that constants are enclosed with apostrophes (').

Variables are not enclosed or are enclosed with quotation marks ("). For example, in

```
select " a " a1 , ' a ' a2 , a a3 , a1 is equivalent to a3 , and a2
represents a constant a .
```

Function name	Description
abs(number)	Returns the absolute value of the number.
asin(number)	Returns the arcsine of the number.
attribute(key)	Returns the device tag that corresponds with the key. If a tag with the specified key is not found, the returned value is null . When you debug your SQL statements, because there is no real device or tag, the returned value is null.
concat(string1, string2)	Strings. Example: concat(field,' a').
cos(number)	Returns the cosine of the number.
cosh(number)	Returns the hyperbolic cosine of the number.
crypto(field,String)	Encrypts the value of the field. The String parameter represents an algorithm. Available algorithms include MD2, MD5, SHA1, SHA-256, SHA-384, and SHA-512.
deviceName()	Returns the name of the current device. When you debug your SQL statements, because there is no real device, the returned value is null.
endswith(input, suffix)	Validates whether the input value ends with the suffix string.
exp(number)	Returns a value raised to the power of a number.
floor(number)	Rounds a number down, toward zero, to the nearest multiple of significance. Returns an integer that is equal to or smaller than the number.
log(n, m)	Returns the logarithm of a number according to the base that you have specified. If you do not specify the value of m, log(n) is returned.
lower(string)	Returns a lower-case string.
mod(n, m)	Returns the remainder after a number has been divided by a divisor.

Function name	Description
<code>nanvl(value, default)</code>	Returns the value of a property. If the value of the property is null, the function returns default.
<code>newuuid()</code>	Returns a random UUID.
<code>payload(textEncoding)</code>	Returns the string generated by encoding the message payload that is sent by a device. The default encoding is UTF-8, which means that <code>payload()</code> and <code>payload('utf-8')</code> will return the same result.
<code>power(n,m)</code>	Raises number <code>n</code> to power <code>m</code> .
<code>rand()</code>	Returns a random number greater than or equal to 0 and less than 1.
<code>replace(source, substring, replacement)</code>	Replaces a specific column. Example: <code>replace(field,' a' , ' 1')</code> .
<code>sin(n)</code>	Returns the sine of <code>n</code> .
<code>sinh(n)</code>	Returns the hyperbolic sine of <code>n</code> .
<code>tan(n)</code>	Returns the tangent of <code>n</code> .
<code>tanh(n)</code>	Returns the hyperbolic tangent of <code>n</code> .
<code>timestamp(format)</code>	Returns the formatted timestamp of the current system time. The value of <code>format</code> is optional. If you do not specify the format, the 13-digit timestamp of the current system time will be returned. Examples: <code>timestamp() = 1543373798943</code> , <code>timestamp('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'') = 2018-11-28T10:56:38Z</code> .

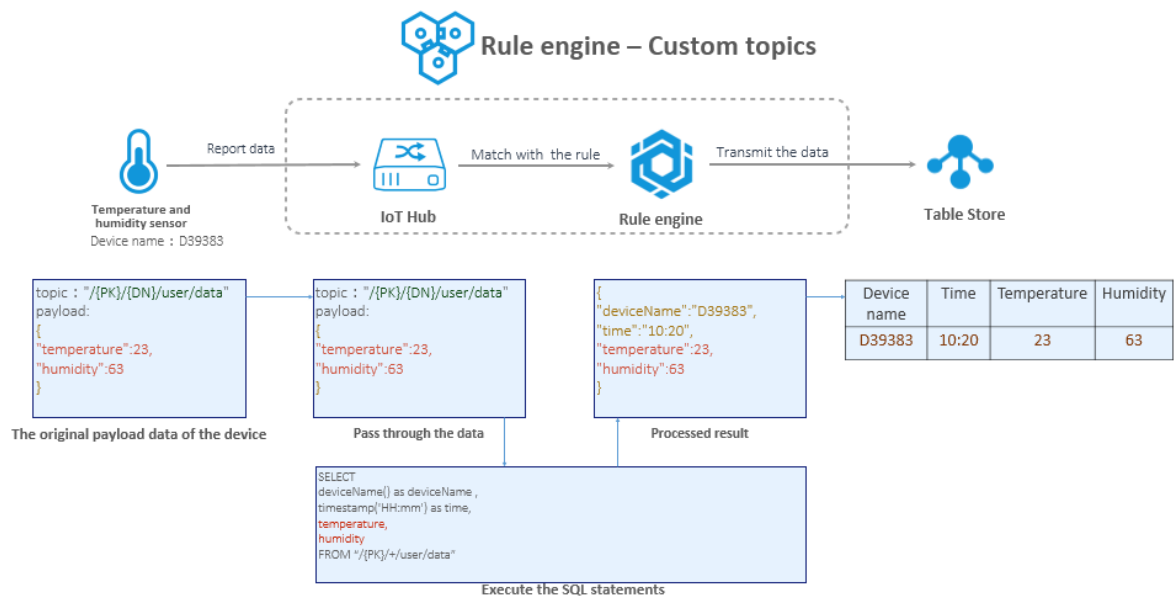
Function name	Description
timestamp_utc(format)	Returns the formatted UTC timestamp of the current system time. The value of format is optional. If you do not specify the format, the 13-digit timestamp of the current system time will be returned. Examples: <code>timestamp_utc()</code> = 1543373798943, <code>timestamp_utc('yyyy-MM-dd\'T\'HH:mm:ss\'Z\')</code> = 2018-11-28T02:56:38Z
topic(number)	Returns a segment of a topic. For example, for topic /abcdef/ghi, if you use the function <code>topic()</code>, “ /abcdef/ghi” will be returned; If you use the function <code>topic(1)</code>, “ abcdef” will be returned; If you use the function <code>topic(2)</code>, “ghi” will be returned.
upper(string)	Returns an upper-case string.
to_base64(*)	If the original payload data is binary data, you can call this function to convert the binary data to a base64String data.
substring(target, start, end)	Returns the part of the target string between the start index (included) and end index (not included). The data type of the target must be String or Integer, and Integer data will be parsed to String data. Examples: • <code>substring('012345', 0) = "012345"</code> • <code>substring('012345', 2) = "2345"</code> • <code>substring('012345', 2.745) = "2345"</code> • <code>substring(123, 2) = "3"</code> • <code>substring('012345', -1) = "012345"</code> • <code>substring(true, 1.2)</code> error • <code>substring('012345', 1, 3) = "12"</code> • <code>substring('012345', -50, 50) = "012345"</code> • <code>substring('012345', 3, 1) = ""</code>

2.1.6 Data forwarding procedure

Data forwarding provided by the rules engine function can only process data that is published to topics. This topic describes the procedure of data forwarding and the formats of the data at different stages during data forwarding.

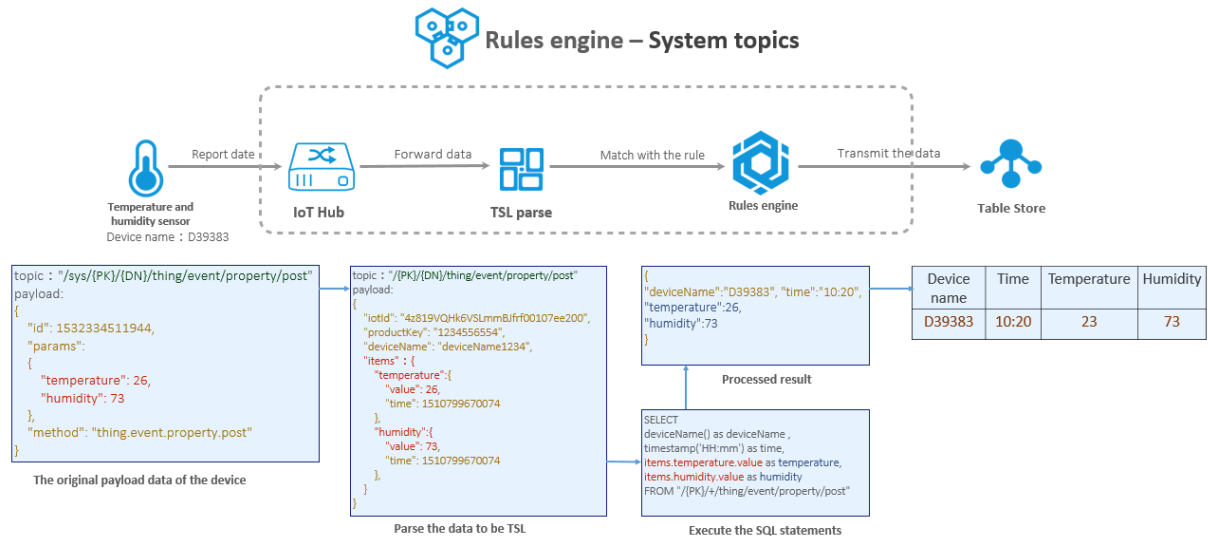
Custom topics

Data published to custom topics is forwarded transparently to the IoT Platform by data forwarding. The structure of the data is not changed. The following figure shows the data forwarding procedure:



System topics

Data published to system topics is in the Alink JSON format. During data forwarding, the data is parsed according to the TSL and then processed by the SQL statements of a rules engine. For more information about the data format, see [Data format](#). The following figure shows the data forwarding procedure:



Note:

During data forwarding, parameter `params` in the payload is replaced by parameter `items` after the data is parsed according to the TSL.

2.1.7 Data format

If you want to use rules engine to forward data, you need to write a SQL statement to process data using message topics. Therefore, the format in which data is stored in these topics must be able to be parsed by SQL statements. For IoT Platform Basic edition topics, the data format is defined manually. For IoT Platform topics, the data format of custom topics is defined manually, and the data format of system topics is pre-defined by the system. For scenarios where the data format is pre-defined, data is strictly processed according to the format. This topic explains the pre-defined data format of system defined topics.

Messages about device properties reported by devices

By using the following topic, you can obtain the device properties reported by devices .

Topic: `/sys/{productKey}/{deviceName}/thing/event/property/post`

Data format:

```

{
  "iotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
  "productKey ":" 1234556554 ",
  "deviceName ":" deviceName 1234 ",
  "gmtCreate ": 1510799670 074 ,

```

```

    " deviceType ":" Ammeter ",
    " items ":{
        " Power ":{
            " value ":" on ",
            " time ": 1510799670  074
        },
        " Position ":{
            " time ": 1510292697  470 ,
            " value ":{
                " latitude ": 39 . 9 ,
                " longitude ": 116 . 38
            }
        }
    }
}

```

Parameter descriptions:

Parameter	Type	Description
iotId	String	The unique identifier of the device.
productKey	String	The unique identifier of the product to which the device belongs.
deviceName	String	The name of the device.
deviceType	String	The node type of the device.
items	Object	Device data.
Power	String	The property name. See the TSL description of the product for all the property names.
Position	String	The property name. See the TSL description of the product for all the property names.
value	Defined in TSL	Property values
time	Long	The time when the property is created. If the device does not report the time, the time when the property is generated on the cloud will be used.
gmtCreate	Long	The time when the message is generated.

Messages about events reported by devices

By using the following topic, you can obtain event information reported by devices.

Topic: / sys /{ productKey }/{ deviceName }/ thing / event /{ tsl . event . identifier }/ post

Data format:

```
{
  " identifier ":" BrokenInfo ",
  " Name ":" Damage rate report ",
  " type ":" info ",
  " iotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
  " productKey ":" X5eCzh6fEH 7 ",
  " deviceName ":" 5gJtxDVeGA kaEztpisjX ",
  " gmtCreate ": 1510799670 074 ,
  " value ":{
    " Power ": " on ",
    " Position ":{
      " latitude ": 39 . 9 ,
      " longitude ": 116 . 38
    }
  },
  " time ": 1510799670 074
}
```

Parameter descriptions:

Parameter	Type	Description
iotId	String	The unique identifier of the device.
productKey	String	The unique identifier of the device product.
deviceName	String	The name of the device.
type	String	Event type. See the TSL of the product for details.
value	Object	Parameters of the event.
Power	String	The parameter name of the event.
Position	String	The parameter name of the event
time	Long	The time when the event is generated. If the device does not report the time, the time recorded on the cloud will be used.
gmtCreate	Long	The time when the message is generated.

Device lifecycle change messages

By using the following topic, you can obtain messages about device creation and deletion, and about devices being enabled and disabled.

Topic: / sys /{ productKey }/{ deviceName }/ thing / lifecycle

Data format:

```
{
```

```

" action " : " create | delete | enable | disable ",
" iotId " : " 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
" productKey " : " X5eCzh6fEH 7 ",
" deviceName " : " 5gJtxDVeGA kaEztpisjX ",
" deviceSecret " : "",
" messageCreateTime ": 1510292739 881
}

```

Parameter descriptions:

Parameter	Type	Description
action	String	- create: Create devices. - delete: Delete devices. - enable: Enable devices. - disable: Disable devices.
iotId	String	The unique identifier of the device.
productKey	String	The unique identifier of the product.
deviceName	String	The name of the device.
deviceSecret	String	The device secret. This parameter is only included when the value of action is create.
messageCreateTime	Integer	The timestamp when the message is generated, in milliseconds.

Device topological relationship update messages

By using the following topic, you can obtain messages about topological relationship creation and removal between sub-devices and gateways.

Topic: `/ sys /{ productKey }/{ deviceName }/ thing / topo / lifecycle`

Data format:

```

{
" action " : " add | remove | enable | disable ",
" gwIotId ": " 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
" gwProductKey ": " 1234556554 ",
" gwDeviceName ": " deviceName 1234 ",
" devices ": [
{
" iotId ": " 4z819VQHk6 VSLmmBJfrf 00107ee201 ",
" productKey ": " 1234556556 9 ",
" deviceName ": " deviceName 1234 "
}
],
" messageCreateTime ": 1510292739 881
}

```

```
}
```

Parameter descriptions:

Parameter	Type	Description
action	String	· add: Add topological relationships. · remove: Delete topological relationships. · enable: Enable topological relationships. · disable: Disable topological relationships.
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The unique identifier of the gateway product.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices whose topological relationship with the gateway will be updated.
iotId	String	The unique identifier of the sub-device.
productKey	String	The unique identifier of the sub-device product.
deviceName	String	The name of the sub-device.
messageCreateTime	Integer	The timestamp when the message is generated, in milliseconds.

Messages about detected sub-devices reported by gateways

In some cases, gateways can detect sub-devices and report their information. By using the following topic, you can obtain the sub-device information reported by gateways.

Topic: `/ sys /{ productKey }/{ deviceName }/ thing / list / found`

Data format:

```
{
  " gwIotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
  " gwProductKey ":" 1234556554 ",
  " gwDeviceName ":" deviceName 1234 ",
  " devices ":[
    {
```

```

        "iotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee201 ",
        "productKey ":" 1234556556 9 ",
        "deviceName ":" deviceName 1234 "
    }
]
}

```

Parameter descriptions:

Parameter	Type	Description
gwIotId	String	The unique identifier of the gateway device.
gwProductKey	String	The unique identifier of the gateway product.
gwDeviceName	String	The name of the gateway device.
devices	Object	The sub-devices that are detected by the gateway.
iotId	String	The unique identifier of the sub-device.
productKey	String	The unique identifier of the sub-device product.
deviceName	String	The name of the sub-device.

Devices return result data to the cloud

By using the following topic, you can obtain request execution results from devices when you send operation requests to devices using an asynchronous method. If an error occurs when sending the request, you will receive an error message from this topic.

Topic: / sys / { productKey } / { deviceName } / thing / downlink / reply / message

Data format:

```

{
    "gmtCreate ": 1510292739 881 ,
    "iotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
    "productKey ":" 1234556554 ",
    "deviceName ":" deviceName 1234 ",
    "requestId ": 1234 ,
    "code ": 200 ,
    "message ":" success ",
    "topic ":" / sys / 1234556554 / deviceName 1234 / thing / service / property / set ",
    "data ":{
    }
}

```

}

Parameter descriptions

Parameter	Type	Description
gmtCreate	Long	The timestamp when the message is generated.
iotId	String	The unique identifier of the device.
productKey	String	The unique identifier of the product.
deviceName	String	The name of the device.
requestId	Long	The request message ID.
code	Integer	The code for the result message.
message	String	The description of the result.
data	Object	The result data reported by the device. For pass-through communication, the result data will be converted by the parsing script.

Response information:

Parameter	Message	Description
200	success	The request is successful.
400	request error	Internal service error.
460	request parameter error	The request parameters are invalid. The device has failed input parameter verification.
429	too many requests	Too many requests in a short time.
9200	device not activated	The device is not activated yet.
9201	device offline	The device is offline now.
403	request forbidden	The request is prohibited because of an overdue bill.

Messages about device status

By using the following topic, you can obtain the online and offline status of devices.

Topic: { productKey }/{ deviceName }/ mqtt / status

Data format:

```
{
  " productKey ":" 1234556554 ",
  " deviceName ":" deviceName 1234 ",
  " gmtCreate ": 1510799670 074 ,
  " deviceType ":" Ammeter ",
  " iotId ":" 4z819VQHk6 VSLmmBJfrf 00107ee200 ",
  " action ":" online | offline ",
  " status ":{
    " value ":" 1 ",
    " time ": 1510292697 471
  }
}
```

Parameter descriptions:

Parameter	Type	Description
iotId	String	The unique identifier of the device.
productKey	String	The unique identifier of the device product.
deviceName	String	The name of the device.
status	Object	The status of the device.
Value	String	1: online; 0: offline.
time	Long	The time when the device got online or offline.
gmtCreate	Long	The time when the message is generated.
action	String	The action of device status change: go online or go offline.

2.1.8 Regions and zones

Before you create a rule to send device data to other Alibaba Cloud products, make sure that the target Alibaba Cloud products have been released in the region of the device and support the format of your data.

Table 2-2: List of supported regions and zones

	China (Shanghai)		Singapore		Japan (Tokyo)		US (Silicon Valley)		Germany (Frankfurt)	
	JSON	Binary	JSON	Binary	JSON	Binary	JSON	Binary	JSON	Binary
Table Store	√	-	√	-	√	-	√	-	√	-

RDS (ApsaraDB for RDS)	√	-	√	-	√	-	√	-	√	-
Message Service	√	√	√	√	√	√	√	√	√	√
Function Compute	√	√	√	√	-	-	-	-	-	-

2.2 Data Forwarding Examples

2.2.1 Forward data to another topic

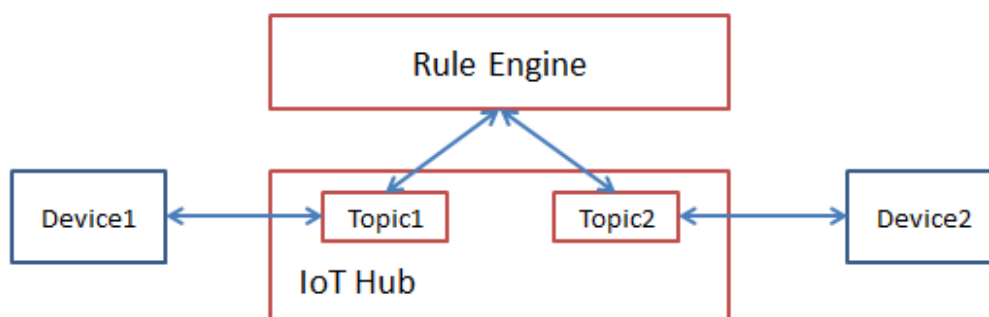
You can forward the data that is processed based on SQL rules to another topic for machine-to-machine (M2M) communication and other applications.

Prerequisites

Before configuring forwarding, follow the instructions in [Create and configure a rule](#) to write a SQL script and filter the data.

Context

The following document describes how to forward data from Topic1 to Topic2 based on the rules engine settings:



Procedure

1. Click Add Operation next to Data Forwarding. The Add Operation page appears.

Add Operation

Select Operation:

Publish to another Topic

* Topic :

/a topic/20190710/airpurifier1/user/update/error

Custom

aircleaner

airpurifier1

user/update/error

OK Cancel

2. Follow the instructions on the page to configure the parameters.

- Select Operation: Select Publish to Another Topic.
- Topic: The topic to which the data is forwarded. You need to complete this topic after selecting a product. You can use the `${ }` expression to quote the context value. For example, `${ dn }/ get` allows you to select the devicename from the message. The suffix of this topic is get.

2.2.2 Forward data to Table Store

You can configure the rules engine data forwarding function to forward data to Table Store.

Prerequisites

Before you configure forwarding, complete the following tasks:

- In the IoT Platform console, create a forwarding rule and write SQL statements for data processing.

For more information, see [设置数据流转规则](#).

In this example, the following SQL statement is defined:

```
SELECT  deviceName  as  deviceName ,  items . PM25 . value  as
        PM25 ,  items .  WorkMode . value  as  WorkMode
FROM    "/ sys / alktuxe ****/ aircleaner  thing / event / property /
post "  WHERE
```

- In the Table Store console, create instances and tables for data receiving and storage.

For more information about Table Store, see [Table Store documentation](#) .

Procedure

1. On the Data Forwarding Rule Details page of the rule, click Add Operation in the Data Forwarding section. In the Add Operation dialog box, select Save to Table Store.



Note:

Binary data cannot be forwarded to Table Store.

Select Operation:

Save to Table Store

This method will save the data to [Table Store](#). For more information, see [Documentation](#)

* Region:

China (Shanghai)

* Instance:

rulesengine

Create Instance

* Data Sheet:

iottestsheet2

Create Table

* Primary Key:

device鍵

\${deviceName}

* Primary Key:

id鍵

AUTO_INCREMENT

* Role:

AliyunIOTAccessingOTSRole

Create RAM Role

OKCancel

2. Set parameters as prompted, and then click OK.

Parameter	Description
Select Operation	Select Save to Table Store.
Region	Select the region of the Table Store instance that receives data.
Instance	Select the Table Store instance that receives data.
Data Sheet	Select the table that receives data..

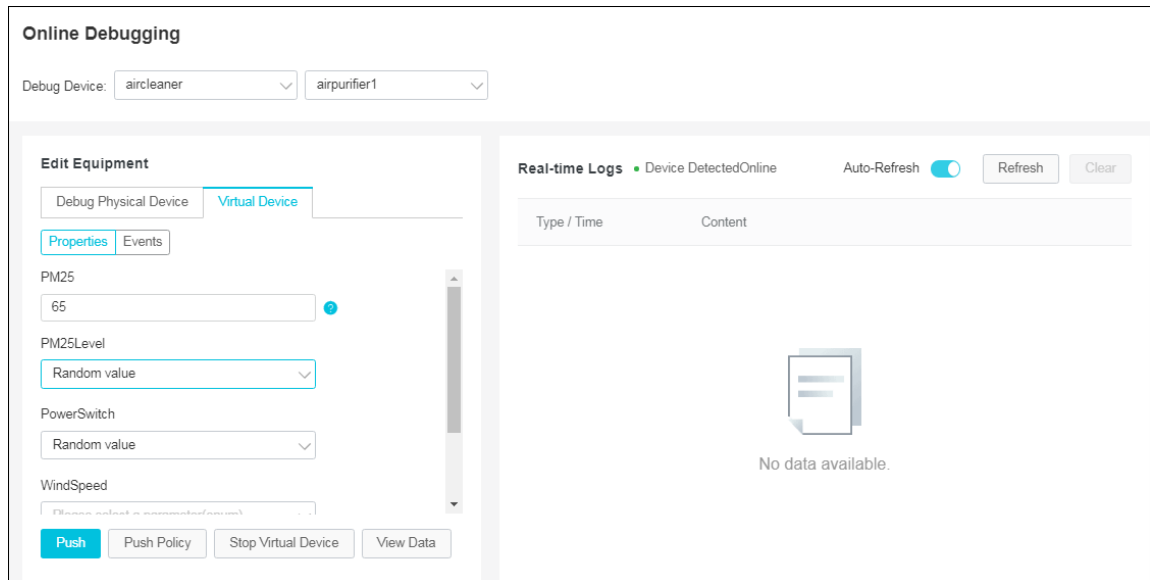
Parameter	Description
Primary Key	To set the value for a primary key of the table, you must use the corresponding field value in the SELECT statement of the forwarding rule. When data is forwarded, this value is saved as the value of the primary key.  Note: - You can set this parameter in the format of <code>\${}</code>. For example, <code>\${deviceName}</code> indicates that the value of the primary key is the value of <code>DeviceName</code> in the message. - If the primary key is an auto-increment column, you do not need to specify the value for the primary key. Table Store automatically generates a value for this primary key column. Therefore, the value of an auto-increment primary key is automatically set to <code>AUTO_INCREMENT</code> and cannot be modified. For more information about auto-increment primary keys, see Auto-increment function of the primary key column.
Role	Authorize IoT Platform to write data to Table Store. You must create a role with Table Store write permissions in the RAM console and assign the role to IoT Platform.

- Return to the Data Forwarding Rules page, and click Start in the Actions column of the corresponding rule.

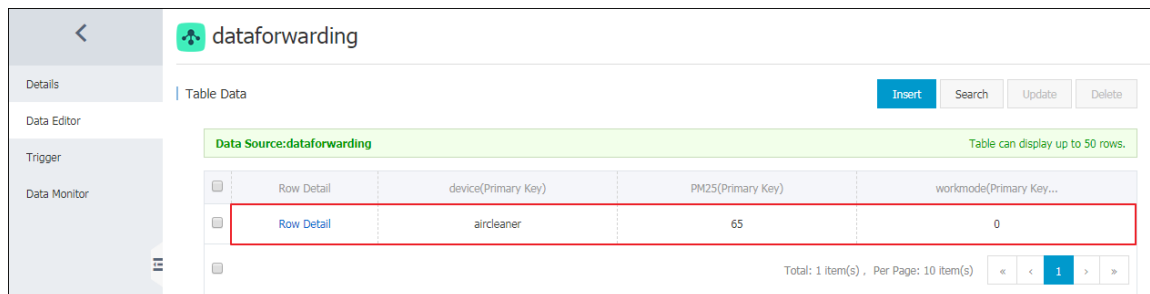
After the rule is started, when a message is published to the topic that is defined in the SQL statement, only the message data defined by the SELECT fields is forwarded to the table in Table Store.

4. Simulate data push to test data flow.

- a) In the left-side navigation pane of the IoT Platform console, choose **Maintenance** > **Online Debug**.
- b) Select the device for debugging, and use a Virtual Device to push analog data to IoT Platform. For more information, see [虚拟设备调试](#).



- c) After the data is pushed, go to the Data Editor page of the target table in the Table Store console to check whether the specified data has been received.



2.2.3 Forward data to ApsaraDB for RDS

You can configure the rules engine to forward processed data to ApsaraDB for RDS instances in VPCs.

Limits

- The ApsaraDB for RDS instances and your IoT Platform service must be in the same region. For example, if your devices are in cn-shanghai region, the data can only be forwarded to RDS instances in the cn-shanghai region.
- Only RDS instances in VPCs are supported.

- Only MySQL instances and SQL Server instances are supported.



Note:

MySQL 8.0 is not supported.

- Databases in classic mode and master mode are supported.
- Binary data cannot be forwarded to ApsaraDB for RDS.

Preparations

- Follow the instructions in [Create and configure a rule](#) to create a rule and write a SQL script for processing data.
- Create an ApsaraDB for RDS instance that is in the same region as your devices, and then create a database and a data table.

Procedure

1. Click Add Operation next to Data Forwarding, and then select Save to RDS.

Select Operation:

Save to RDS ✓

This operation will save the data to RDS. For more information, see [RDS](#). For more information, see [Documentation](#)

Special reminder: This operation is only for RDS instances of proprietary networks, and will add a record to your RDS whitelist for IoT access to your database, do not delete.

Region :

China East 2

* RDS Instance:

rm-uf6073u3cfvw49499 ✓ [Create Instance](#)

* Database

iottest

* Account:

iottest ✓ [Create Account](#)

* Enter password:

* Table Name:

test

* Key :

name

* Value :

\${deviceName}

[Delete](#)

[Add Field](#)

Role:

AliyunIoTAccessingRDSRole ✓ [Create RAM Role](#)

2. Configure the following parameters as prompted:

Parameter	Action
Select Operation	Select Save to RDS.
RDS Instance	Select the VPC RDS instance to which IoT Platform data is to be forwarded.

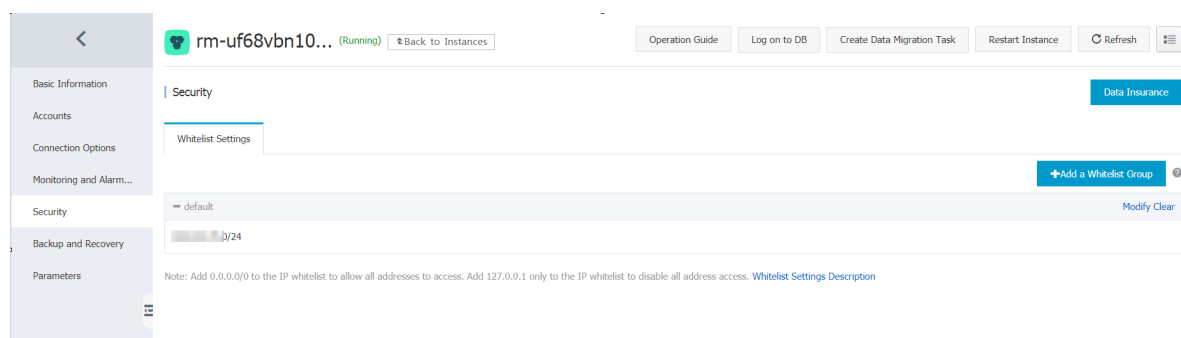
Parameter	Action
Database	Enter the name of the target database.  Note: If your database is in the master mode, you need to manually enter the database name.
Account	Enter the account of the RDS database. The account requires the permissions to read and write data to the database. Otherwise, rules engine cannot write data to the database.  Note: After rules engine obtains the account, rules engine only writes data that matches this rule to the database.
Password	Enter the password to log on to the database.
Table Name	Enter the name of the table that will store data from IoT Platform. Rules engine will then write data to this database table.
Key	Enter a field name of the database table. Rules engine will then write data to this field.
Value	Enter a field of the message that you have defined in the data processing SQL statement. This is the value of Key.  Note: - Make sure that the data type of the Value field is the same as that of the Key field. Otherwise, the data cannot be written into the database. - You can enter a variable, such as <code>\${ deviceName }</code>, to indicate that device names selected from the topic messages are used as the value.
Role	Set the role that authorizes IoT Platform to write data to RDS database table. If you have not created such a role, click Create RAM Role and create a role in the RAM console.

3. In the Rules page, click the Start button corresponding to the rule to start this rule.
4. Once the configuration is complete, the rules engine will add the following IP addresses to the whitelist to connect to RDS. If one or more of the following IP

addresses are not listed, you need to manually add them to the whitelist of the RDS instance:

- China (Shanghai): 100.104.123.0/24
- Singapore: 100.104.106.0/24
- US (Silicon Valley): 100.104.8.0/24
- US (Virginia): 100.104.133.64/26
- Germany (Frankfurt): 100.104.160.192/26
- Japan (Tokyo): 100.104.160.192/26

On the Security page of the RDS console, you can set and view the whitelist.



2.2.4 Forward data to Message Service

By using rules engine to forward data from IoT Platform to [Message Service \(MNS\)](#).

The message transmission performance between devices and servers is improved.

The advantages are described in the following section.

Data forwarding

- Devices send data to application servers

Devices send messages to IoT Platform, where the messages are processed with rules engine and forwarded to specified MNS topics. The application server can then call the relevant APIs of MNS to subscribe to topics for messages from devices

.

One advantage of this method is that using MNS to receive and store messages prevents message packet loss during server downtime. Another advantage is that MNS can process a massive amount of messages simultaneously, which means services remain available even if the server has to process a number of concurrent tasks.

- **Application servers send data to devices**

The application server calls the relevant APIs of IoT Platform to publish messages to IoT Platform, and devices subscribe to related topics for messages from the server.



Mes

Procedure

1. Log on to the [RAM console](#), and create a role with the permission to write messages from IoT Platform into MNS.

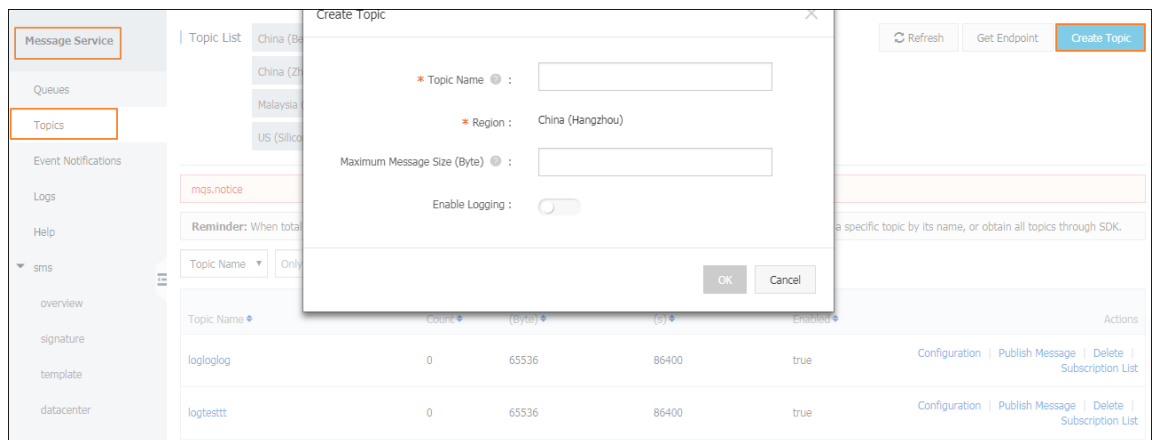
Then, when you are configuring the data forwarding rule in IoT Platform, you can apply this role to allow IoT Platform to write data into MNS. Without applying such a role, IoT Platform cannot forward data to MNS.

For more information about roles, see [RAM role management](#).

2. In the [MNS console](#), create a topic that is to receive messages from IoT Platform.

a. Click Topics > Create Topic.

b. In the Create Topic dialog box, enter a name for the topic, and then click OK.

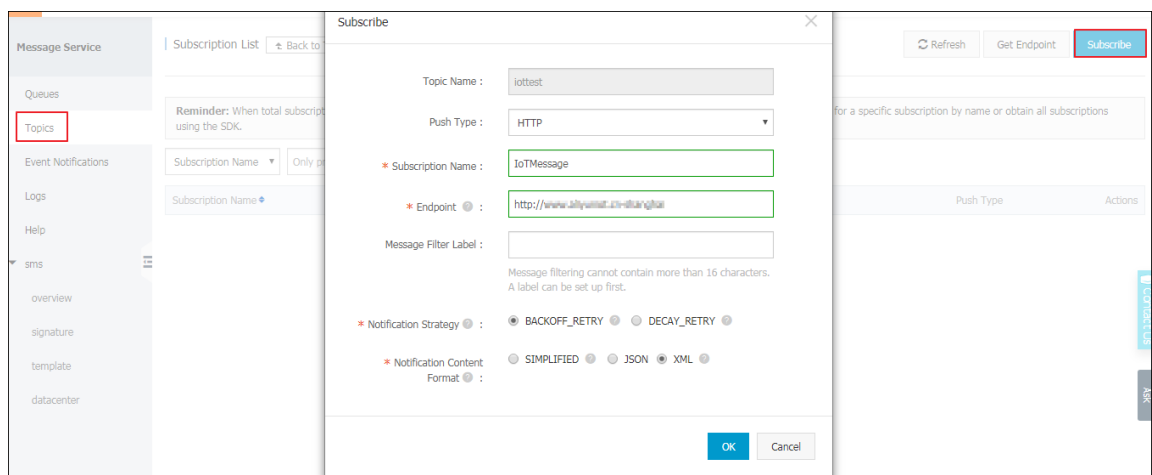


c. On the Topic List page, find the topic and click Subscription List in the Actions column.

d. On the Subscription List page, click Subscribe.

e. Create a subscriber for this topic. A subscriber is a server that subscribes to the topic for messages from IoT Platform.

An MNS topic can have multiple subscribers.



For more information, see the [MNS documentations](#).

3. Go to the [IoT Platform console](#) and, on the Rules page, click Create Rule and then create a rule

4. Go back to the Rules page, find the newly created rule and click Manage on the right.

5. On the Data Flow Details page, write the SQL statement that is used to process and filter messages. For more information, see [Create and configure a rule](#) and [SQL statements](#).
6. On the Data Flow Details page, click Add Operation next to Data Forwarding.

Add Operation

Select Operation:

Send to Message Service

This operation will push the data to Message Service For more information, see [Documentation](#)

* Region:

China (Shanghai)

* Theme:

iottest

[Create Theme](#)

* Role:

AliyunIOTAccessingMNSRole

[Create RAM Role](#)

OK

7. In the Add Operation dialog box, enter information of the MNS topic.

Parameter description:

Parameter	Description
Select Operation	Select the Alibaba Cloud product which will be the data forwarding target. Here, select Send to Message Service.

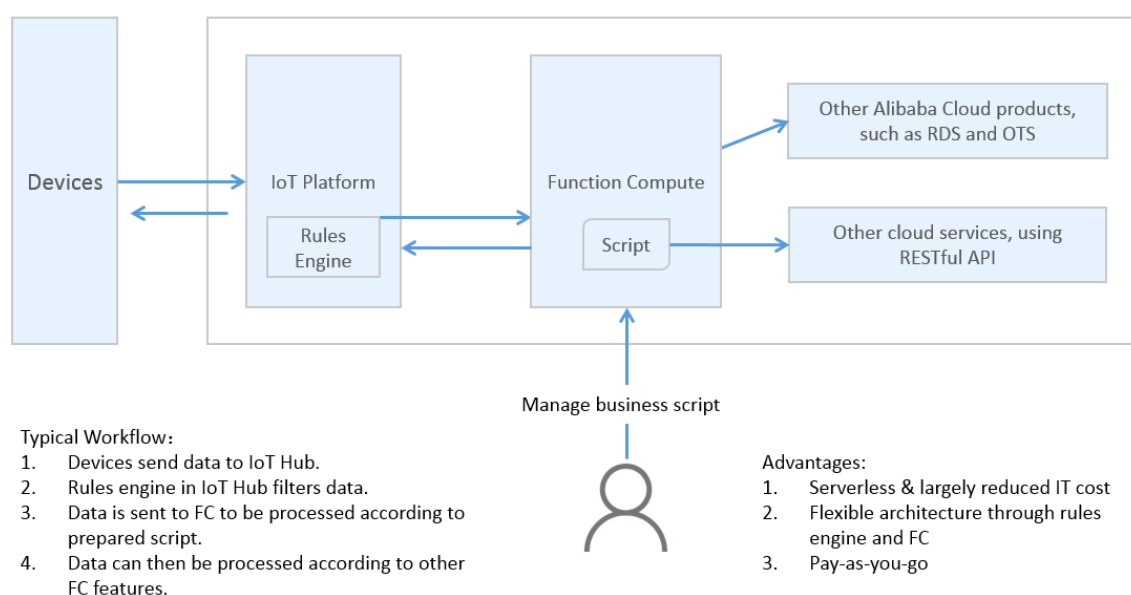
Parameter	Description
Region	Select the region where the MNS topic is.
Theme	Select the MNS topic that is to receive data from IoT Platform.
Role	The role with the permission that IoT Platform can write data into MNS.

8. On the Rules page, click Start corresponding to this rule to run the rule.

Then, IoT Platform can forward messages of the specified IoT Platform topic to the specified MNS topic.

2.2.5 Forward data to Function Compute

Rules engine can forward processed data from IoT Platform to Function Compute (FC).



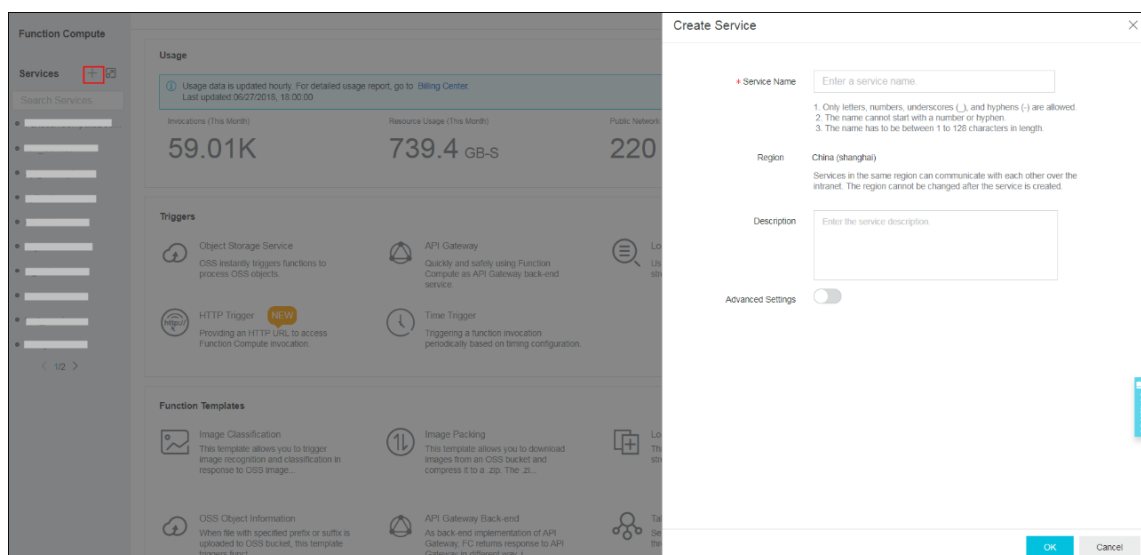
Procedure:

1. On the Function Compute console, create a service and function.
2. Create a rule to send data processed on IoT Platform to FC, and then enable the rule.
3. Send a message to the topic that has rules engine configured.
4. View the function execution statistics on the Function Compute console, or check whether the configuration result is correct based on specific business logic of the function.

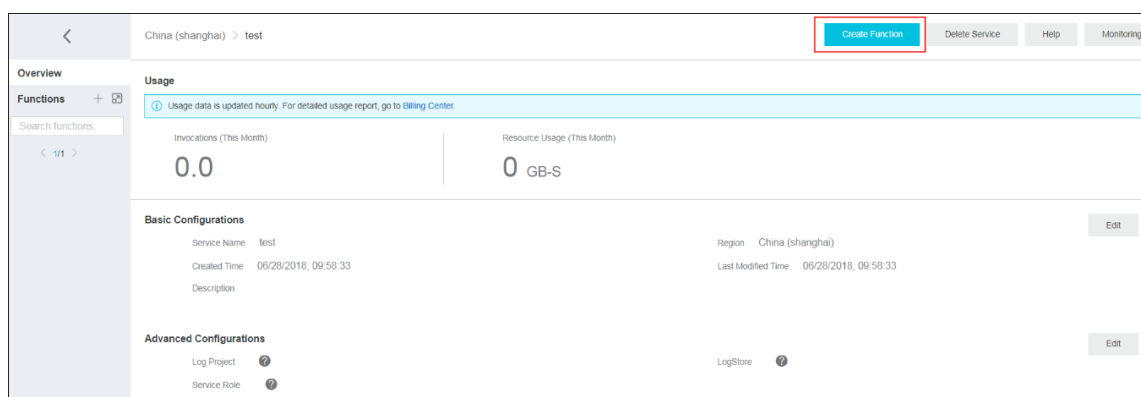
Procedure

1. Log on to the Function Compute console. Create a service and function.

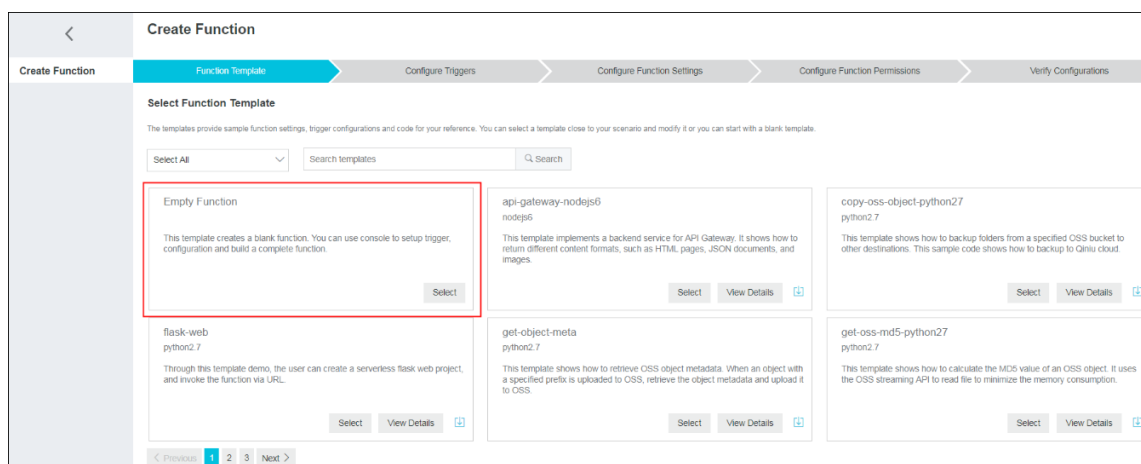
- a. Create a service. Service Name is required. Configure other parameters as required.



- b. After you have created a service, create a function.



- c. Select a function template. A blank template is used as an example.



d. Set parameters for the function.

The function is configured to directly display data on the Function Compute console.

The screenshot shows the 'Create Function' interface in the Function Compute console. It includes sections for 'Function Information', 'Code Configuration', 'Environment Variables', and 'Runtime Environment'. The 'Function Information' section contains fields for 'Service Name', 'Function Name', 'Function Description', and 'Runtime'. The 'Code Configuration' section has radio buttons for 'Function Code' (In-line Edit, Import from OSS, Upload Zip File, Upload Folder) and a 'Select File' button. The 'Environment Variables' section has a table with 'Key' and 'Value' columns. The 'Runtime Environment' section has fields for 'Function Handler', 'Memory', and 'Timeout'. The 'Next' button is located at the bottom right.

In the proceeding parameters,

Service Name: Select the service created in [1.a](#).

Function Name: Specify the name of your function.

Runtime: Configure the running environment for the function, for example, java8.

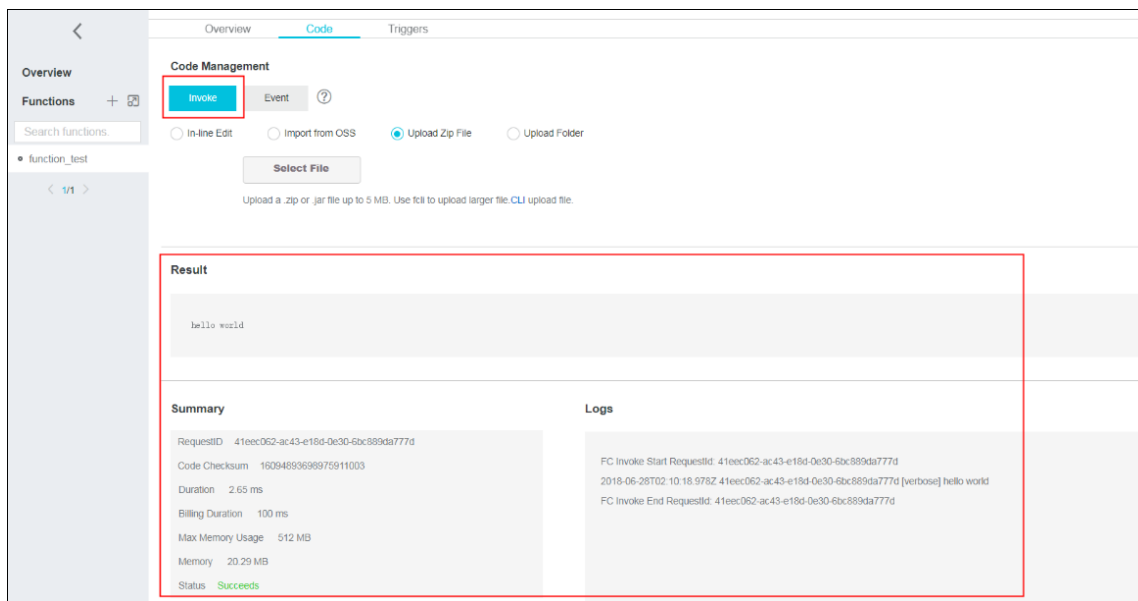
Code Configuration: Upload your code.

Function Handler: Configure the function entry called to run FC. Set it to `com.aliyun.fc.FcDemo::handleRequest`.

Configure other parameters as required. For more information, see configurations in [Function Compute](#).

e. Verify whether the function runs as intended.

After you create a function, you can run it on the Function Compute console for verification. FC will display information about function output and requests on the Function Compute console.



2. Configure rules engine after the function successfully passes the verification.
3. Before you configure rules engine, follow the instructions in [Create and configure a rule](#) to write a SQL script to process the data.

**Note:**

Data in JSON and binary formats can be forwarded to FC.

4. Click a rule name to go to the Rule Details page.

5. Select Data Forwarding Add Operation. On the Add Operation page, configure parameters:

Add Operation

Select operation:

Send to Function Compute

This operation will push the data to [Function Compute](#) For more information, see [Documentation](#)

* Region:

* Service:

test_service

* Function:

function_test

Create Service

* Authorization:

AliyunIOTAccessingFCRole

Create Function

Create RAM Role

OK

Cancel

- **Select Operation:** Select Function Compute.
- **Region:** Select the region that your need to forward data based on your business requirements. If the region does not have any relevant resources, go to Function Compute Console to create resources.



Note:

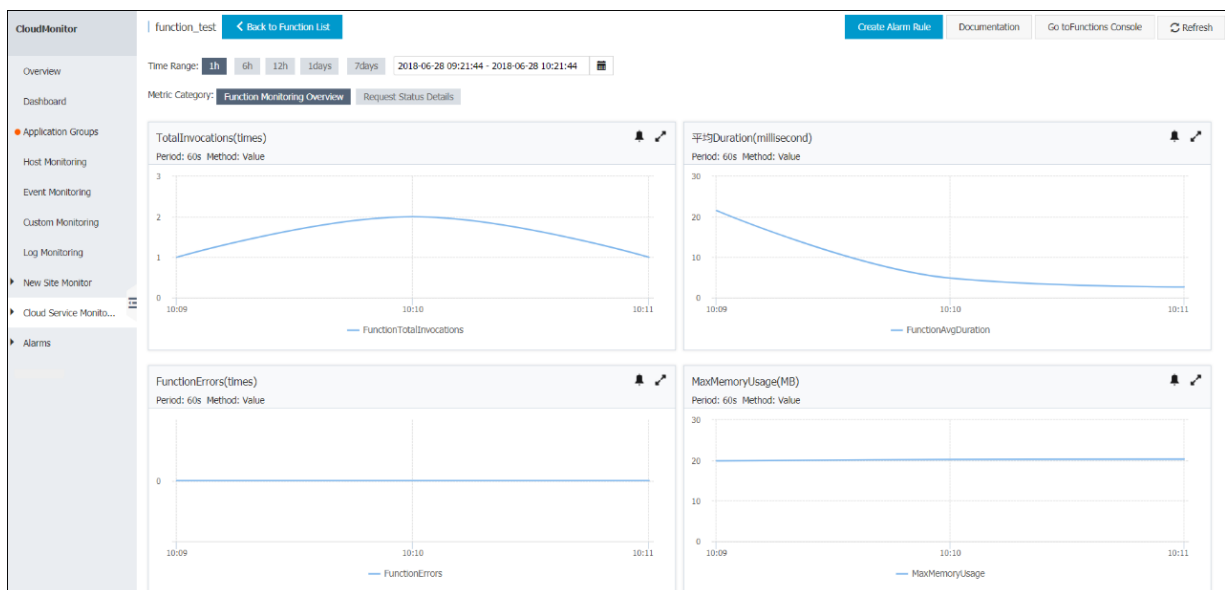
Data forwarding to FC is supported in regions including China (Shanghai), Singapore, and Japan (Tokyo).

- **Service:** Select a service based on your region. If there are no services available, click **Create Service**.
- **Function:** Select a function based on your region. If there are no functions available, click **Create Function**.
- **Authorization:** Specify the role granted IoT Platform the permission to operate functions. You need to create a role with permissions to operate functions before you assign the role to rules engine.

6. **Enable the rule.** After you run the rule, IoT Platform sends the processed data to FC based on the compiled SQL statements. The Function Compute console directly displays the received data based on the defined function logic.

Verify the forwarding result

The Function Compute console collects monitored statistics about function execution . Statistics are delayed for five minutes, after which you can view monitored statistics about function execution on the dashboard.



3 Monitoring and Maintenance

3.1 Real-time monitoring

3.1.1 Real-time monitoring

In the IoT Platform console, the Real-time Monitoring page displays the number of online devices, the number of upstream and downstream messages, and the number of messages that were forwarded by the rules engine. In addition, you can set CloudMonitor alert rules to monitor the resource usage of your IoT Platform and receive alerts.

Display data

To view real-time monitoring data, follow these steps:

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose Maintenance > Real-time Monitoring.
3. Select the products and time range of the data to be viewed.

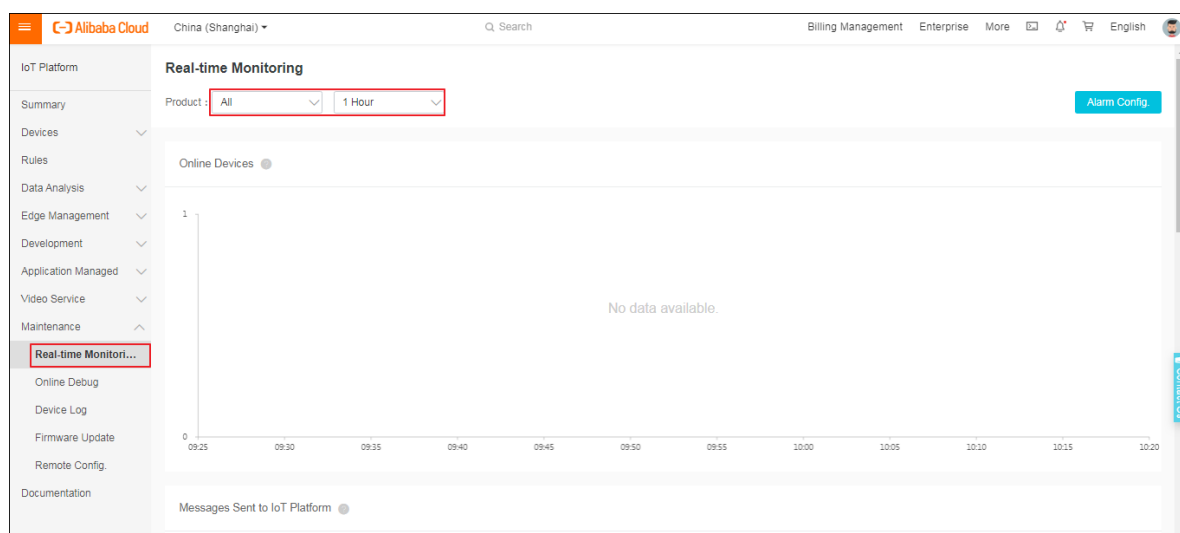


Table 3-1: Time range descriptions

Time range	Description
1 Hour	Displays the statistics of the last hour. Statistics are collected every 1 minute.
1 Day	Displays the statistics of the last 24 hours. Statistics are collected every 5 minutes.

Time range	Description
1 Week	Displays the statistics of the last seven days. Statistics are collected every 15 minutes.

**Note:**

The abscissa values displayed on the Real-time Monitoring page do not represent the collection cycle.

The following table describes the statistical data on the Real-time Monitoring page.

Data	Description
Online Devices	The number of devices that have established persistent connections with IoT Platform. The data is collected with delays. A displayed value is the average value within a collection cycle. The data is displayed based on the protocol that is used to communicate with IoT Platform.
Messages Sent to IoT Platform	The number of messages that the devices send to IoT Platform. The data is collected with delays. A displayed value is the total value within a collection cycle. The data is displayed based on the protocol that is used to communicate with IoT Platform.
Messages Sent from IoT Platform	The number of messages sent from IoT Platform to devices and servers. The data is collected with delays. A displayed value is the total value within a collection cycle. The data is displayed based on the protocol that is used to communicate with IoT Platform.
Messages Forwarded Through Rule Engine	The number of messages forwarded by the rules engine data forwarding function. The data is collected with delays. A displayed value is the total value within a collection cycle. The data is displayed based on the target cloud service to which the messages were forwarded .

Alarm Config.

On the Real-time Monitoring page, click Alarm Config.. The Create Alarm Rule page of the CloudMonitor console appears. You can also directly access the [Create Alarm Rule](#) page to create an alert.

You can create threshold-triggered and event-triggered alert rules in the CloudMonitor console.

- Create a threshold-triggered alert rule

IoT Platform allows you to use CloudMonitor to monitor IoT Platform by multiple metrics. These metrics include the number of real-time online devices using a specific communication protocol, the number of messages sent from devices to IoT Platform, the number of messages sent from IoT Platform to devices, the number of messages forwarded by the rules engine to other Alibaba Cloud service

, the number of property report failures, the number of event report failures, the number of service call failures, and the number of property setting failures.

On the Create Alarm Rule page, configure the parameters and then click Confirm.

Create Alarm Rule
Back to

1
Related Resource

Product: IoT Platform
Resource Range: Instance
Region: China East 2 (Shanghai)
Instance: Shared Instance
product: aircleaner1

2
Set Alarm Rules

Alarm Rule: testrule
Rule Description: DeviceEventReportError
1Minute
1 periods
Number
>=
Threshold
+Add Alarm Rule
Mute for: 24 h
Effective Period: 00:00 To: 23:59

3
Notification Method

Table 3-2: Alert rule parameters

Parameter	Description
Product	Select IoT Platform.
Resource Range	Includes the following value options: - All Resources: An alert is sent when any instance under your IoT Platform service meets the description of the alert rule. - Instance: An alert is sent only when the specified products meet the description of the alert rule.
Region	This parameter is available only when you set Resource Range to Instance. Select the region of the IoT Platform instance monitored by this alert rule.

Parameter	Description
Instance	Select an IoT Platform instance to be monitored and select one or more products.
Alarm Rule	Set the name of the alert rule.
Rule Description	Set the description of the alert rule. It defines the condition in which an alert is triggered. You must configure the following items: - Select a monitoring metric for the rule. - Select a scan period for the rule. For example, if the scan period is set to 60 minutes, scans are performed every 60 minutes. - Set the triggering condition. For example, an alert is triggered only if the number of devices exceeds 5,000 for three consecutive scan periods.
Mute for	Set the period of time before which the alert is sent again if the exception persists after the alert is triggered.
Effective Period	Set the time range when the alert rule is applied. CloudMonitor applies the alert rule to monitor the specified metric only during the specified effective period.
Notification Method	Set notification parameters, such as the notification contacts and notification methods.

For more information about setting threshold-triggered alert rules, see [Procedure](#).

- Create an event-triggered alert rule

You can use an event-triggered alert rule to monitor the following IoT Platform events:

- The upstream QPS of any device reaches the upper limit.
- The downstream QPS of any device reaches the upper limit.
- The number of connection requests per second for the current account reaches the upper limit.
- The upstream QPS for the current account reaches the upper limit.
- The downstream QPS for the current account reaches the upper limit.
- The engines rule data forwarding QPS for the current account reaches the upper limit.

Go to the [Alarm Rules](#) page of the CloudMonitor console, and choose Event Alarm > Create Event Alert.

The following figure shows how to create an event-triggered alert rule.

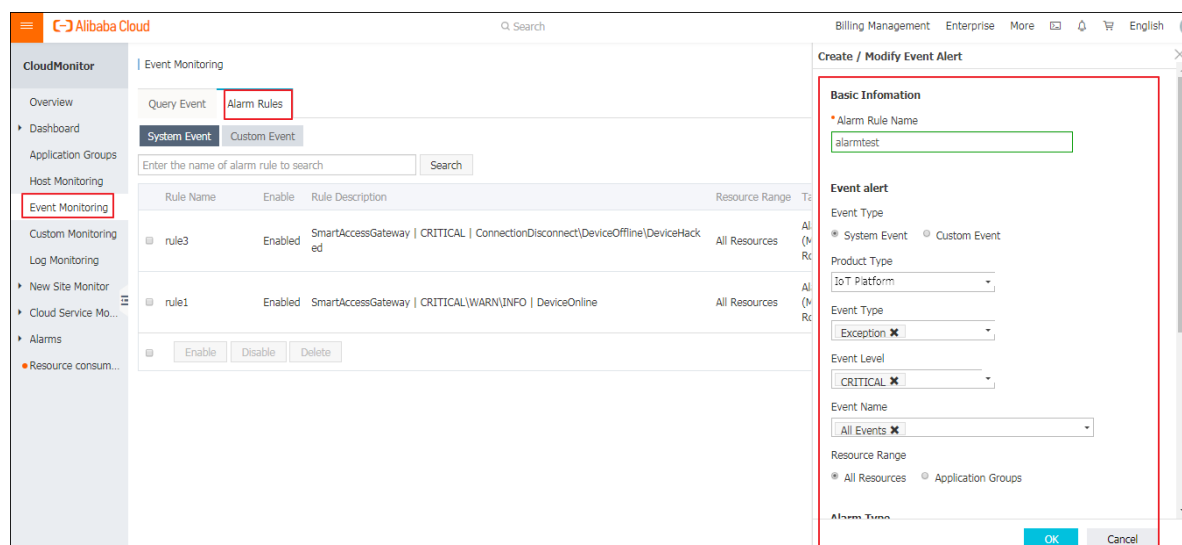


Table 3-3: Event-triggered alert rule parameters

Parameter	Description
Alarm Rule Name	Set the name of the alert rule.
Event Type	Select System Event.
Product Type	Select IoT Platform.
Event Type	Select All types or Exception.
Event Level	Select All Levels or select one or more specific event levels.

Parameter	Description
Event Name	Select one or more events to be monitored.
Resource Range	Select All Resources.
Alarm Type	Set the alert contacts and notification methods.

For more information about setting event-triggered alert rules, see [Create an event alert rule](#) in the CloudMonitor documentation.

3.2 Online debug

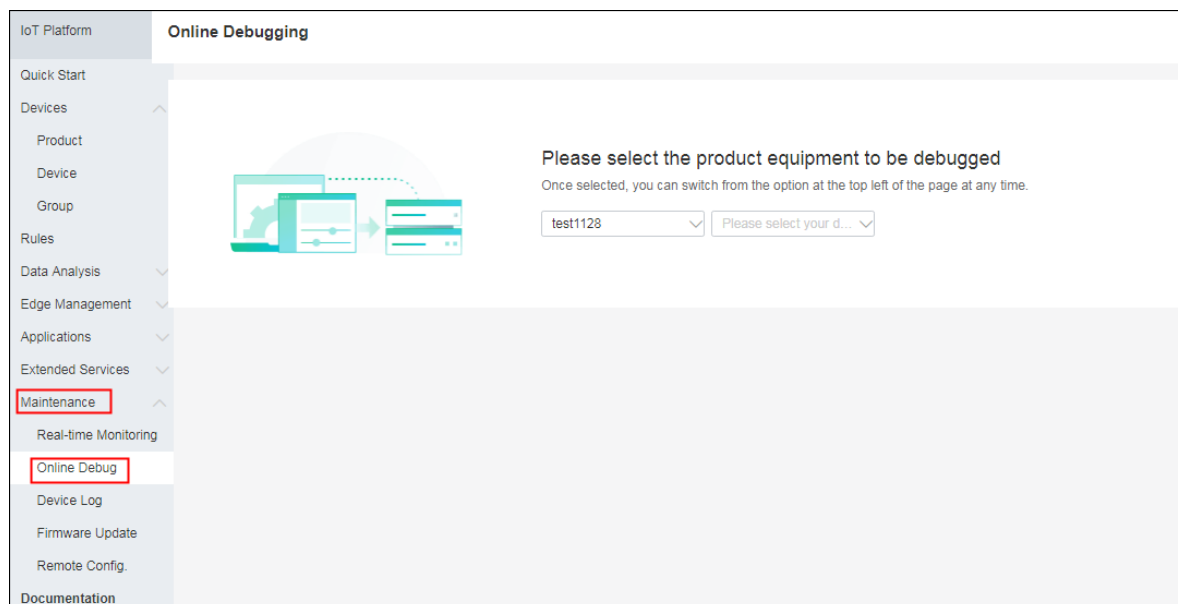
3.2.1 Debug applications using Physical Devices

After you complete the device configuration, you can use the online debugging function in the IoT Platform console to test and debug the applications.

Procedure

1. Log on to the IoT Platform console and then, in the left-side navigation pane, click **Maintenance > Online Debug**.
2. On the Online Debugging page, select the device to be debugged.

After you select a device, you are automatically directed to the debugging page.



3. Select Debug Physical Device.

4. Select the feature that you want to test.

- If you select a property, you must select Set or Get as the operation method. .
- If you select an event, select Get as the operation method.



Note:

If you have not defined properties or events for the product, go to the Product Details page to define features for the product.

5. Dispatch the command.

- Set a property: Enter a property in the format of `{" YourPropertyIdentifier ": Value }`, and then click Dispatch Command. You can then see the operation result from the device log.
- Get a property: Click Dispatch Command. Then, the latest property information reported by the device is displayed in the box.
- Call a service: Enter an input parameter in the format of `{" YourServiceInputParameter ": Value }`, and then click Dispatch Command. You can then see the operation result from the Real-time Logs.
- Get an event: Click Dispatch Command. Then, the latest event information reported by the device is displayed in the box.

3.2.2 Debug applications using virtual devices

IoT Platform provides virtual devices to help developers debug applications.

Context

A typical IoT development process is as follows: a device SDK is developed, the devices report data to IoT Platform, and the developers use the data to develop applications. However, this development process is time consuming. To resolve this issue, IoT Platform provides virtual devices that simulate the physical devices connecting to IoT Platform and reporting defined properties and events. You can then use the data reported by the virtual devices to debug your applications. After the physical devices connect to IoT Platform, the corresponding virtual devices will automatically become inactive.

Limits:

- The minimum time interval for pushing data is 1 second.
- The maximum number of messages that can be pushed at a specific interval is 1,000.
- The maximum number of times you can use the Push method per day is 100.

Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose Maintenance > Online Debug
3. On the Online Debugging page, select the device to be debugged.

After you select a device, you are automatically directed to the debugging page.

4. Choose Virtual Device > Start Virtual Device.

**Note:**

If the physical device is active or disabled, you cannot start the corresponding virtual device.

5. Set the content for the simulated push.

- If the device data type is Alink JSON, you can enter values of properties and events.

For a property value, you can enter a value that complies with the data type and the value range of the property, or you can enter the function `random()` to generate a random value.

The following example shows the Properties page of a device, where the value 220 is entered for `Voltage`.

Online Debugging

Debug Device: donotparse | parsetest

Edit Equipment

Debug Physical Device | Virtual Device

Attribute configuration | Event report

voltage

220

longitude

Enter a parameter(double)

latitude

Enter a parameter(double)

altitude

Enter a parameter(double)

CoordinateSystem

Please select a parameter(en...)

Push | Push Policy | Stop Virtual Device

View Data

Real-time Logs • Device DetectedOnline

Auto-Refresh ☒ Refresh Clear

Type / Time	Content
No data available.	

- If the device data type is Do not parse/Custom, you can enter a Base64 string. The length of string cannot exceeds 4096 characters.

Online Debugging

Select device: 9S2PzrX | F9C9d09S2PzrXKaqCtIZ

Debug Device

Debug Physical Device | Virtual Device

Properties | Events

Enter a parameter(text)

Push | Push Policy | Stop Virtual Device

View Data

Real-time Logs • Device DetectedOnline

Auto-Refresh ☒ Refresh Clear

Type / Time	Content
No data available.	

6. Select a data push method.

- **Push:** Push the data immediately.
- **Push Policy:**
 - **At Specific Time:** Push the data at your specified time.
 - **At Specific Interval:** Push the data regularly at your specified time interval in your specified time range. The unit of time interval is seconds.

Result

After the push operation is executed, the operation log is displayed on the Real-time Logs tab page.

After the data is pushed, click View Data to view the device details page. On the Status tab page, you can view property information that has been pushed, and on the Events tab page you can view event information that has been pushed.



Note:

If you have set a Push Policy, the data will be pushed according to the policy. After the data has been pushed, the operation log, property information, or event information will be displayed on the corresponding page.

3.3 Device log

IoT Platform provides the log service function. You can query device log entries on the Device Log page in the IoT Platform console. This topic describes the error codes in the device log entries and the corresponding troubleshooting methods.

Query device log entries

Log entries can be divided into the following types:

- [Device activity analysis](#)
- [Upstream analysis](#)
- [Downstream analysis](#)
- [TSL data analysis](#)

To query device log entries, follow these steps:

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose Maintenance > Device Log.

3. Select a product, log type, set filters such as device name, and then click Search.

The following filters are available:

Filter	Description
Device name	Enter a device name. You can search the log entries of the device with the specified name.
Keywords	Enter keywords to search for the specified log entries.
Message ID	Enter a message ID. A Message ID is a unique identifier generated by IoT Platform for a message. You can query the forwarding status and content of a message based on its message ID.
Status	Select a status to search for specific log entries. Valid options: · All · Successful · Failed
Time range	Select a time range.



Note:

Among all status codes of log entries, a value of 200 indicates that the request is successful, and other values indicate that the request has failed. For more information about the failed status codes, see the following sections.

Device activity analysis logs

Device activities can generate device connection and disconnection logs.

Error code	Description	Cause	Troubleshooting
400	A request error occurs	This error may be caused by one of the following: - The device is disconnected because another device is connected to IoT Platform using the same device certificate. IoT Platform identifies a device based only on the device certificate information (productKey, deviceName, and deviceSecret). Possible causes include: - The same device certificate is installed on multiple devices. - The network or power supply of the device is unstable . The device is reconnected to IoT Platform immediately after an abrupt network outage or power failure. In this case, IoT Platform identifies the device that is reconnected as a new device. - The device has been deleted from IoT Platform. - The device has been disabled on IoT Platform.	- Go to the corresponding Device Details page in the IoT Platform console to view the time next to activated At, that is the activation time. Then, determine whether another device uses the same device certificate to connect to IoT Platform based on the activation time. - In the Device List section of the Devices page, search for the device to check whether the device is deleted. - In the IoT Platform console, check whether the status of the corresponding device is displayed as Disabled.

Upstream and downstream analysis logs

- Upstream message logs are generated when any of the following occasions occurs: a device publishes messages to topics, messages are forwarded to the rules engine, and the rules engine data forwarding function forwards messages to other topics or other Alibaba Cloud services.
- Downstream message logs are generated when messages are sent to devices from the cloud.

Error code	Description	Cause	Troubleshooting
1901	The message fails to be sent due to poor network conditions, such as the congestion of the TCP write buffer.	The data channel between the device and the server is blocked. The block may be due to the slow network transmission speeds, or because the device cannot handle any more messages.	Check network conditions and device message consumption capabilities.
1902	When the message is transmitted over the network, an exception occurs.	The sending failure is caused by a network exception.	Check network conditions.
1903	The format of the topic is invalid.	The format of the message topic is invalid.	Check the topic format.
1904	IoT Platform receives an invalid RRPC response.	The RRPC response received by IoT Platform does not have the corresponding RRPC request. This error may occur if the request times out.	Check the RRPC response from the device to determine whether the RRPC request has timed out.

Error code	Description	Cause	Troubleshooting
1905	IoT Platform does not receive any RRPC response before the timeout timer expires.	After IoT Platform sends a RRPC request to the device , IoT Platform does not receive any RRPC response from the device before the timeout timer expires.	View the log entry on the device to check whether the RRPC request received by the device has been responded.
1950	When the message is transmitted over the network, a network connection exception occurs.	The sending failure is caused by a network exception.	Check the network status.
1951	Unknown response type.	The device sends an unknown message to IoT Platform.	Check the type of the message that is sent by the device. If you are using the Alibaba Cloud device SDK , contact Customer Service or submit a ticket.
9200	The device is inactive.	The device is not activated on IoT Platform. After a new device is registered, the device is activated only after it is connected to IoT Platform and reports data to IoT Platform.	Check the status of the device in the IoT Platform console.
9201	The device is disconnected.	The device is disconnected from IoT Platform.	Check the status of the device in the IoT Platform console.

Error code	Description	Cause	Troubleshooting
9236	Topic authentication fails.	The permission of the topic that is used to publish or subscribe to messages does not match.	Go to the Topic List tab page of the device in the IoT Platform console, and make sure that the topic permission is correct. The topic used to publish messages must be granted the Publish permission. The topic used to subscribe to messages must be granted the Subscribe permission.
9324	A throttling error occurs.	The requests from the device or the tenant are too many.	Reduce the frequency of message sending, or contact Customer Service.
9321	The parameters are invalid.	The input request parameters are invalid.	Check the corresponding parameter settings as prompted.
9320	The payload is invalid.	The format of the payload sent by the device is invalid.	Check whether the payload format is standard.
9331	An internal error occurs with the destination cloud service.	An internal error occurs with the cloud service for which the message is destined.	Based on the error code in the log entry, go to the official website of the corresponding cloud service to troubleshoot the error or contact Customer Service.
9332	The cloud service configuration is invalid.	You specify an invalid forwarding destination configuration when you configure message forwarding. As a result, an error occurs when IoT Platform connects to the destination cloud service.	View the data forwarding rule to check whether the configuration of the data destination is correct and whether the resource exists. Based on the error code in the log entry, go to the official website of the corresponding cloud service to troubleshoot the error.

Error code	Description	Cause	Troubleshooting
9333	A cloud service authorization error occurs.	IoT Platform may be granted incorrect permissions to access the destination cloud service.	Check your Alibaba Cloud RAM authorization policy.
9399	An unknown internal server error occurs.	IoT Platform has an internal error.	Contact Customer Service or submit a ticket.

TSL data analysis logs

TSL data analysis logs are generated for the following operations: property or event reporting, property setting, service calling, and responding to property or service calls.

If the data format is Do not parse/Custom, in addition to the log content, the hexadecimal raw data is also displayed.

Log format description

Parameter	Description
id	The ID of the Alink protocol message. This ID is used to identify messages exchanged between the device and IoT Platform.
params	The request parameters.
code	The returned result code.
method	The request method.
type	The message type. Valid values include upstream and downstream.
scriptData	The input and output parameters in the data parsing when the data format is Do not parse/Custom.
downOriginalData	The original downstream Alink JSON data that requires parsing when the data format is Do not parse/Custom.
downTransformedData	The downstream data after parsing when the data format is Do not parse/Custom.
upOriginalData	The original upstream data that requires parsing when the data format is Do not parse/Custom.

Parameter	Description
upTransfor medData	The upstream Alink JSON data after parsing when the data format is Do not parse/Custom.

Error codes about service call and property setting failures

When a service is called, IoT Platform checks whether the input parameters of the service comply with the definition of the service in the TSL model.

Error code	Description	Cause	Troubleshooting
9201	The device is disconnected.	The device is disconnected from IoT Platform.	Check the device status in the IoT Platform console.
9200	The device is not activated.	The device is not activated on IoT Platform. A newly registered device must report data to IoT Platform to be activated.	Check the device status in the IoT Platform console.
6208	The device has been disabled.	After a device is disabled, you cannot set the properties or call the services.	Check the device status in the IoT Platform console. If the device is disabled, enable the device and then try the operation again.
6300	The method parameter is not found when the system verifies the input parameters based on the TSL model.	The method parameter, which is required by the Alink protocol, is not found in the Alink (standard) data reported by the device or in the parsed data of the custom (do not parse) data reported by the device.	View the property reporting log entry in the IoT Platform console and check the reported data. You can also view the log entry on the device to check the reported data.

Error code	Description	Cause	Troubleshooting
6206	An error occurs when IoT Platform queries the service definition.	The service is not found.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Check whether the service is defined in the TSL model . If the service is defined , verify that the input parameters of the service are the same as those defined in the TSL model.
6200	The script does not exist.	If the data format of the device is Do not parse/ Custom, the script will be used to parse the data when IoT Platform calls the service. This error code is returned if you do not define a parsing script.	Go to the corresponding Product Details page in the IoT Platform console, and check whether the parsing script exists. If the parsing script exists, resubmit the script and then try the operation again.
6201	The parsing result is empty.	The parsing script runs correctly, but returns an empty result. For example, the response of rawDataToProtocol is null, or the response of protocolToRawData is null or empty.	Check the script content to identify the cause.

Error code	Description	Cause	Troubleshooting
6207	The data format is incorrect.	This error may occur when IoT Platform calls the service synchronously or when the device reports data. When IoT Platform calls the service synchronously, possible causes include the following: · The format of the data returned by the device is incorrect. · The parsed data format is incorrect if the data format is Do not parse/ Custom. · The data format of the service is incorrect.	To view the valid data format required by the service, see API documentation and the TSL model. To view the corresponding Alink JSON format, see Alink protocol documentation .
Error codes about system exceptions			
5159	An error occurs when the system obtains the property information from the TSL model.	A system exception occurs.	Submit a ticket in the console.
5160	An error occurs when the system obtains the event information from the TSL model.		

Error code	Description	Cause	Troubleshooting
5161	An error occurs when the system obtains the service information from the TSL model.		
6661	An error occurs when the system queries the tenant information.		
6205	An error occurs when IoT Platform calls the service.		

Error codes about property and event report failures

When a device is reporting a property or an event, the property or the event will be verified based on the TSL model of the device.

Error code	Description	Cause	Troubleshooting
6106	The number of the reported properties exceeds the upper limit.	A device can report up to 200 properties at one time.	View the property reporting log entry in the IoT Platform console and check the number of the reported properties. You can also view the log entry on the device to check the number of the reported properties.

Error code	Description	Cause	Troubleshooting
6300	The method parameter is not found when the system verifies the input parameters based on the TSL model.	The method parameter , which is required by the Alink protocol, is not found in the Alink (standard) data reported by the device or in the parsed data of the custom (do not parse) data reported by the device.	View the property reporting log entry in the IoT Platform console and check the reported data . You can also view the log entry on the device to check the reported data.
6320	The property information is not found when the system verifies the input parameters based on the TSL model.	The specified property is not found in the TSL model of the device.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Check whether the specified property is defined in the TSL model. If the property is not defined, define the property.
6450	The method does not exist in the Alink JSON formatted data.	The method parameter is not found in the Alink standard data reported by the device or in the parsed data of the custom (do not parse) data reported by the device.	View the property reporting log entry in the IoT Platform console and check whether the method parameter is included in the data reported by the device. You can also view local device logs.

Error code	Description	Cause	Troubleshooting
6207	The data format is incorrect.	This error may occur when IoT Platform calls the service synchronously or when the device reports data. When the device reports data, the possible cause is that the Alink data reported by the device or the returned data after parsing is not in the JSON format.	See Alink protocol documentation to view the valid data format, and then report data accordingly.
Error codes about system exceptions			
6452	A throttling error occurs.	Traffic throttling is triggered because too many requests are submitted.	Submit a ticket in the console.
6760	The storage quota of the tenant is exceeded.	A system exception occurs.	Submit a ticket in the console.

Error codes about response failures to service calls and property settings

Error code	Description	Cause	Troubleshooting
Common error codes			
460	The parameters are invalid.	The request parameters are invalid.	Submit a ticket in the console.
500	An internal system error occurs.	An unknown error occurs in the system.	Submit a ticket in the console.
400	A service request error occurs.	An unknown error occurs when IoT Platform calls the service.	Submit a ticket in the console.

Error code	Description	Cause	Troubleshooting
429	Too many requests are submitted in a short period of time.	Traffic throttling is triggered because too many requests are submitted in a short period of time.	Submit a ticket in the console.
Error codes about system exceptions			
6452	A throttling error occurs.	Traffic throttling is triggered because too many requests are submitted.  Note: This error code may be returned if the data format of the device is Alink JSON.	Submit a ticket in the console.

Common error codes about TSL models

When a service is being called or a device is reporting a property or an event, the input parameters of the service, the property, or the event will be verified based on the TSL model of the device.

Error code	Description	Cause	Troubleshooting
6321	The identifier of the property is not found in the TSL model.	A system exception occurs.	Submit a ticket in the console.
6317	The TSL model is incorrect.	A system exception occurs.	Submit a ticket in the console.

Error code	Description	Cause	Troubleshooting
6302	The parameters are not found.	When the system verifies the input parameters of the service, the required parameters are not found in the request.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Check the input parameters of the service in the TSL model and make sure that you have passed in all required parameters.
6306	The input parameter does not comply with the integer data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: · The data type of the parameter is different from the data type defined in the TSL model. · The parameter value is not in the range defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the parameter value is in the value range defined in the TSL model.
6307	The input parameter does not comply with the 32-bit float data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: · The data type of the parameter is different from the data type defined in the TSL model. · The parameter value is not in the range defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the parameter value is in the value range defined in the TSL model.

Error code	Description	Cause	Troubleshooting
6322	The input parameter does not comply with the 64-bit float data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: - The data type of the parameter is different from the data type defined in the TSL model. - The parameter value is not in the range defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the parameter value is in the value range defined in the TSL model.
6308	The input parameter does not comply with the Boolean data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: - The data type of the parameter is different from the data type defined in the TSL model. - The parameter value is not in the range defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the parameter value is in the value range defined in the TSL model.
6309	The input parameter does not comply with the enum data specifications defined in the TSL model.	The data type of the parameter is different from the data type defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model.

Error code	Description	Cause	Troubleshooting
6310	The input parameter does not comply with the text data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: · The data type of the parameter is different from the data type defined in the TSL model. · The length of the parameter exceeds the upper limit defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the parameter length does not exceed the upper limit.
6311	The input parameter does not comply with the date data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: · The data type of the parameter is different from the data type defined in the TSL model. · The input data is not a UTC timestamp.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the input data is a UTC timestamp.
6312	The input parameter does not comply with the struct data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: · The data type of the parameter is different from the data type defined in the TSL model. · The number of the parameters contained in a struct is different from the number defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is the same as the data type defined in the TSL model and that the number of the parameters contained in a struct is the same as the number defined in the TSL model.

Error code	Description	Cause	Troubleshooting
6304	The input parameter is not found in the struct defined in the TSL model.	The input parameter is not found in the struct defined in the TSL model.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Check the input parameters with the TSL model for inconsistencies.
6324	The input parameter does not comply with the array data specifications defined in the TSL model.	When the system verifies a parameter based on the TSL model, the following errors may occur: • The elements in the passed-in array do not match the array definition in the TSL model. • The number of elements in the array exceeds the maximum number defined in the TSL model.	• Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Check the definition of the corresponding array for inconsistencies. • View the upstream log entry in the IoT Platform console to check the number of elements in the array data reported by the device.
6328	The value of the input parameter is not an array data.	The value of the input parameter is not an array data.	Go to the corresponding Product Details page in the IoT Platform console, and view the TSL model. Make sure that the data type of the input parameter is array.
6325	The element type in the array data is not supported by IoT Platform.	The element type is not supported. Only the following element types can be included in an array: int32, float, double, text, and struct.	Make sure that the element type is supported by IoT Platform.
Error codes about system exceptions			

Error code	Description	Cause	Troubleshooting
6318	A system exception occurs when the system parses the TSL model.	A system exception occurs.	Submit a ticket in the console.
6329	An error occurs when the system parses the array data specifications in the TSL model.		
6323	The data specifications defined in the TSL model are incorrectly formatted.		
6316	An error occurs when the system parses the parameters in the TSL model.		
6314	The data type in the TSL model is not supported.		

Error code	Description	Cause	Troubleshooting
6301	An error occurs when the system verifies the data format of the input parameters based on the TSL model.		
Error codes about data parsing scripts			
26010	Traffic throttling is triggered because too many requests are submitted.	Too many requests are submitted in a short period of time.	Submit a ticket in the console.
26001	The content of the parsing script is empty.	The parsing script content is not found.	Go to the corresponding Product Details page in the IoT Platform console, and check your data parsing script. Make sure that the script is saved and submitted. The parsing script cannot be a draft.
26002	An exception occurs when you run the script.	The script runs correctly, but the script content is incorrect. For example, the script contains syntax errors.	Log on to the IoT Platform console, use the same parameters to run the script for debugging, and then modify the script accordingly. The console only provides a basic script running environment. It does not verify the script details. We recommend that you check your script carefully before you submit it.

Error code	Description	Cause	Troubleshooting
26006	The required method is not found in the script.	The script runs correctly , but the script content is incorrect. The script must contain the protocolToRawData and rawDataToP rotocol methods. If they are not found, this error will be reported.	Go to the corresponding Product Details page in the IoT Platform console , and make sure that the protocolToRawData and rawDataToProtocol methods are defined in the script.
26007	The returned data format is incorrect after data parsing.	The script runs correctly , but the data format of the returned result is incorrect. The script must contain the protocolToRawData and rawDataToP rotocol methods. The result data of protocolToRawData must be byte[] array, and the result data of rawDataToProtocol must be jsonObj (JSON object). This error code is returned if the data format of the returned result does not match one of the defined data formats. For example , after a device reports data , the result is returned to the device. The returned result will also be parsed . If you have not defined protocolToRawData in the script, the returned data may be incorrect.	Check the script in the IoT Platform console. Enter the input parameters, run the script, and check whether the data format of the returned result is correct.

3.4 Firmware update

IoT Platform provides the firmware update function. To update firmware, you need to configure your device to support OTA updates. Then, in the IoT Platform console, you can upload a firmware file and push the firmware update file to devices. This topic describes how to configure firmware updates and manage firmware file versions.

Prerequisites

Before you use the firmware update function, make sure that you have developed your device to support OTA updates.

- If you use device SDKs, see [OTA updates](#).
- If you use AliOS Things, see [OTA tutorial for AliOS Things](#).

Procedure

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, click Maintenance > Firmware Update



Note:

To provide better services, IoT Platform now allows you to manage firmware versions by product. When you use the new version of the firmware update function for the first time, please associate the previously uploaded firmware files with products manually. You can only associate a firmware file to one product. After you associate the existing firmware files to products, you can add new firmware files.

3. On the Firmware Update page, click New Firmware.



Note:

Each Alibaba Cloud account can have up to 500 firmware files.

4. In the Add Firmware dialog box, enter the firmware information and upload the firmware file.

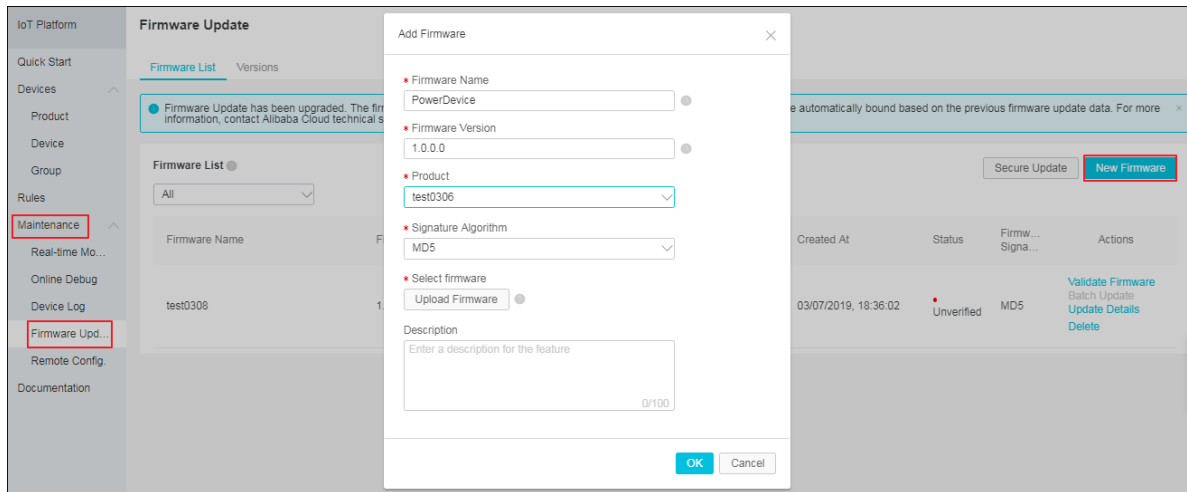


Table 3-4: Parameter description

Parameter	Description
Firmware Name	Enter a firmware name. The name must be 4 to 32 characters in length and can contain letters, numbers, Chinese characters, and underscores (_). It cannot begin with an underscore.
Firmware Version	Enter a version for the firmware. The version must be 1 to 64 characters in length and can contain letters, numbers, periods (.), hyphens (-), and underscores (_).
Product	Select the product to which the firmware belongs.
Signature Algorithm	Supported signature algorithms are MD5 and SHA256.
Upload Firmware	Upload a firmware file. Only files in BIN, TAR, GZ, and Zip format are supported. The size of a firmware file cannot exceed 1,000 MB.

5. (Optional) if your devices use chips with AliOS Things, you can use the secure update function.

We recommend that you activate the secure update function to ensure the integrity and confidentiality of the firmware. The secure update function requires device

information for firmware verification and firmware signature verification. If you use AliOS Things, see [OTA tutorial for AliOS Things](#).

- a) On the Firmware Update page, click Secure Update.
- b) In the Secure Update dialog box, turn the button of the secure update function to Activated for the products whose devices use AliOS Things.

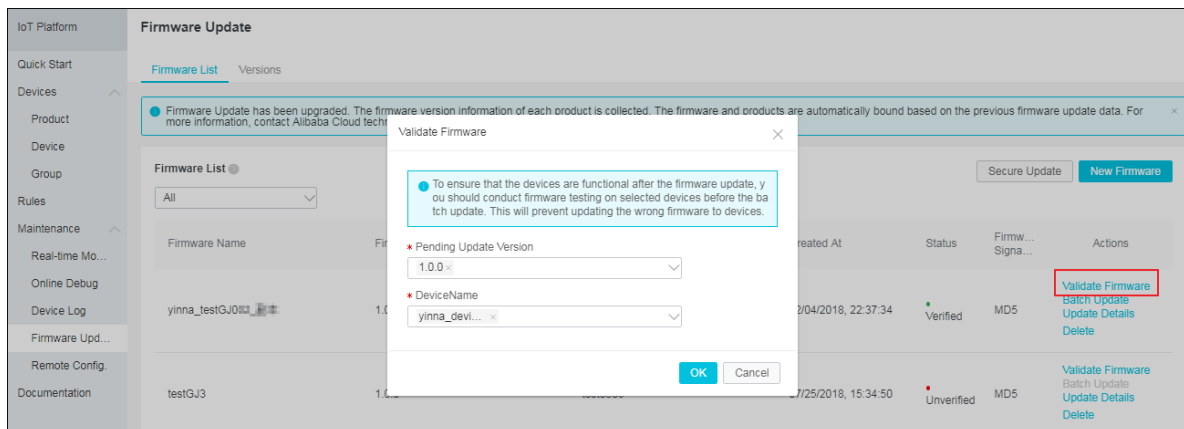
When the secure update function is Activated, you can click the corresponding Copy button to copy the key for device signature use.

6. In the firmware list, click the corresponding Validate Firmware button, and then test the uploaded firmware file on one or more devices.



Note:

After the firmware file is uploaded to IoT Platform, you must test the firmware file on one or more devices first. Only when you confirm that the devices have been successfully updated can the firmware file be used for batch update. You can launch multiple validations for a firmware.



Parameter	Description
Pending Update Version	The drop-down box displays the current firmware versions of all devices of the product. Select one or more versions that you want to update to the new version. After you select the versions, the devices with these firmware versions will be displayed when you click the drop-down button of DeviceName.
DeviceName	Select one or more devices to test the firmware file.



Note:

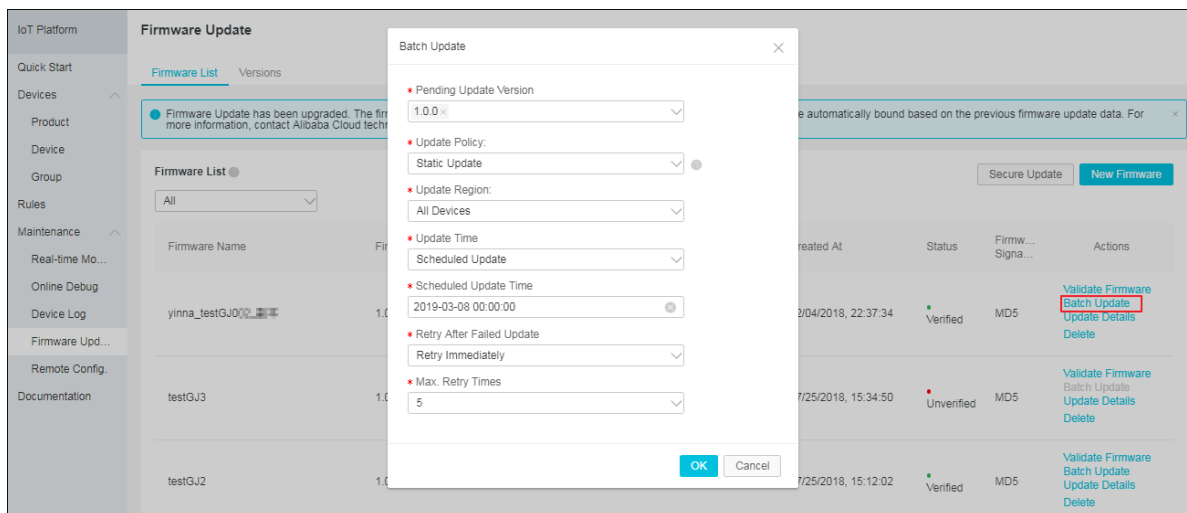
- Devices receive the firmware update notifications:
 - If the devices that connect to IoT Platform through MQTT are online, they will immediately receive the update notifications. If the devices are offline, the system will push the update notifications to the devices when they go online again.
 - If the devices using other connection protocols (such as CoAP or HTTPS) are online, they will immediately receive the update notifications. If the devices are offline, they cannot receive the notifications.
- Provided that you perform a firmware validation operation, the firmware status will change from Unverified to Verified. However, the status of the firmware does not indicate that the test devices have been updated successfully. Click Update Details to see the update result.

7. Click Batch Update, configure an update method, and then push update notifications to devices.



Note:

Make sure that the firmware file has successfully passed the verification before you perform a batch update.



Parameter	Description
Pending Update Version	The drop-down box displays the current firmware versions of all devices of the product. Select one or more versions that you want to update to the new version.

Parameter	Description
Update Policy	• Static Update: Only update activated devices that meet the specified criteria. • Dynamic Update: All devices that meets the specified criteria receive an update notification. If you select Dynamic Update, the system maintains the scope of devices that need to be updated, including devices that have reported the current versions and newly activated devices.
Apply Update to	• All Devices: All devices that belong to the product will be updated. • Selected Devices: If you select Selected Devices, Device Range field will appear. You then need to select devices to be updated. Only selected devices will be updated.  Note: You can select multiple pending versions if you select to update specified devices. The version that you previously selected for update is selected by default. If you have not specified any version, all versions are selected by default.
Update Time	Specify a time when the update performs. • Update Now: Update immediately after the request is submitted. • Scheduled Update: Manually specify a time for the system to push the update requests to devices. You can specify a time in the range of five minutes to seven days later.  Note: Scheduled Update is available only when the update policy is Static Update. If you specify a scheduled update time, in the Pending tab page of Firmware Details, you can see the scheduled update time.

Parameter	Description
Retry After Failed Update	Configure that when the system retries to send update request again if the update fails. Options: · Do Not Retry · Retry Immediately · Retry in 10 Minutes · Retry in 30 Minutes · Retry in 1 hour · Retry in 24 hours
Max. Retry Times	Select how many times the system can retry. Options: · 1 · 2 · 5

Result

Click Update Details to view the update status.

- **Pending:** This tab page lists the devices which are selected for update. Two types of pending status are available: Pending (Device offline) and Pending (Scheduled time: xxxx-xx-xx xx:xx:xx)
 - If the device is offline and the update time is scheduled for a later time, the status is shown as Pending (Scheduled time: xxxx-xx-xx xx:xx:xx).
 - When it reaches the scheduled time, and the device is still offline, the status will change to Pending (Device offline).
- **Updating:** This tab page lists the devices that have received the update notifications and have reported their update progresses to the console. If no update progress is received from the device, the progress ratio is 0.
- **Update Successful:** This tab page lists the devices which have been successfully updated.
- **Update Failed:** This tab page lists the devices that have failed the update and provides the reasons. The following are some causes of update failures:
 - The device has another update task in progress. After the device has finished the current update task, you can try to update it for this version again.
 - During the updating progress, a firmware package download failure, firmware file extraction failure, verification failure, or other failures occurred. In these cases, you can try updating again.

Click Versions on the Firmware Update page and then select a product to view the firmware used by the devices of the product.

- **Version Distribution:** Displays the percentages of firmware usages in the product. Names and versions of the top five firmware are displayed, and other firmware are grouped in Others.
- **Versions and Devices:** Displays all the firmware versions used by devices of the product and the number of devices that use the versions.
- **Device List:** Displays all the devices of the product. You can select a firmware version to view the devices that use this version.

3.5 Remote configuration

IoT Platform provides the remote configuration function, which allows device configurations to update online when the device is in service.

Prerequisites

- You have activated the remote configuration function in the IoT Platform console. If you have not activated this function, log on to the IoT Platform console and then, in the left-side navigation pane, click Maintenance > Remote Config.. Then, click Enable Service.
- You have configured your device SDK to support the remote configuration function. Define `FEATURE_SE RVICE_OTA_ ENABLED = y` in the device SDK. The SDK provides the `linkkit_cota_init` operation to initialize remote configurations such as Config Over The Air (COTA).

Introduction to the remote configuration function

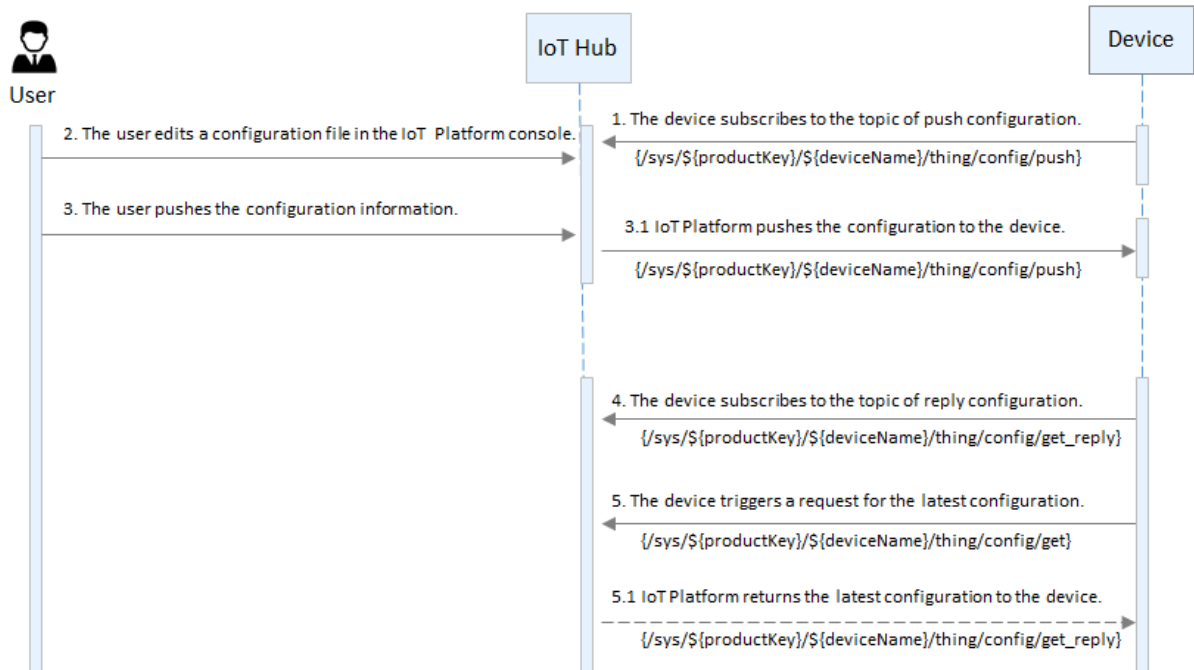
Developers often need to update device configurations, such as the system parameters, network parameters, and security policies of devices. Generally, device configurations are updated using the firmware update function. However, firmware update requires more time for firmware version maintenance, and devices must stop their services in order to install the update. To streamline the device configuration update process, IoT Platform provides the remote configuration function. This function enables you to complete configuration updates without service interruption.

With the remote configuration function, you can perform the following operations:

- Enable or disable remote configuration.

- Edit configuration files and perform version management in the IoT Platform console.
- Update the configuration information for all devices of a product at one time.
- Enable devices to send requests for configuration update from IoT Platform.

Remote configuration flow chart:



The processes involved in remote configuration include the ability to:

- Edit and save configuration files in the IoT Platform console.
- Push configuration updates to all devices of a product in the IoT Platform console. Then, when the devices receive the update requests, they immediately update their configurations.
- Devices can also send requests for configuration updates from IoT Platform, and then perform update when configuration information is received.

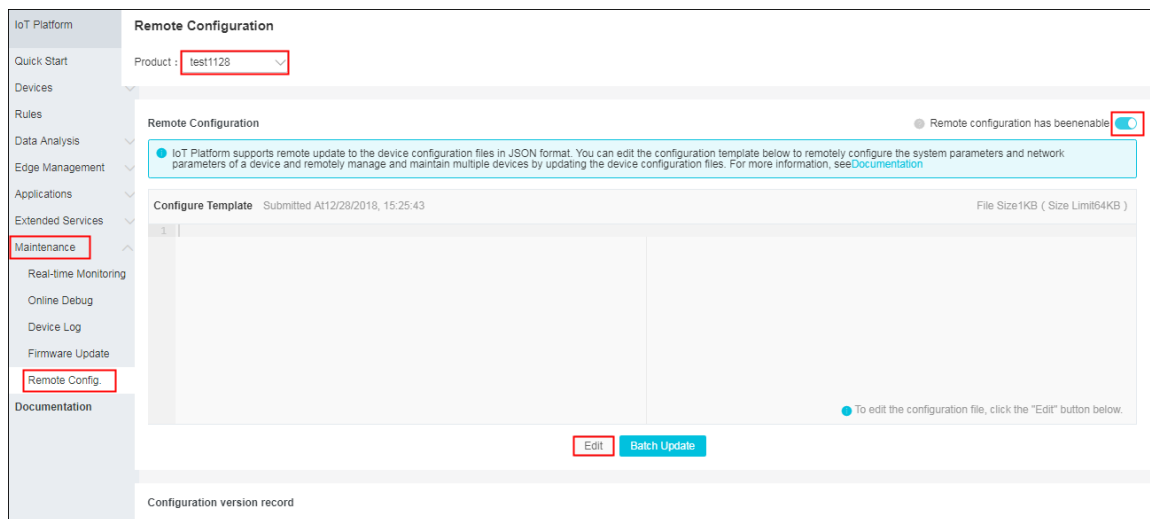
Use the remote configuration function

The remote configuration function is mainly designed for two scenarios, namely, you want to push configuration updates to devices from IoT Platform, or you want to allow devices to send requests for configuration updates. The process of using the remote configuration function varies based on different scenarios.

Scenario 1: Push configuration information to devices from IoT Platform.

In the IoT Platform console, you can push device configuration updates to all devices of a product.

1. Connect the devices to IoT Platform and configure the devices to subscribe to the topic `/ sys / ${ productKey } / ${ deviceName } / thing / config / push` .
2. In the IoT Platform console, edit a configuration file.
 - a. In the left-side navigation pane, click Maintenance > Remote Config..
 - b. Select the product for which you want to use the remote configuration function, and enable the function.



Note:

- Only if you enable the remote configuration function for the selected product can you edit a configuration template file for it.
- If the remote configuration function is not enabled, devices of the product cannot be updated in this way.

- A configuration template file that you edit here is used by all the devices of the product. Currently, you cannot push a configuration file to a specified device.

c. Click Edit, and then edit a configuration template in the area of Configuration Template.

IoT Platform

Quick Start

Devices

Rules

Data Analysis

Edge Management

Applications

Extended Services

Maintenance

Real-time Monitoring

Online Debug

Device Log

Firmware Update

Remote Config.

Documentation

Remote Configuration

Product : test1128

Remote Configuration

Remote configuration has been enable

IoT Platform supports remote update to the device configuration files in JSON format. You can edit the configuration template below to remotely configure the system parameters and network parameters of a device and remotely manage and maintain multiple devices by updating the device configuration files. For more information, see [documentation](#)

Configure Template Submitted At 01/09/2019, 17:28:47

File Size 1KB (Size Limit 64KB)

```
{  
  "temperature": 50  
}
```

Cancel Save

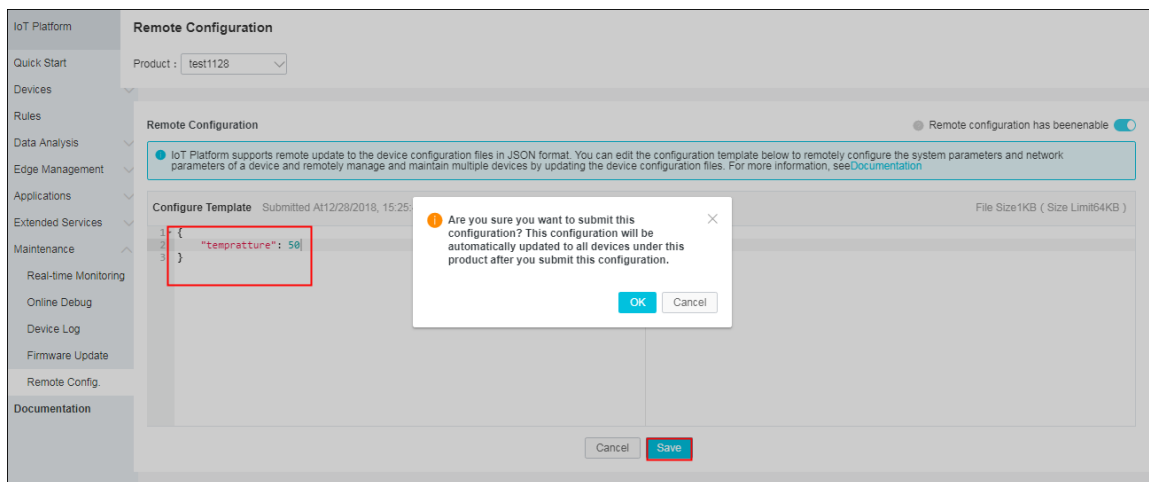


Note:

- Remote configuration files are JSON files. IoT Platform does not have special requirements for the configuration content. The system only checks the format of the data when you submit the configuration file. This is to prevent errors that are caused by format errors.

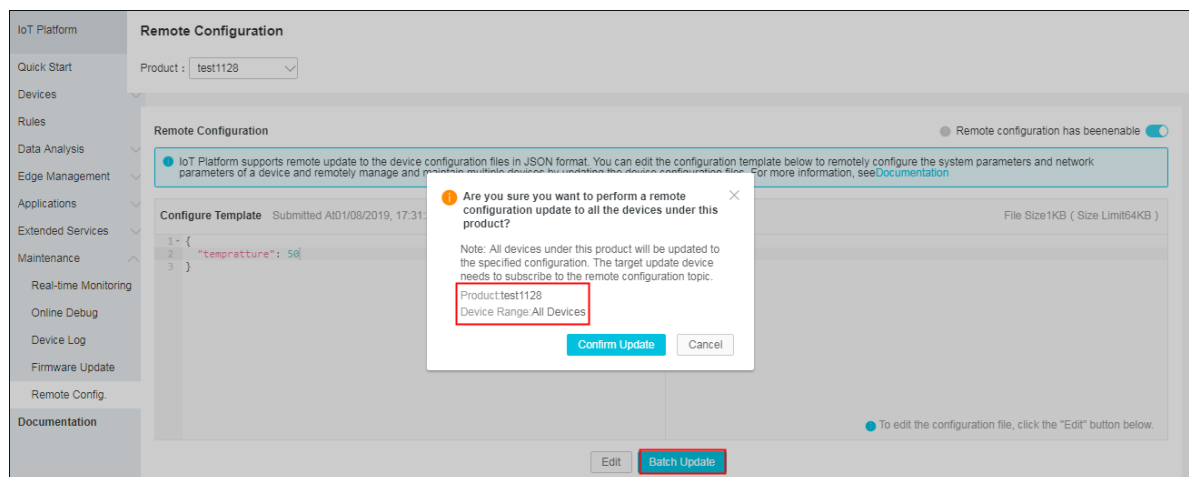
- The configuration file can be up to 64 KB. The file size is dynamically displayed in the upper-right corner of the editing area. Configuration files larger than 64 KB cannot be submitted.

d. After you have completed editing the configuration information, click **Save** to generate the configuration file. The system then allows devices to send requests for the configuration file.



3. Push the configuration file to devices. Click **Batch Update** and then IoT Platform sends the configuration file to all the devices of the product.

After you click **Batch Update**, the system may initiate SMS authentication to verify your account. If authentication is required, you need to first complete account verification, and then the system sends the configuration file to the devices.



Note:

- **Operation frequency limit:** You can only perform a batch update once per hour.

- If you want to stop pushing configuration updates, disable the remote configuration function for the product. The system then stops pushing the update file and will deny update requests from devices.

4. Devices automatically update the configuration after receiving the configuration file from IoT Platform.

Configuration file management:

The latest five configuration files are saved in the console by default. After you edit and save a new version of configuration file, the previous version is automatically displayed in the configuration version record list. You can view the update time and content of the displayed five versions.

Remote Configuration

test001

Remote Configuration ● Remote configuration has been enabled ☒

IoT Platform supports remote update to the device configuration files in JSON format. You can edit the configuration template below to remotely configure the system parameters and network parameters of a device and remotely manage and maintain multiple devices by updating the device configuration files. For more information, see [Documentation](#).

Configure Template Submitted At 11/20/2018, 22:33:10 File Size 1KB (Size Limit 64KB)

```

1 {
2   "setNetConfig": "xxxxxxxx"
3 }

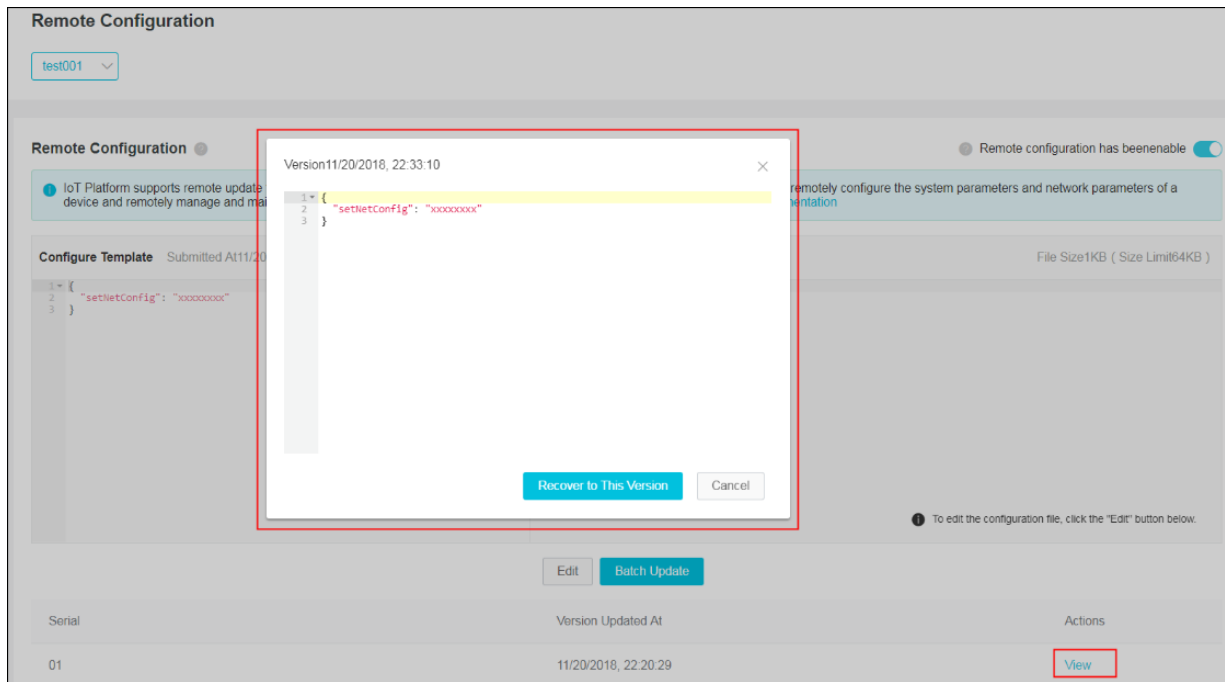
```

To edit the configuration file, click the "Edit" button below.

Edit Batch Update

Serial	Version Updated At	Actions
01	11/20/2018, 22:20:29	View

Click View to view the configuration content of the version. Click Recover to This Version, and the configuration content of this version will be displayed in the editing box. You can edit the content and then save it as a new version.



Scenario two: Devices send requests for configuration information.

If devices are configured to send requests for configuration information, you need to enable the remote configuration function. To do so, follow these steps:

1. Configure the devices to subscribe to the topic `/ sys / ${ productKey } / ${ deviceName } / thing / config / get_reply` .
2. In the IoT Platform console, enable the remote configuration function and edit a configuration file. For detailed steps, see the related procedures in [Scenario 1](#).
3. Configure the devices to call the `linkkit_invoke_cota_get_config` operation to trigger requests for remote configuration.
4. Configure the devices to send requests for the latest configuration updates through the topic `/ sys / ${ productKey } / ${ deviceName } / thing / config / get` .
5. IoT Platform returns the latest configuration information to the devices after receiving the requests.
6. The devices use the `cota_callback` function to process the configuration file that is sent through the remote configuration function.

4 Generic protocol SDK

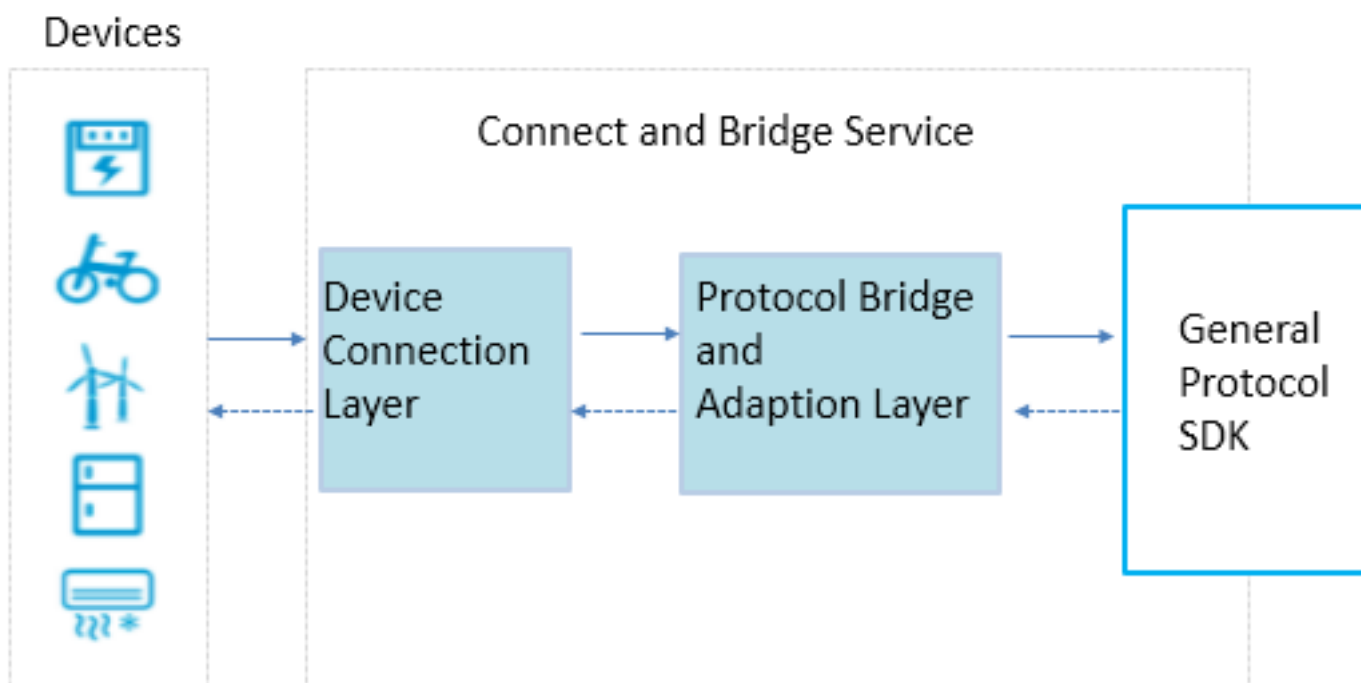
4.1 Overview

Alibaba Cloud IoT Platform supports communication over MQTT, CoAP, or HTTP. Other types of protocols, such as the fire protection agreement GB/T 26875.3-2011, Modbus, and JT808, are not supported. In specific scenarios, some devices may not be able to directly connect to IoT Platform. You must use the generic protocol SDK to build a bridge for your devices or platforms with IoT Platform, so that they can communicate with each other.

Architecture

The generic protocol SDK is a self-adaptive protocol framework. This SDK is used to provide a bridge service for the bi-directional communication between IoT Platform and your devices or platforms.

The following figure shows the architecture.



Scenarios

The generic protocol SDK can be applied to the following scenarios:

- Your device cannot be directly connected to IoT Platform because of the network or hardware restrictions.
- Your device supports only protocols that are not supported by IoT Platform.
- A connection is already established between the device and your server. You want to connect the device to IoT Platform without modifying the device and protocol.
- The device is directly connected to your server. Additional logic processing is required.

Features

The generic protocol SDK enables the bridge server to communicate with IoT Platform.

Basic features:

- Allows you to manage configurations based on a configuration file.
- Allows you to manage device connections.
- Provides upstream communication capabilities.
- Provides downstream communication capabilities.

Advanced features:

- Allows you to manage configurations based on interfaces.
- Provides interfaces that can be called to report properties, events, and tags.

Terms

Term	Description
device	The device in a real IoT scenario that cannot directly communicate with IoT Platform by using the protocols supported by IoT Platform.
bridge server	The server to which the device is connected. This server uses a specific protocol to communicate with the device and uses the generic protocol SDK to communicate with IoT Platform.
original protocol	The specific protocol used between the device and the bridge server. The generic protocol SDK does not involve the definition and implementation of the original protocol.
original device identifier	The unique identifier used by the device to communicate with the bridge server over the original protocol. Among the generic protocol SDK interface parameters, the <code>originalIdentity</code> parameter specifies the identifier of the device's original identity.

Term	Description
device certificate	The device certificate information obtained after you register the device with IoT Platform. The information includes ProductKey, DeviceName, and DeviceSecret. In a scenario that uses the generic protocol, you do not need to install the device certificate on the device. Instead, you must configure the generic protocol SDK file: <code>devices . conf</code> . The bridge maps the originalIdentity of the device to the device certificate.
bridge certificate	The device certificate information returned after you register the bridge device with IoT Platform. The information includes ProductKey, DeviceName, and DeviceSecret. The bridge certificate uniquely identifies the bridge in IoT Platform.

Development and deployment

Create products and devices

Log on to the IoT Platform console and create products and devices. For more information, see [Create a product](#) and [Create a device](#) or [Create multiple devices at a time](#).

Obtain the device certificate of the bridge. This certificate must be provided when you configure the generic protocol SDK.



Note:

Bridge is a virtual concept. You can use any device certificate as the certificate information of the bridge.

Configure the generic protocol SDK

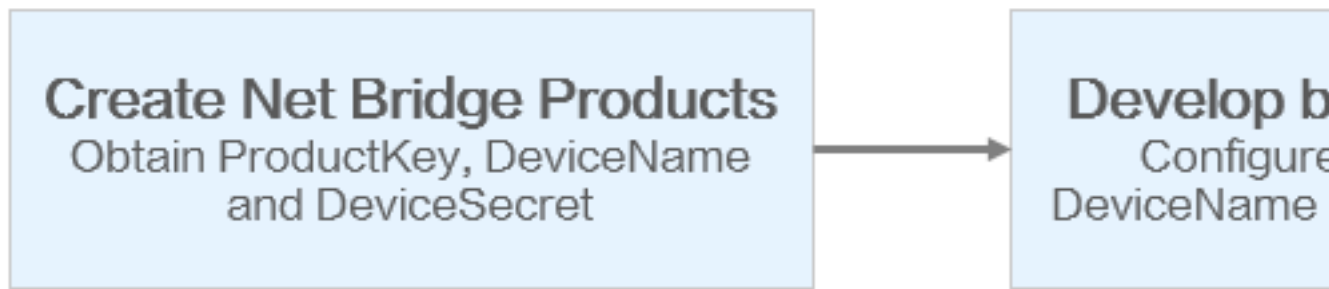
The generic protocol SDK supports only the Java language. Only JDK 1.8 and later versions are supported.

For more information about how to configure the generic protocol SDK, see [Use the basic features](#).

Deploy the bridge service

You can deploy a developed bridge service on Alibaba Cloud in a scalable manner by using Alibaba Cloud services such as [ECS](#) and [SLB](#). You can also deploy the bridge service in local environment to ensure secure communication.

The following figure shows the procedures of using ECS to deploy the bridge service:



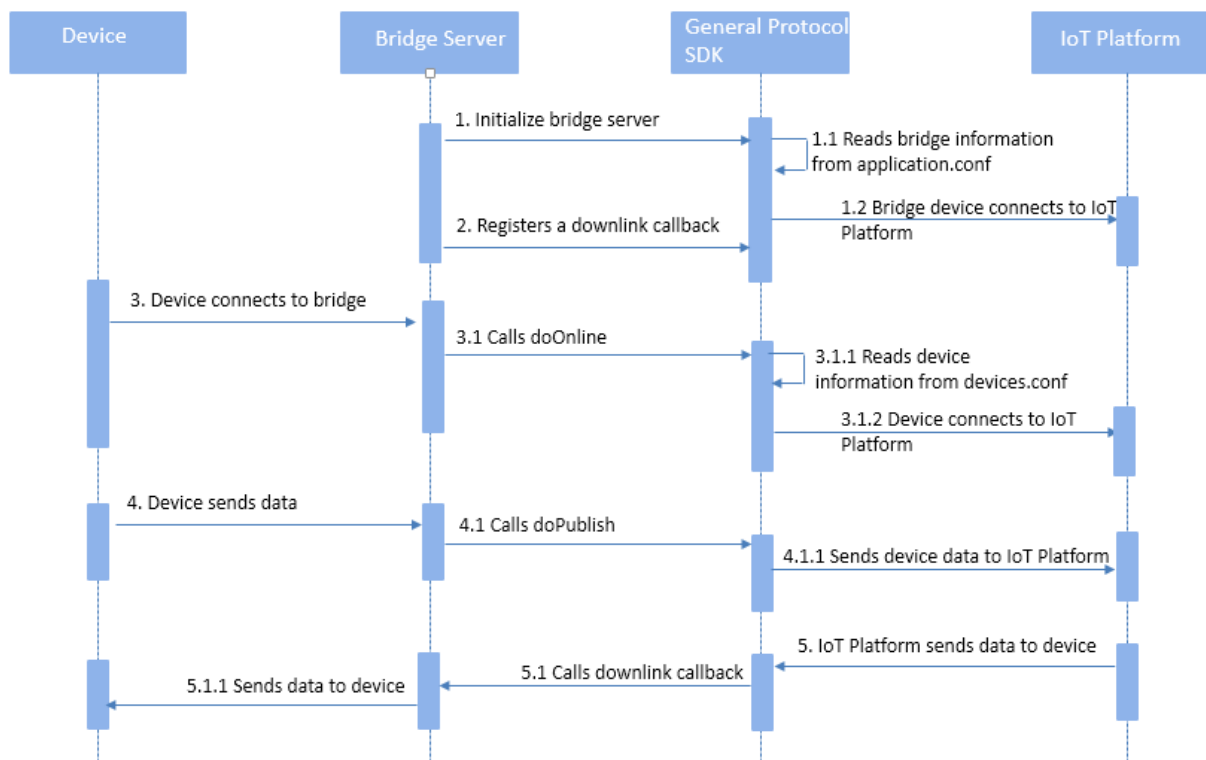
4.2 Use the basic features

Based on the generic protocol SDK, your device can connect to and communicate with Alibaba Cloud IoT Platform by using the bridge service. This topic describes how to configure the generic protocol SDK to implement basic capabilities, including device connection and disconnection and message upstreaming and downstreaming.

See [generic protocol SDK demo](#) in GitHub.

Flow diagram

The following flow diagram shows the overall process for how to use the generic protocol SDK to connect a device to IoT Platform.



Import the SDK

Add the following dependency in your maven project to import the generic protocol SDK.

```
< dependency >
  < groupId > com . aliyun . openservic es </ groupId >
  < artifactId > iot - as - bridge - sdk - core </ artifactId >
  < version > 2 . 0 . 0 </ version >
</ dependency >
```

Initialization

Initialize the SDK

You must create a `BridgeBootstrap` object and call the `bootstrap` method. After the generic protocol SDK initialization is complete, the SDK reads the bridge information and initiates a request for the bridge to connect to IoT Platform.

In addition to calling `bootstrap`, you can also register the `DownlinkChannelHandler` callback with the generic protocol SDK to receive downstream messages from IoT Platform.

Sample code:

```
BridgeBootstrap bridgeBootstrap = new BridgeBootstrap();
bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
    @Override
    public boolean pushToDevice(Session session, String
topic, byte[] payload) {
        // get message from cloud
        String content = new String(bytes);
        log.info("Get DownLink message, session:{}", {},
{}", session, topic, content);
        return true;
    }

    @Override
    public boolean broadcast(String topic, byte[] payload
) {
        return false;
    }
});
```

Configure bridge information

By default, a bridge is configured based on a configuration file. By default, the configuration file is read from `application.conf` under the default resource file path of the Java project (generally `src/main/resources/`). The file is in the format of [HOCON](#) (JSON superset). The generic protocol SDK uses `typesafe.config` to parse the configuration file.

You can configure a bridge device either by specifying a bridge device or dynamically registering a bridge device. This topic only describes how to specify a bridge device. For more information about how to dynamically register a bridge device, see [Dynamically register a bridge device](#).

Table 4-1: Bridge configuration parameters

Parameter	Required	Description
productKey	Yes	The key of the product to which the bridge device belongs .

Parameter	Required	Description
deviceName	No	The device name of the bridge device. · You must provide this parameter if you have registered the bridge device in advance and want to configure the device based on the specified device certificate information. · You do not need to provide this parameter if you have not registered the bridge device in advance and want to use the MAC address of the bridge server as the device name to dynamically register a device with IoT Platform.

Parameter	Required	Description
deviceSecret	No	The device secret of the bridge device. • You must provide this parameter if you have registered the bridge device in advance and want to configure the device based on the specified device certificate information. • You do not need to provide this parameter if you choose to dynamically register the bridge device rather than have the bridge device registered in advance.

Parameter	Required	Description
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway service. The bridge device and IoT Platform establish a persistent connection over the HTTP/2 protocol. The endpoint is in the format of <code>\${productKey}.iot-as-http2.\${RegionId}.aliyuncs.com:443</code>. Replace <code>\${ProductKey}</code> with the ProductKey of the product to which your bridge device belongs. Replace <code>\${RegionId}</code> with the ID of the region where your service is located.

Parameter	Required	Description
authEndpoint	Yes	The service URL for device authentication. The device authentication service URL is in the format of <code>https://iot-auth.{RegionId}.aliyuncs.com/auth/bridge</code>. Replace <code>{RegionId}</code> with the ID of the region where your service is located. For more information about regions, see Regions and zones. For example, if the region is China (Shanghai), then the device authentication service address is <code>https://iot-auth.cn-shanghai</code>.

Parameter	Required	Description
popClientProfile	No	This parameter must be provided if you use the MAC address of the bridge server to dynamically register the bridge device. For more information, see Dynamically register a bridge device.

Use the following format to configure the bridge device certificate:

```
# Server endpoint
http2Endpoint = "https:// altn70BmTc d . iot - as - http2 . cn
- shanghai . aliyuncs . com : 443 "
authEndpoint = "https:// iot - auth . cn - shanghai . aliyuncs .
com / auth / bridge "

# Gateway device info , productKey & deviceName &
deviceSecret
productKey = ${ bridge - ProductKey - in - Iot - Platform }
deviceName = ${ bridge - DeviceName - in - Iot - Platform }
deviceSecret = ${ bridge - DeviceSecret - in - Iot - Platform }
```

Device authentication and connection

Configure device connection

The device connection interface in the generic protocol SDK:

```
/**
 * Device authentication
 * @param newSession Device session information , which
is returned in a downstream callback .
 * @param originalId entity The original identity of
the device
 * @return
 */
```

```
public boolean doOnline ( Session newSession , String
originalId entity );
```

When the device is connected to the bridge device, it must pass in a session. When a downstream message is called back, the session is called back to the bridge device. The session contains the original identifier field, so that the bridge device can determine which device the message came from.

In addition, the session also has an optional channel field, which can be designed to store device connection information. For example, your bridge server is built based on Netty. You can use this field to store the channel object corresponding to the persistent connection of the device. When a message is sent from IoT Platform, the bridge device can directly obtain the channel from the session for subsequent operations. The data type of the channel field is Object. The generic protocol SDK does not process data stored in the channel field. You can also store any device-related information in the channel field according to the scenario.

Sample code for device connection:

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler ();
// Create a session
Object channel = new Object ();
Session session = Session.newInstance ( originalId entity ,
channel );
// Connect the device to the bridge
boolean success = uplinkHandler.doOnline ( session ,
originalId entity );
if ( success ) {
    // If the device is connected , the bridge device
    accepts new communication requests from the device .
} else {
    // If the device connection fails , the bridge device
    rejects subsequent communication requests , such as
    disconnection requests .
}
```

Map an original identifier to a device certificate

You must configure the mapping between the device certificate and the original identifier of a device. By default, a configuration file is used to configure the mapping. The configuration file is read from `devices.conf` under the default resource file path of the Java project (generally `src / main / resources /`). The file is in the format of [HOCON](#) (JSON superset). The generic protocol SDK uses `typesafe.config` to parse the configuration file.

Use the following format to configure the device certificate information:

```
{ device - originalId entity } {
  productKey : ${ device - ProductKey - in - Iot - Platform }
  deviceName : ${ device - DeviceName - in - Iot - Platform }
  deviceSecret : ${ device - DeviceSecret - in - Iot - Platform }
}
```

Parameter	Required	Description
productKey	Yes	The key of the product to which the device belongs.
deviceName	Yes	The device name.
deviceSecret	Yes	The device secret.

Device sends data to IoT Platform

The interface for data upstreaming in the generic protocol SDK:

```
/**
 * Send upstream messages from the device by
 * synchronously calling the interface
 * @param originalId entity The original identifier of
 * the device
 * @param protocolMsg The message to be sent,
 * including the topic, payload, and QoS information
 * @param timeout The timeout period in seconds
 * @return Indicates whether the message is sent
 * successfully within the timeout period
 */
boolean doPublish (String originalId entity, ProtocolMessage
protocolMsg, int timeout);
/**
 * Send upstream messages from the device by
 * asynchronously calling the interface
 * @param originalId entity The original identifier of
 * the device
 * @param protocolMsg The message to be sent,
 * including the topic, payload, and QoS information
 * @return After this interface is called, CompletableFuture
 * is returned immediately. The caller can
 * further process this Future.
 */
CompletableFuture < ProtocolMessage > doPublishAsync (String
originalId entity,
ProtocolMessage
protocolMsg);
```

Sample code:

```
DeviceIdentity deviceIdentity =
    ConfigFactory.getDeviceConfigManager().getDeviceIdentity(
originalId entity);
ProtocolMessage protocolMessage = new ProtocolMessage();
protocolMessage.setPayload("Hello world".getBytes());
protocolMessage.setQos(0);
```

```

protocolMessage . setTopic ( String . format ("% s /% s / update
",
        deviceId entity . getProduct Key (), deviceId entity .
getDeviceName ());
// Synchronous sending
int timeoutSeconds = 3 ;
boolean success = upLinkHandler . doPublish ( originalId
entity , protocolMessage , timeoutSeconds );
// Asynchronous sending
upLinkHandler . doPublishAsync ( originalId entity ,
protocolMessage );

```

Bridge device pushes data to device

When the bridge device calls the bootstrap method, it registers

DownlinkChannelHandler with the generic protocol SDK. When the generic protocol SDK receives a downstream message, it calls back the pushToDevice method in DownlinkChannelHandler. You can edit the pushToDevice method to configure the bridge device to process downstream messages.



Note:

Do not create a time-consuming logic in the pushToDevice method. Otherwise, the thread that receives downstream messages will be blocked. Use the asynchronous transmission if a time-consuming logic or I/O logic exists, for example, sending downstream messages through a persistent connection to the devices.

Sample code:

```

private static ExecutorService executorService = new
ThreadPoolExecutor (
    Runtime . getRuntime (). availableProcessors (),
    Runtime . getRuntime (). availableProcessors () * 2 ,
    60 , TimeUnit . SECONDS ,
    new LinkedBlockingQueue <> ( 1000 ),
    new ThreadFactoryBuilder (). setDaemon ( true ). setNameFor
mat ( " bridge - downlink - handle -% d " ). build (),
    new ThreadPoolExecutor . AbortPolicy ());
public static void main ( String args []) {
    // Use application . conf & devices . conf by default
    bridgeBootstrap = new BridgeBootstrap ();
    bridgeBootstrap . bootstrap ( new DownlinkChannelHandler
() {
        @ Override
        public boolean pushToDevice ( Session session ,
String topic , byte [] payload ) {
            // get message from cloud
            // get downlink message from cloud
            executorService . submit (() -> handleDown
LinkMessage ( session , topic , payload ));
            return true ;
        }
        @ Override
        public boolean broadcast ( String s , byte [] bytes )
{

```

```

        return false ;
    }
});
}
private static void handleDownLinkMessage ( Session
session , String topic , byte [] payload ) {
    String content = new String ( payload );
    log . info ( " Get DownLink message , session : {}, topic :
{}", content : {} ", session , topic , content );
    Object channel = session . getChannel ();
    String originalId entity = session . getOriginalIdentity
();
    // for example , you can send the message to device
    via channel , it depends on you specific server
    implementation
}

```

Parameter	Description
Session	A session is transmitted by a device when the device is connecting to the bridge device. A session can be used to identify the device to which the downstream message is sent.
topic	The topic of the downstream message.
payload	The payload of a downstream message in binary format.

Device disconnection

A device is disconnected under the following situations:

- When the bridge device is disconnected from IoT Platform, all connected devices are automatically disconnected from IoT Platform.
- The bridge device reports a disconnection request for a device to IoT Platform.

The interface for bridge device to report device disconnection in the generic protocol SDK:

```

/**
 * Report a disconnection request to IoT Platform for
 * a device
 * @param originalId entity The original identifier of
 * the device
 * @return Indicates whether the message is sent
 * successfully
 */

```

```
boolean doOffline ( String originalId entity );
```

Sample code:

```
upLinkHandler . doOffline ( originalId entity );
```

4.3 Use the advanced features

This topic describes how to use the advanced features of the generic protocol SDK. The advanced features include customizing the configuration file path, configuring dynamic bridge registration, calling the data reporting interfaces encapsulated in the generic protocol SDK to report properties, events, and tags.

Customize configurations

By default, the configuration file of a bridge device and the mapping configuration file of the device certificate are read from `application.conf` and `devices.conf`, respectively, under a fixed path. The generic protocol SDK allows you to customize configurations. Before you call `bootstrap`, call the `ConfigFactory.init` method to customize the path of a configuration file. You can also customize an instance to implement the corresponding interface.

Sample code to customize configurations:

```
// Define config
// You can specify the location path of config files
// or you can create an instance and implement the
// corresponding interface
// Config.init() must be called before bridgeBootstrap.bootstrap()
ConfigFactory.init (
    ConfigFactory.getBridgeConfigManager ("application -
self - define.conf"),
    selfDefineDeviceConfigManager);
bridgeBootstrap.bootstrap();

private static DeviceConfigManager selfDefineDeviceConfig
igManager = new DeviceConfigManager () {
    @Override
    public DeviceIdentity getDeviceIdentity ( String
originalId entity ) {
        // Suppose you dynamically get deviceInfo in
other ways
        return devicesMap.get ( originalId entity );
    }

    @Override
    public String getOriginalIdentity ( String productKey ,
String deviceName ) {
        // you can ignore this
        return null ;
    }
}
```

```
};
```

Dynamically register a bridge device

When you need to deploy a bridge application on a large number of servers, it is cumbersome to specify different bridge devices for different bridge servers. You can configure the bridge information file `application.conf` to dynamically register bridge devices with IoT Platform. You must provide the `productKey` and `popClientProfile` parameters in the configuration file. The generic protocol SDK will call the IoT Platform API and use the bridge servers' MAC codes as the device names to register bridge devices.



Note:

- To dynamically register bridge devices, you only need to modify the bridge configuration file. The call code is the same as [Use the basic features](#).
- If the bridge information is already specified in the bridge configuration file, no device is created. The generic protocol SDK calls the IoT Platform API and uses the bridge server's MAC code as the device name to register a bridge device only if the following conditions are met: The `deviceName` and `deviceSecret` parameters are left empty in the configuration file; all parameters in `popClientProfile` are specified. If a device is already registered using the current MAC code, the device is directly used as the bridge device.
- If a bridge is configured by using this method, we recommend that you do not perform debugging on a local client by using the configurations for the production environment. Each time the program is debugged on a local client, the generic protocol SDK uses the MAC code of the client to register a bridge device, and associates all devices in the device configuration file `devices.conf` with the bridge. We recommend that you use dedicated devices for testing to perform debugging to avoid interference with the production environment.

Table 4-2: Configuration parameters

Parameter	Required	Description
<code>productKey</code>	Yes	The ProductKey of the product to which the bridge device belongs.

Parameter	Required	Description
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway service. The bridge device and IoT Platform establish a persistent connection over the HTTP/2 protocol. The endpoint is in the format of <code>\${productKey}.iot-as-http2.\${RegionId}.aliyuncs.com:443</code>. Replace <code>\${productKey}</code> with the ProductKey of the product to which your bridge device belongs. Replace <code>\${RegionId}</code> with the ID of the region where your service is located. For more information about regions, see Regions and zones. For example, if the ProductKey of the bridge device is <code>alabcabc123</code>, the region is China (Shanghai), then the HTTP/2 gateway service endpoint is <code>alabcabc123.iot-as-http2.cn-shanghai.aliyuncs.com:443</code>.
authEndpoint	Yes	The service URL for device authentication. The device authentication service URL is in the format of <code>https://iot-auth.\${RegionId}.aliyuncs.com/auth/bridge</code>. Replace <code>\${RegionId}</code> with the ID of the region where your service is located. For more information about regions, see Regions and zones. For example, if the region is China (Shanghai), then the device authentication service address is <code>https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge</code>.

Parameter	Required	Description
popClientProfile	Yes	After this parameter is configured, the generic protocol SDK calls the IoT Platform API to automatically register bridge devices. For more information, see the following table: Parameters in popClientProfile.

Table 4-3: Parameters in popClientProfile

Parameter	Required	Description
accessKey	Yes	The AccessKey ID of your Alibaba Cloud account. Log on to the Alibaba Cloud console and click your account avatar to go to the Account Management page. You can create or view the AccessKey information.
accessSecret	Yes	The AccessKey Secret of your Alibaba Cloud account.
name	Yes	The IoT Platform service region to which the bridge device connects. This parameter indicates the region to which the product identified by <code>productKey</code> belongs. For more information about regions, see Regions and zones.
region	Yes	The ID of the IoT Platform service region to which the bridge device connects. This parameter indicates the region to which the product identified by <code>productKey</code> belongs. This parameter is expressed in the same way as the <code>name</code> parameter.
product	Yes	The product name. Set the value to <code>IoT</code>.

Parameter	Required	Description
endpoint	Yes	The endpoint of the API. The endpoint is in the format of <code>iot . \${ RegionId }. aliyuncs . com .</code> Replace <code>\${RegionId}</code> with the ID of the region where your service is located. For more information about regions, see Regions and zones. For example, If the region is China (Shanghai), the endpoint is <code>iot . cn - shanghai . aliyuncs . com .</code>

Sample code to dynamically register bridge devices:

```
# Server endpoint
http2Endpoint = " https ://${ YourProductKey }. iot - as - http2
.cn - shanghai . aliyuncs . com : 443 "
authEndpoint = " https :// iot - auth . cn - shanghai . aliyuncs .
com / auth / bridge "

# Gateway device info
# You can also specify productKey only , and dynamic
register deviceName & deviceSecret in runtime
productKey = ${ YourProductKey }

# If you dynamic register gateway device using your
mac address , you have to specify ' popClientProfile '
# otherwise you can ignore it
popClientProfile = {
    accessKey = ${ YourAliyun AccessKey }
    accessSecret = ${ YourAliyun AccessSecret }
    name = cn - shanghai
    region = cn - shanghai
    product = Iot
    endpoint = iot . cn - shanghai . aliyuncs . com
}
```

Call interfaces to report TSL data

To facilitate use and reduce your encapsulation operations, the generic protocol SDK encapsulates data reporting interfaces. They are `reportProperty`, `fireEvent`, and `updateDeviceTag`. The device can use these interfaces to report properties, report events, and update device tags.

Prerequisites and usage guidelines:

- Before you call `reportProperty` and `fireEvent` to report properties and events, log on to the [IoT Platform console](#) and go to the Product Details page of the corresponding product. Then, click the Define Feature tab and define properties and events. For more information, see [Define features](#).
- If the tag that is specified in `updateDeviceTag` already exists, the tag value is updated. If the tag does not exist, the tag is automatically created. To check the call result, you can log on to the IoT Platform console and go to the Device Details page of the corresponding device.

Sample code:

```
TslUplinkH andler  tslUplinkH andler = new  TslUplinkH andler
();
// report  property
// Property 'testProp' is defined in IoT Platform Web
Console
String requestId = String . valueOf ( random . nextInt ( 1000 ));
tslUplinkH andler . reportProp erty ( requestId , originalId
entity , " testProp ", random . nextInt ( 100 ));

// fire  event
// Event 'testEvent' is defined in IoT Platform Web
Console
requestId = String . valueOf ( random . nextInt ( 1000 ));
HashMap < String , Object > params = new  HashMap < String ,
Object > ();
params . put ( " testEventP aram ", 123 );
tslUplinkH andler . fireEvent ( originalId entity , " testEvent ",
ThingEvent Types . INFO , params );

// update  device  tag
// 'testDevice Tag' is a tag key defined in IoT
Platform Web Console
requestId = String . valueOf ( random . nextInt ( 1000 ));
tslUplinkH andler . updateDevi ceTag ( requestId , originalId
entity , " testDevice Tag ", String . valueOf ( random . nextInt (
1000 )));
```

The parameters in this example are described as follows:

Parameter	Description
<code>requestId</code>	The request ID.
<code>originalIdentity</code>	The original identifier of the device.
<code>testProp</code>	The identifier of the property. For this example, make sure that you have defined a property with the identifier as <code>testProp</code> in the IoT Platform console. This sample code indicates to report the value of property <code>testProp</code> .

Parameter	Description
<code>random.nextInt(100)</code>	The property value to be reported. The value range of the property value is also defined in the IoT Platform console. In this example, use <code>random . nextInt (100)</code> to indicate a random number less than 100.
<code>testEvent</code>	The identifier of the event. For this example, make sure that you have defined an event with the identifier as <code>testEvent</code> in the IoT Platform console. This sample code indicates to report event <code>testEvent</code> .
<code>ThingEventTypes.INFO</code>	The event type. <code>ThingEvent Types</code> specifies the event type. A value of <code>INFO</code> indicates that the event type is Info. For this example, make sure that you have selected Info as the event type when you defined event <code>testEvent</code> in the IoT Platform console.
<code>params</code>	The output parameters of the event. The identifier, data type, and value range of output parameters are also defined in the IoT Platform console. In this example, the identifier of the output parameter is <code>testEventParam</code> , and the value is 123.
<code>testDeviceTag</code>	The key of the tag. The data type is String. In this example, the key is <code>testDeviceTag</code> . Set the key of the tag as instructed based on your requirements. For more information, see Device tags .
<code>String.valueOf(random.nextInt(1000))</code>	The value of the tag. The data type is String. In this example, <code>String . valueOf (random . nextInt (1000))</code> indicates a random number less than 1000. Set the value of the tag as instructed based on your requirements. For more information, see Device tags .

5 RRPC

5.1 What is RRPC?

Because the Message Queuing Telemetry Transport (MQTT) protocol uses a publish/subscribe-based asynchronous communication method, this protocol is not suitable for scenarios where the server needs to synchronously send requests to devices and receive responses from the devices. In response to the issue, IoT Platform enables synchronous request and response communication without the need to modify the MQTT protocol. To do so, the server calls the IoT Platform API.

Terminology

- **RRPC:** RRPC is short for Revert-RPC. RPC (Remote Procedure Call) uses a form of client-server interaction, and allows you to execute a procedure in a remote place without knowing the details for the remote interaction. RRPC allows you to send a request to a specified device and receive a response from the device.
- **RRPC request message:** The message that is sent to a device from the cloud.
- **RRPC response message:** The response message that is sent to the cloud from a device.
- **RRPC message ID:** A unique message ID that is generated by IoT Platform for each RRPC request.
- **RRPC subscription topic:** A topic that a device subscribes to for RRPC messages. The topic includes a wildcard (+).

Message communication using RRPC

1. When IoT Platform receives an API call from the server, it sends an RRPC request message to the device. The message body is any input data, and the topic is the topic defined by IoT Platform, which includes the unique RRPC message ID.
2. After the device receives the request message, it returns an RRPC response message to the cloud according to the defined topic format, and including the RRPC message ID. IoT Platform extracts the message ID from the topic, matches the ID with the ID of the request, and then sends the response to the server.

3. If the device is offline when the call is performed, IoT Platform returns an error message to the server indicating that the device is offline. If the device does not send any response message within the timeout period (eight seconds), IoT Platform then returns a timeout error to the server.

Topic format

Topics are implemented in different formats for different methods.

- For information about system topics, see [System-defined topics](#).
- For information about custom topics, see [Custom topics](#).

5.2 System-defined topics

With RRPC method, you can establish communications between devices and IoT Platform by using system-defined topics. These topics include the ProductKey and DeviceName of the devices.

System-defined topics

The formats of system-defined topics that are used in RRPC calls are as follows:

- RRPC request topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/request/\${messageId}
- RRPC response topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/response/\${messageId}
- RRPC subscription topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/request/+

In the topic formats, \${YourProductKey} and \${YourDeviceName} are device information used to identify a device, and \${messageId} is the RRPC message ID issued by IoT Platform.

Use RRPC

1. Call RRpc API

Call the RRpc API and input your device information into the SDK. For API calling method, see [RRpc](#).

The following example uses Java SDK to show the calling method:

```
RRpcRequest request = new RRpcRequest();
request.setProductKey("testProductKey");
request.setDeviceName("testDeviceName");
```

```
request . setRequest Base64Byte ( Base64 . getEncoder ().
encodeToSt ring (" hello world "));
request . setTimeout ( 3000 );
RRpcRespon se response = client . getAcsResp onse ( request
);
```

2. The device returns the response.

When the device receives the RRPC request message, it returns a RRPC response message based on the request topic format.

The device extracts the message ID from the request topic, `/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/${messageId}`, generates a corresponding response, and then sends a response message to IoT Platform.

5.3 Custom topics

RRPC supports calling custom topics so that devices can communicate with the cloud. A communication topic contains the entire custom topic.

Topic formats

The format of a topic for RRPC is as follows:

- Request topic: `/ext/rrpc/${messageId}/${topic}`
- Reply topic: `/ext/rrpc/${messageId}/${topic}`
- Subscription topic: `/ext/rrpc+/ ${topic}`

In the preceding formats, `${messageId}` indicates the message ID generated by IoT Platform, and `${topic}` indicates the topic you created.

RRPC connection

1. Connect the device to the cloud SDK.

Call the RRPC API to connect your device to the cloud SDK. For more information about the call method, see [RRPC](#).

The following example uses the Java SDK for the call method:

```
RRpcReques t request = new RRpcReques t ();
request . setProduct Key (" testProduc tKey ");
request . setDeviceN ame (" testDevice Name ");
request . setRequest Base64Byte ( Base64 . getEncoder ().
encodeToSt ring (" hello world "));
request . setTopic ("/ testProduc tKey / testDevice Name / get
"); // If you want to use your custom topic , enter
the custom topic .
request . setTimeout ( 3000 );
```

```
RRpcResponse response = client . getAcsResponse ( request );
```

To use a custom topic, make sure that your Java SDK (aliyun-java-sdk-iot) version is 6.0.0 or later.

```
< dependency >
  < groupId > com . aliyun </ groupId >
  < artifactId > aliyun - java - sdk - iot </ artifactId >
  < version > 6 . 0 . 0 </ version >
</ dependency >
```

2. Connect the device to the cloud.

If you want the cloud to send RRPC call requests to the device using a custom topic, when you configure the MQTT communication protocol you must add the parameter `ext=1` into `clientId`. For more information, see [Establish MQTT over TCP connections](#).

For example, the original `clientId` that the device sends is as follows:

```
mqttClientId : clientId +"| securemode = 3 , signmethod =
hmacsha1 , timestamp = 132323232 |"
```

After `ext=1` is added to the `clientId`, the `clientId` that the device sends is as follows:

```
mqttClientId : clientId +"| securemode = 3 , signmethod =
hmacsha1 , timestamp = 132323232 , ext = 1 |"
```



Note:

If you use RRPC to establish communication between your devices and the cloud, and you use a custom topic, make sure that:

- The topic variable in the message that is sent from the cloud is not empty.
- The parameter `ext=1` is added into `clientId`.

3. Return the reply topic.

The request topic can be used as the reply topic because the format of the reply topic is the same as that of the request topic, and the `messageId` is not extracted.

6 Device shadows

6.1 Device Shadow overview

IoT Platform provides the Device Shadow function to cache property information for a device. If the device is online, the device can directly receive commands from IoT Platform. If the device is offline, the device can actively request for cached commands from IoT Platform after it comes online again.

A device shadow is a JSON file that is used to store the reported status and desired status information for a device.

Each device has only one shadow. A device can obtain and set the shadow over MQTT for status synchronization. The synchronization is bi-directional, either from the shadow to the device or from the device to the shadow.

Scenarios

- **Scenario 1:** In an unstable network, a device frequently disconnects from and reconnects to IoT Platform.

The device frequently disconnects from and reconnects to IoT Platform due to network instability. When an application that interacts with the device requests the current device status, the device is offline, which leads to a request failure. When the device is reconnected, the application fails to initiate another device status request.

The Device Shadow function can synchronize with the device to update and store the latest device status information in the device shadow. The application can obtain the current device status information from the device shadow despite of the connection status.

- **Scenario 2: Multiple applications simultaneously request the device status information.**

In a stable network, a device must respond to each status request from multiple applications, even if the responses are the same. The device may be overloaded with the requests.

By using the Device Shadow function, the device only needs to synchronize status information to the device shadow that is stored in IoT Platform. Applications can request the latest device status information from the device shadow instead of the target device. In this way, applications are decoupled from the device.

- **Scenario 3: Device disconnection**
 - In an unstable network, a device frequently disconnects from and reconnects to IoT Platform. When an application sends a control command to the device, the device is offline and the command fails to be dispatched to the device.
 - Quality of Service 1 or 2 (QoS 1 or 2) may solve this issue. However, we recommend that you do not use this method. This method increases the workload of the server.
 - By using the Device Shadow function, IoT Platform stores the control commands from the application to the device shadow. Each command is stored with the timestamp when the command was received. After the device is reconnected to IoT Platform, the device obtains these commands and checks the timestamp of each command to determine whether to run the command.
 - A device goes offline and fails to receive commands from the application. When the device is reconnected, the device runs only the valid commands by checking the timestamp of each command that is pulled from the device shadow.

View and update a device shadow

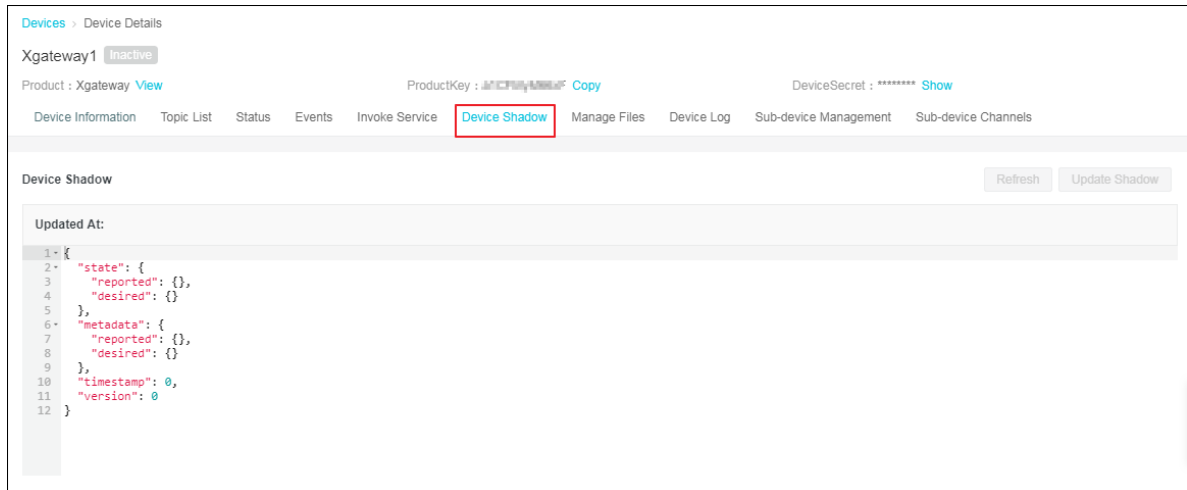
You can view and update the shadow of a device in the IoT Platform console.

Procedure:

1. Log on to the [IoT Platform console](#).
2. From the left-side navigation pane, choose Devices > Device.
3. Click View next to the corresponding device. The Device Details page appears.

4. Click the Device Shadow tab.

You can view the shadow that contains the latest information that is reported by the device.



5. Click Update Shadow, and enter the desired status information in the "desired" section.

For more information about the shadow file format, see [Device shadow JSON format](#).

The device obtains the desired status information by subscribing to a specific topic. When the device is online, IoT Platform pushes the desired value to the device in real time.

When the device is offline, the device's shadow caches the desired status information. After the device comes online again, it actively pulls the latest desired status information from IoT Platform.

Related API operations

Obtain a device shadow: [GetDeviceShadow](#)

Update a device shadow: [UpdateDeviceShadow](#)

6.2 Device shadow JSON format

Format of the device shadow JSON file

The format is as follows:

```
{
  "state": {
```

```
" desired ": {
  " attribute1 ": integer2 ,
  " attribute2 ": " string2 ",
  ...
  " attributeN ": boolean2
},
" reported ": {
  " attribute1 ": integer1 ,
  " attribute2 ": " string1 ",
  ...
  " attributeN ": boolean1
},
" metadata ": {
  " desired ": {
    " attribute1 ": {
      " timestamp ": timestamp
    },
    " attribute2 ": {
      " timestamp ": timestamp
    },
    ...
    " attributeN ": {
      " timestamp ": timestamp
    }
  },
  " reported ": {
    " attribute1 ": {
      " timestamp ": timestamp
    },
    " attribute2 ": {
      " timestamp ": timestamp
    },
    ...
    " attributeN ": {
      " timestamp ": timestamp
    }
  }
},
" timestamp ": timestamp ,
" version ": version
}
```

The JSON properties are described in [Table 6-1: JSON property](#).

Table 6-1: JSON property

Property	Description
desired	The desired status of the device. The application writes the desired property of the device, without accessing the device.

Property	Description
reported	The status that the device has reported. The device writes data to the reported property to report its latest status. The application obtains the status of the device by reading this property.
metadata	The device shadow service automatically updates metadata according to the updates in the device shadow JSON file. State metadata in the device shadow JSON file contains the timestamp of each property. The timestamp is represented as epoch time to obtain exact update time.
timestamp	The latest update time of the device shadow JSON file.
version	When you request updating the version of the device shadow, the device shadow checks whether the requested version is later than the current version. If the requested version is later than the current one, the device shadow updates to the requested version. If not, the device shadow rejects the request. The version number is increased according to the version update to ensure the latest device shadow JSON file version.

Example of the device shadow JSON file:

```
{
  " state " : {
    " desired " : {
      " color " : " RED ",
      " sequence " : [ " RED ", " GREEN ", " BLUE " ]
    },
    " reported " : {
      " color " : " GREEN "
    }
  },
  " metadata " : {
    " desired " : {
      " color " : {
        " timestamp " : 1469564492
      },
      " sequence " : {
        " timestamp " : 1469564492
      }
    },
    " reported " : {
      " color " : {
        " timestamp " : 1469564492
      }
    }
  }
}
```

```
}  
}  
},  
"timestamp " : 1469564492 ,  
" version " : 1  
}
```

Empty properties

- The device shadow JSON file contains the desired property only when you have specified the desired status. The following device shadow JSON file, which does not contain the desired property, is also effective:

```
{  
  " state " : {  
    " reported " : {  
      " color " : " red ",  
    }  
  },  
  " metadata " : {  
    " reported " : {  
      " color " : {  
        " timestamp " : 1469564492  
      }  
    }  
  },  
  " timestamp " : 1469564492 ,  
  " version " : 1  
}
```

- The following device shadow JSON file, which does not contain the reported property, is also effective:

```
{  
  " state " : {  
    " desired " : {  
      " color " : " red ",  
    }  
  },  
  " metadata " : {  
    " desired " : {  
      " color " : {  
        " timestamp " : 1469564492  
      }  
    }  
  },  
  " timestamp " : 1469564492 ,  
  " version " : 1  
}
```

Array

The device shadow JSON file can use an array, and must update this array as a whole when the update is required.

- **Initial status:**

```
{  
  "reported " : { " colors " : [ " RED ", " GREEN ", " BLUE " ] }  
}
```

- **Update:**

```
{  
  "reported " : { " colors " : [ " RED " ] }  
}
```

- **Final status:**

```
{  
  "reported " : { " colors " : [ " RED " ] }  
}
```

6.3 Device shadow data stream

IoT Platform predefines two topics for each device to enable data transmission. The predefined topics have fixed formats.

- **Topic:** `/shadow/update/${YourProductKey}/${YourDeviceName}`

Devices and applications publish messages to this topic. When IoT Platform receives messages from this topic, it will extract the status information in the messages and will update the status to the device shadow.

- **Topic:** `/shadow/get/${YourProductKey}/${YourDeviceName}`

The device shadow updates the status to this topic, and the device subscribes to the messages from this topic.

Take a lightbulb device of a product `bulb_1` as an example to introduce the communication among devices, device shadows, and applications. In the following example, the ProductKey is `aliDeEf****` and the DeviceName is `lightbulb`. The device publishes messages to and subscribes to messages of the two custom topics using the method of QoS 1.

Device reports status automatically

The flow chart is shown in [Figure 6-1: Device reports status automatically](#).

Figure 6-1: Device reports status automatically

1. When the lightbulb is online, the device uses topic `/ shadow / update / aliDeEf ****/ lightbulb` to report the latest status to the device shadow.

Format of the JSON message:

```
{
  " method ": " update ",
  " state ": {
    " reported ": {
      " color ": " red "
    }
  },
  " version ": 1
}
```

The JSON parameters are described in [Table 6-2: Parameter description](#).

Table 6-2: Parameter description

Parameter	Description
method	The operation type when a device or application requests the device shadow. When you update the status, This parameter <code>method</code> is required and must be set to <code>update</code> .
state	The status information that the device sends to the device shadow. The <code>reported</code> field is required. The status information is synchronized to the reported field of the device shadow.
version	The version information contained in the request. The device shadow only accepts the request and updates to the specified version when the new version is later than the current version.

2. When the device shadow accepts the status reported by the device lightbulb, the JSON file of device shadow is successfully updated.

```
{
  "state" : {
    "reported" : {
      "color" : "red"
    }
  },
  "metadata" : {
    "reported" : {
      "color" : {
        "timestamp" : 1469564492
      }
    }
  },
  "timestamp" : 1469564492
  "version" : 1
}
```

3. After the device shadow has been updated, it will return the result to the device (lightbulb) by sending a message to the topic `/ shadow / get / aliDeEf ****/ lightbulb`.

- If the update is successful, the message is as follows:

```
{
  "method ":" reply ",
  "payload" : {
    "status ":" success ",
    "version" : 1
  },
  "timestamp" : 1469564576
}
```

- If an error occurred during the update, the message is as follows:

```
{
  "method ":" reply ",
  "payload" : {
    "status ":" error ",
    "content" : {
      "errorcode" : "${ errorcode }",
      "errormessage" : "${ errormessage }"
    }
  },
  "timestamp" : 1469564576
}
```

```
}
```

Error codes are described in [Table 6-3: Error codes](#).

Table 6-3: Error codes

errorCode	errorMessage
400	Incorrect JSON file.
401	The method field is not found.
402	the state field is not found.
403	Invalid version field.
404	The reported field is not found.
405	The reported field is empty.
406	Invalid method field.
407	The JSON file is empty.
408	The reported field contains more than 128 attributes.
409	Version conflict.
500	Server exception.

Application changes device status

The flow chart is shown in [Figure 6-2: Application changes device status](#).

Figure 6-2: Application changes device status

1. The application sends a command to the device shadow to change the status of the lightbulb.

The application sends a message to topic `/ shadow / update / aliDeEf **** / lightbulb /`. The message is as follows:

```
{
  "method ": " update ",
  "state ": {
    "desired ": {
      "color ": " green "
    }
  },
  "version ": 2
}
```

```
}
```

2. The application sends an update request to update the device shadow JSON file.

The device shadow JSON file is changed to:

```
{
  "state " : {
    "reported " : {
      "color " : " red "
    },
    "desired " : {
      "color " : " green "
    }
  },
  "metadata " : {
    "reported " : {
      "color " : {
        "timestamp " : 1469564492
      }
    },
    "desired " : {
      "color " : {
        "timestamp " : 1469564576
      }
    }
  },
  "timestamp " : 1469564576 ,
  "version " : 2
}
```

3. After the update, the device shadow sends a message to the topic `/ shadow / get / aliDeEf ****/ lightbulb` and returns the result of update to the device. The result message is created by the device shadow.

```
{
  "method ":" control ",
  "payload " : {
    "status ":" success ",
    "state " : {
      "reported " : {
        "color " : " red "
      },
      "desired " : {
        "color " : " green "
      }
    },
    "metadata " : {
      "reported " : {
        "color " : {
          "timestamp " : 1469564492
        }
      },
      "desired " : {
        "color " : {
          "timestamp " : 1469564576
        }
      }
    }
  }
}
```

```

},
"version ": 2 ,
"timestamp ": 1469564576
}

```

4. When the device lightbulb is online and has subscribed to the topic `/ shadow / get / aliDeEf ****/ lightbulb`, the device receives the message and changes its color to green according to the `desired` field in the request file. After the device has updated the status, it will report the latest status to the cloud.

```

{
  method ": " update ",
  " state ": {
    " reported ": {
      " color ": " green "
    }
  },
  "version ": 3
}

```

If the timestamp shows that the command has expired, you give up the update.

5. After the latest status has been reported successfully, the device sends a message to the topic `/ shadow / update / aliDeEf ****/ lightbulb` to empty the property of desired field. The message is as follows:

```

{
  " method ": " update ",
  " state ": {
    " desired ":" null "
  },
  "version ": 4
}

```

6. After the status has been reported, the device shadow is synchronously updated. The device shadow JSON file is as follows:

```

{
  " state " : {
    " reported " : {
      " color " : " green "
    }
  },
  "metadata " : {
    " reported " : {
      " color " : {
        " timestamp " : 1469564577
      }
    },
    " desired " : {
      " timestamp " : 1469564576
    }
  },
  "version " : 4
}

```

```
}
```

Devices request for device shadows

The flow chart is shown in [Figure 6-3: The device requests for device shadow](#).

Figure 6-3: The device requests for device shadow

1. The device lightbulb sends a message to the topic `/ shadow / update / aliDeEf *****/ lightbulb` and obtains the latest status saved in the device shadow. The message is as follows:

```
{
  " method ": " get "
}
```

2. When the device shadow receives above message, the device shadow sends a message to the topic `/ shadow / get / aliDeEf *****/ lightbulb`. The message is as follows:

```
{
  " method ":" reply ",
  " payload ": {
    " status ":" success ",
    " state ": {
      " reported ": {
        " color ": " red "
      },
      " desired ": {
        " color ": " green "
      }
    },
    " metadata ": {
      " reported ": {
        " color ": {
          " timestamp ": 1469564492
        }
      },
      " desired ": {
        " color ": {
          " timestamp ": 1469564492
        }
      }
    },
    " version ": 2 ,
    " timestamp ": 1469564576
  }
}
```

```
}
```

Devices delete device shadow attributes

The flow chart is shown in [Figure 6-4: Delete device shadow attributes](#).

Figure 6-4: Delete device shadow attributes

The device lightbulb is to delete the specified attributes saved in the device shadow.

The device sends a JSON message to the topic `/ shadow / update / aliDeEf ****/ lightbulb` . See the message in the following example.

To delete attributes, set the value of `method` to `delete` and set the values of the attributes to `null` .

- Delete one attribute:

```
{
  "method ": " delete ",
  " state ": {
    " reported ": {
      " color ": " null ",
      " temperatur e ":" null "
    }
  },
  "version ": 1
}
```

- Delete all the attributes:

```
{
  "method ": " delete ",
  " state ": {
    " reported ":" null "
  },
  "version ": 1
}
```

7 Configure the NTP service

IoT Platform provides the NTP service to resolve the following issues on embedded devices: limited resources, no NTP service available in the system, and inaccurate timestamp.

How NTP works

Based on the NTP protocol, IoT Platform acts as the NTP server. A device sends a message of a specific topic to IoT Platform with the sending time in the message payload. IoT Platform adds the message receiving time and response sending time to the payload of the response packet. After the device receives the response, the device records its local time when it receives the response. All these four time will be used to calculate the time difference between the device and IoT Platform to obtain the exact current time on the device.



Note:

The NTP service can be used for time calibration only after the device is connected to IoT Platform.

An embedded device, which does not have an accurate time after it is powered, cannot pass the certificate verification during the TLS connection establishment process. If it does not connect to IoT Platform, this issue cannot be resolved by the NTP service of IoT Platform.

NTP service procedure

Request topic: `/ ext / ntp / ${ YourProduc tKey } / ${ YourDevice Name } / request`

Response topic: `/ ext / ntp / ${ YourProduc tKey } / ${ YourDevice Name } / response`



Note:

ProductKey and DeviceName are part of the device certificate, which can be obtained from the IoT Platform console.

1. The device subscribes to the topic: `/ ext / ntp / ${ YourProduc tKey } / ${ YourDevice Name } / response` .

- The device publishes a QoS 0 message with the current timestamp of the device in the payload to the topic `/ ext / ntp / ${ YourProductKey } / ${ YourDeviceName } / request` . For example:

```
{
  " deviceSend  Time ":" 100 "
}
```



Note:

The data type of the timestamp, which supports Long and String.

Only QoS 0 messages are supported for this feature.

- The device receives a response from the NTP server. The payload includes the following information:

```
{
  " deviceSend  Time ":" 100 ",
  " serverRecv  Time ":" 1010 ",
  " serverSend  Time ":" 1015 ",
}
```

- The device calculates the current exact Unix time.

The time when the device receives the message from the server is recorded as `${deviceRecvTime}`, and the exact time on the device is: $(\${Serverrecv\ time} + \${serversend\ time} + \${deviceRecv\ time} - \${devicesend\ time}) / 2$

Example

In this example, the device time is 100, the server time is 1000, the network delay is 10, and the time spent before the server sends a response for a received request is 5.

	Device time	Server time
deviceSend	100 (deviceSendTime)	1000
serverReceive	110	1010 (serverRecvTime)
serverSend	115	1015 (serverSendTime)
deviceReceive	125 (deviceRecvTime)	1025

The device calculates the current exact Unix time as $(1010 + 1015 + 125 - 100) / 2 = 1025$.

The current server time is 1015. If the device directly uses the timestamp returned from the server, the device will have a time error due to the network delay.

8 Accounts and logon

This topic describes IoT Platform accounts and how to log on to the IoT Platform console.

8.1 Log on to the console using the primary account

The primary account has full operation permissions on all resources under this account, and supports modifying account information.

Log on to the IoT Platform console using the primary account

You have full operation permissions on IoT Platform when logging on to the console using the primary account.

1. Visit the [Alibaba Cloud official website](#).
2. Click Console.
3. Log on to the console using your account and password.



Note:

To retrieve an account or password, click **Forgot Username** or **Forgot Password** on the logon page to start the retrieval process.

4. Click **Products** in the console to display all products and services that are provided by Alibaba Cloud.
5. Search for **IoT Platform**, and click **IoT Platform** in the result to enter the IoT Platform console.



Note:

If you have not activated the IoT Platform service, the IoT Platform console prompts you to activate this service on the homepage. Click **Activate Now** to activate it quickly.

After entering the IoT Platform console, you can manage products, devices, and rules.

Create access control using the primary account

The primary account has full permissions, so the leakage of the primary account may cause serious security risks. Therefore, do not disclose your account and password when you authorize others to access your Alibaba Cloud resources. Instead, you should use Resource Access Management (RAM) to create sub-accounts and assign

the required access permissions to these sub-accounts. All users except the primary account user or administrator access the resources using sub-accounts. For more information about accessing IoT Platform using RAM users, see [Use RAM users](#) and [Custom permissions](#).

8.2 Resource Access Management (RAM)

This chapter describes IoT Platform access control.

8.2.1 RAM and STS

Resource Access Management (RAM) and Security Token Service (STS) are access control systems provided by Alibaba Cloud. For more information about RAM and STS, see [RAM help documentation](#).

RAM is used to control the permissions of accounts. By using RAM, you can create and manage RAM users. You can control what resources RAM users can access by granting different permissions to them.

STS is a security token management system. It is used to manage the short-term permissions granted to RAM users. You can use STS to grant permissions to temporary users.

Background

RAM and STS enable you to securely grant permissions to users without exposing your account AccessKey. Once your account AccessKey is exposed, your resources will be exposed to major security risks. Individuals who obtain your AccessKey can perform any operation on the resources under your account and steal personal information.

RAM is a mechanism used to control long-term permissions. After creating RAM users, you can grant them different permissions. AccessKeys of RAM users if exposed do not have the same risk as an account AccessKey being exposed. If the AccessKey of any RAM user is exposed, information potentially exposed is limited. RAM users are valid for a long term.

Unlike RAM, which allows you to grant long-term permissions to users, STS enables you to grant users temporary access. By calling the STS API, you can obtain temporary AccessKeys and tokens. You can assign the temporary AccessKeys and tokens to RAM users so they can access specific resources. Permissions obtained from

STS are strictly restricted and have limited validity. Therefore, even if information is unexpectedly exposed, your system will not be severely compromised.

For details about how to use RAM and STS, see *Examples*.

Concepts

Before you use RAM and STS, we recommend that you have a basic understanding of the following concepts:

- **RAM user:** A user that is created using the RAM console. During or after the creation of a RAM User, an AccessKey can be generated for the RAM user. After creating a RAM user, you need to configure the password and grant permissions to it. Once this is completed the RAM user can perform authorized operations. A RAM user can be considered a user with specific operation permissions.
- **Role:** A virtual entity that represents a group of permissions. Roles do not have their own logon password or AccessKey. A RAM user can assume roles. When roles are assumed the RAM user has the associated role privileges.
- **Policy:** A policy defines permissions. For example, a policy defines the permission of a RAM user to read or write to specific resources.
- **Resource:** Cloud resources that are accessible to a RAM user, such as all Table Store instances, a Table Store instance, or a table in a Table Store instance.

The relationship between RAM users and their roles is similar to the relationship between individuals and their identities. For example, the roles of a person might be an employee at work and a father at home. A person plays different roles in different scenarios. When playing a specific role, the person has the privileges of that role. A role itself is not an operational entity. Only after the user has assumed this role is it a complete operational entity. A role can be assumed by multiple users.

Examples

To prevent an account from being exposed to security risks if the account AccessKey is exposed, an account administrator creates two RAM users. These RAM users are named A and B. An AccessKey is generated for each of them. A has the read permission, and B has the write permission. The administrator can revoke the permissions from the RAM users at any time in the RAM console.

Additional, individuals need to be granted temporary access to the API of IoT Platform. In this case, the AccessKey of A must not be disclosed. Instead, the

administrator needs to create a role, C, and grant this role access to the API of IoT Platform. Note that C cannot be directly used currently because there is no AccessKey for C, and C is only a virtual entity that owns access to the IoT Platform API.

The administrator needs to call the AssumeRole API operation of STS to obtain temporary security credentials that can be used to access the IoT Platform API. In the AssumeRole call, the value of RoleArn must be the Alibaba Cloud resource name (ARN) of C. If the AssumeRole call is successful, STS will return a temporary AccessKeyId, AccessKeySecret, and SecurityToken as security credentials. The validity period of these credentials can be specified when AssumeRole is called. The account administrator can deliver these credentials to users who need access to the API of the IoT Platform. This access to the API is temporary.

Why is it complicated to use RAM and STS?

The concepts and use of RAM and STS are complicated. This ensures account security and flexible access control at the cost of service ease of use.

RAM users and roles are separated in order to keep the entity that performs operation separate from the virtual entity that represents a group of permissions. If a user needs multiple permissions, such as the read and the write permissions, but in fact the user only needs one permission at a time, you can create two roles. Grant the read permission and the write permission to these two roles, respectively. Then create a RAM user and assign both roles to the RAM user. When the RAM user needs the read permission, assume the role that includes the read permission. When the RAM user needs the write permission, assume the role that includes the write permission. This reduces the risk of a permission leak occurring in each operation. Additionally, you can assign roles to other accounts and RAM users to grant them the permissions included in the roles. This makes it easier for users to use the role permissions.

STS allows more flexible access control. For example, you can configure the validity period for credentials. However, if long-term credentials are required, you can only use RAM to manage RAM users.

The following sections provide guidelines for using RAM and STS and examples for using them. For more information about APIs provided by RAM and STS, see [API Reference - RAM](#) and [API Reference - STS](#).

8.2.2 Custom permissions

Permissions define the conditions in which the system allows or denies some specified actions on target resources.

Permissions are defined in authorization policies. Custom permissions allow you to define certain permissions by using custom authorization policies. In the Resource Access Management (RAM) console, click **Create Authorization Policy** on the **Policies** page to customize an authorization policy. Select a blank template when customizing an authorization policy.

An authorization policy is a JSON string that requires the following parameters:

- **Action** : Indicates the action that you want to authorize. IoT actions start with `iot:`. For more information about actions and examples, see [Define actions](#).
- **Effect** : Indicates the authorization type, which can be `Allow` or `Deny` .
- **Resource** : Because IoT Platform does not support resource authorization, enter an asterisk `*` instead.
- **Condition** : Indicates the authentication condition. For more information, see [Define conditions](#).

Define actions

Action is an application programming interface (API) operation name. When creating an authorization policy, use `iot:` as the prefix for each action, and separate multiple actions with commas (,). You can also use an asterisk (*) as a wildcard character. For more information about API name definitions that are used on IoT Platform, see [API permissions](#) .

The following are some examples of action definitions.

- Define a single API operation.

```
" Action ": " iot : CreateProd uct "
```

- Define multiple API operations.

```
" Action ": [  
  " iot : UpdateProd uct ",  
  " iot : QueryProdu ct "  
]
```

- Define all read-only API operations.

```
{  
  " Version ": " 1 ",
```

```

" Statement ": [
{
  " Action ": [
    " iot : Query *",
    " iot : List *",
    " iot : Get *",
    " iot : BatchGet *",
    " iot : Check *"
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": [
    " rds : DescribeDB Instances ",
    " rds : DescribeDa tabases ",
    " rds : DescribeAc counts ",
    " rds : DescribeDB InstanceNe tInfo "
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": " ram : ListRoles ",
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": [
    " mns : ListTopic ",
    " mns : GetTopicRe f "
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": [
    " ots : ListInstan ce ",
    " ots : GetInstanc e ",
    " ots : ListTable ",
    " ots : DescribeTa ble "
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": [
    " fc : ListServic es ",
    " fc : GetService ",
    " fc : GetFunctio n ",
    " fc : ListFuncti ons "
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},
{
  " Action ": [
    " log : ListShards ",
    " log : ListLogSto res ",
    " log : ListProjec t "
  ],
  " Resource ": "*",
  " Effect ": " Allow "
},

```

```

    {
      " Action ": [
        " cms : QueryMetri cList "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    }
  ]
}

```

- Define all read-write API operations.

```

{
  " Version ": " 1 ",
  " Statement ": [
    {
      " Action ": " iot :*",
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": [
        " rds : DescribeDB Instances ",
        " rds : DescribeDa tabases ",
        " rds : DescribeAc counts ",
        " rds : DescribeDB InstanceNe tInfo ",
        " rds : ModifySecu rityIps "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": " ram : ListRoles ",
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": [
        " mns : ListTopic ",
        " mns : GetTopicRe f "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": [
        " ots : ListInstan ce ",
        " ots : ListTable ",
        " ots : DescribeTa ble ",
        " ots : GetInstanc e "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": [
        " fc : ListServic es ",
        " fc : GetService ",
        " fc : GetFunctio n ",
        " fc : ListFuncti ons "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    }
  ]
}

```

```

    },
    {
      " Action ": [
        " log : ListShards ",
        " log : ListLogSto res ",
        " log : ListProjec t "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    },
    {
      " Action ": " ram : PassRole ",
      " Resource ": "*",
      " Effect ": " Allow ",
      " Condition ": {
        " StringEqua ls ": {
          " acs : Service ": " iot . aliyuncs . com "
        }
      }
    },
  ],
  {
    {
      " Action ": [
        " cms : QueryMetri cList "
      ],
      " Resource ": "*",
      " Effect ": " Allow "
    }
  ]
}

```

Define conditions

RAM authorization policies currently support multiple authentication conditions, such as the access IP address restrictions, the Hypertext Transfer Protocol Secure (HTTPS)-based access enabler, the multi-factor authentication (MFA)-based access enabler, and access time restrictions. All API operations on IoT Platform support these authentication conditions.

Access control based on source IP addresses

This access control restricts source IP addresses that can access IoT Platform, and supports filtering by Classless Inter-Domain Routing (CIDR) blocks. Typical scenarios are described as follows:

- Apply access control rules to a single IP address or CIDR blocks. For example, the following code indicates that only access requests from IP address 10.101.168.111 or 10.101.169.111/24 are allowed.

```

{
  " Statement ": [
    {
      " Effect ": " Allow ",
      " Action ": " iot :*",
      " Resource ": "*",
      " Condition ": {

```

```

" IPAddress ": {
" acs : SourceIp ": [
" 10 . 101 . 168 . 111 ",
" 10 . 101 . 169 . 111 / 24 "
]
},
" Version ": " 1 "
}

```

- Apply access control rules to multiple IP addresses. For example, the following code indicates that only access requests from IP addresses 10.101.168.111 and 10.101.169.111 are allowed.

```

{
" Statement ": [
{
" Effect ": " Allow ",
" Action ": " iot :*",
" Resource ": " *",
" Condition ": {
" IPAddress ": {
" acs : SourceIp ": [
" 10 . 101 . 168 . 111 ",
" 10 . 101 . 169 . 111 "
]
}
}
}
],
" Version ": " 1 "
}

```

HTTPS-based access control

This access control allows you to enable or disable HTTPS-based access.

For example, the following code indicates that only HTTPS-based access is allowed.

```

{
" Statement ": [
{
" Effect ": " Allow ",
" Action ": " iot :*",
" Resource ": " *",
" Condition ": {
" Bool ": {
" acs : SecureTransport ": " true "
}
}
}
],
" Version ": " 1 "
}

```

MFA-based access control

This access control allows you to enable or disable MFA-based access.

For example, the following code indicates that only MFA-based access is allowed.

```
{
  " Statement ": [
    {
      " Effect ": " Allow ",
      " Action ": " iot :*",
      " Resource ": "*",
      " Condition ": {
        " Bool ": {
          " acs : MFAPresent ": " true "
        }
      }
    }
  ],
  " Version ": " 1 "
}
```

Access time restrictions

This access control allows you to limit the access time of requests. Access requests earlier than the specified time are allowed or rejected.

For example, the following code indicates that only access requests earlier than 00:00:00 Beijing Time (UTC+8) on January 1, 2019 are allowed.

```
{
  " Statement ": [
    {
      " Effect ": " Allow ",
      " Action ": " iot :*",
      " Resource ": "*",
      " Condition ": {
        " DateLessThan ": {
          " acs : CurrentTime ": " 2019 - 01 - 01T00 : 00 : 00 + 08 : 00 "
        }
      }
    }
  ],
  " Version ": " 1 "
}
```

Typical scenarios

Based on these definitions of actions, resources, and conditions, authorization policies are described in the following typical scenarios.

The following is an example of authorization policy that allows access.

Scenario: Assigns IoT Platform access permissions to the IP address 10.101.168.111 /24, and only allows HTTPS-based access before 00:00:00 Beijing Time (UTC+8) on January 1, 2019.

```
{
  "Statement ": [
    {
      "Effect ": " Allow ",
      " Action ": " iot :*",
      " Resource ": ".*",
      " Condition ": {
        " IPAddress ":{
          " acs : SourceIp ": [
            " 10 . 101 . 168 . 111 / 24 "
          ]
        },
        " DateLessThan ": {
          " acs : CurrentTime ": " 2019 - 01 - 01T00 : 00 : 00 + 08 : 00 "
        }
      },
      " Bool ": {
        " acs : SecureTransport ": " true "
      }
    }
  ],
  "Version ": " 1 "
}
```

The following is an example of authorization policy to specify denied access.

Scenario: Rejects read requests from IP address 10.101.169.111.

```
{
  "Statement ": [
    {
      "Effect ": " Deny ",
      " Action ": [
        " iot : Query *",
        " iot : List *",
        " iot : Get *",
        " iot : BatchGet *"
      ],
      " Resource ": ".*",
      " Condition ": {
        " IPAddress ": {
          " acs : SourceIp ": [
            " 10 . 101 . 169 . 111 "
          ]
        }
      }
    }
  ],
  "Version ": " 1 "
}
```

After creating the authorization policy, apply this policy to the RAM users on the User Management page in the RAM console. Authorized RAM users can perform the

operations defined in this policy. For more information about creating RAM users and granting permissions, see [Use RAM users](#).

8.2.3 API permissions

Each operation in the following table represents the value of `Action` that you specify when creating authentication policies for RAM users.

For more information about creating authentication policies for RAM users, see [Custom permissions](#).

Operations	RAM action	Resource	Description
CreateProduct	iot:CreateProduct	*	Create a product.
UpdateProduct	iot:UpdateProduct	*	Update product information
QueryProduct	iot:QueryProduct	*	Query the detailed information of a product.
QueryProductList	iot:QueryProductList	*	Query all the products.
DeleteProduct	iot>DeleteProduct	*	Delete a product.
CreateProductTags	iot:CreateProductTags	*	Create product tags.
UpdateProductTags	iot:UpdateProductTags	*	Update product tags.
DeleteProductTags	iot>DeleteProductTags	*	Delete product tags.
ListProductTags	iot:ListProductTags	*	Query tags of a product.
ListProductByTags	iot:ListProductByTags	*	Query products by tags.
RegisterDevice	iot:RegisterDevice	*	Register a device.
QueryDevice	iot:QueryDevice	*	Query all the devices of a specified product.
DeleteDevice	iot>DeleteDevice	*	Delete a device.
QueryPageByApplyId	iot:QueryPageByApplyId	*	Query the information of devices that are registered at a time.
BatchGetDeviceState	iot:BatchGetDeviceState	*	Query the status of multiple devices at a time.
BatchRegisterDeviceWithApplyId	iot:BatchRegisterDeviceWithApplyId	*	Register multiple devices simultaneously using a given application ID.

Operations	RAM action	Resource	Description
BatchRegisterDevice	iot:BatchRegisterDevice	*	Register multiple devices at a time (not specify device names).
QueryBatchRegisterDeviceStatus	iot:QueryBatchRegisterDeviceStatus	*	Query the processing status and result of device registration of multiple devices.
BatchCheckDeviceNames	iot:BatchCheckDeviceNames	*	Specify device names in batch.
QueryDeviceStatistics	iot:QueryDeviceStatistics	*	Query device statistics.
QueryDeviceEventData	iot:QueryDeviceEventData	*	Query the historical records of a device event.
QueryDeviceServiceData	iot:QueryDeviceServiceData	*	Query the historical records of a device service.
SetDeviceProperty	iot:SetDeviceProperty	*	Set properties for a specified device.
SetDevicesProperty	iot:SetDevicesProperty	*	Set properties for multiple devices.
InvokeThingService	iot:InvokeThingService	*	Invoke a service on a device.
InvokeThingsService	iot:InvokeThingsService	*	Invoke a service on multiple devices.
QueryDevicePropertyStatus	iot:QueryDevicePropertyStatus	*	Query the property snapshots of a device.
QueryDeviceDetail	iot:QueryDeviceDetail	*	Query the detailed information of a device.
DisableThing	iot:DisableThing	*	Disable a device.
EnableThing	iot:EnableThing	*	Enable a device that has been disabled.
GetThingTopo	iot:GetThingTopo	*	Query the topological relationships of a device.
RemoveThingTopo	iot:RemoveThingTopo	*	Delete the topological relationships of a device.

Operations	RAM action	Resource	Description
NotifyAddThingTopo	iot:NotifyAddThingTopo	*	Notify a gateway device to add topological relationships with specified sub-devices.
QueryDevicePropertyData	iot:QueryDevicePropertyData	*	Query the historical records of a device property.
QueryDevicePropertiesData	iot:QueryDevicePropertiesData	*	Query the historical records of device properties.
GetGatewayBySubDevice	iot:GetGatewayBySubDevice	*	Query the gateway device information using the sub-device information.
SaveDeviceProp	iot:SaveDeviceProp	*	Create tags for a device.
QueryDeviceProp	iot:QueryDeviceProp	*	Query all the tags of a device.
DeleteDeviceProp	iot>DeleteDeviceProp	*	Delete a tag of a device.
QueryDeviceByTags	iot:QueryDeviceByTags	*	Query devices by tags.
CreateDeviceGroup	iot>CreateDeviceGroup	*	Create a device group.
UpdateDeviceGroup	iot:UpdateDeviceGroup	*	Update the information of a device group.
DeleteDeviceGroup	iot>DeleteDeviceGroup	*	Delete a device group.
BatchAddDeviceGroupRelations	iot:BatchAddDeviceGroupRelations	*	Add devices to a group.
BatchDeleteDeviceGroupRelations	iot:BatchDeleteDeviceGroupRelations	*	Delete devices from a group.
QueryDeviceGroupInfo	iot:QueryDeviceGroupInfo	*	Query the detailed information of a group.
QueryDeviceGroupList	iot:QueryDeviceGroupList	*	Query all the device groups.
SetDeviceGroupTags	iot:SetDeviceGroupTags	*	Create, update, or delete tags of a group.
QueryDeviceGroupTagList	iot:QueryDeviceGroupTagList	*	Query all the tags of a group.

Operations	RAM action	Resource	Description
QueryDeviceGroupByDevice	iot:QueryDeviceGroupByDevice	*	Query the groups that a specified device is in.
QueryDeviceListByDeviceGroup	iot:QueryDeviceListByDeviceGroup	*	Query devices in a device group.
QuerySuperDeviceGroup	iot:QuerySuperDeviceGroup	*	Query the parent group of a device group.
QueryDeviceGroupByTags	iot:QueryDeviceGroupByTags	*	Query device groups by tags.
StartRule	iot:StartRule	*	Enable a rule.
StopRule	iot:StopRule	*	Stop a rule.
ListRule	iot:ListRule	*	Query all the rules.
GetRule	iot:GetRule	*	Query the details of a rule.
CreateRule	iot:CreateRule	*	Create a rule.
UpdateRule	iot:UpdateRule	*	Update the information of a rule.
DeleteRule	iot>DeleteRule	*	Delete a rule.
CreateRuleAction	iot:CreateRuleAction	*	Create a data forwarding method for a rule.
UpdateRuleAction	iot:UpdateRuleAction	*	Update a data forwarding method.
DeleteRuleAction	iot>DeleteRuleAction	*	Delete a data forwarding method.
GetRuleAction	iot:GetRuleAction	*	Query the detailed information of a data forwarding method.
ListRuleActions	iot:ListRuleActions	*	Query all the data forwarding methods in a rule.
Pub	iot:Pub	*	Publish a message.
PubBroadcast	iot:PubBroadcast	*	Publish a message to the devices that have subscribed to a broadcast topic.

Operations	RAM action	Resource	Description
RRpc	iot:RRpc	*	Send a message to a device and receive a response from the device.
CreateProductTopic	iot:CreateProductTopic	*	Create a topic category for a product.
DeleteProductTopic	iot>DeleteProductTopic	*	Delete a topic category.
QueryProductTopic	iot:QueryProductTopic	*	Query all the topic categories of a product.
UpdateProductTopic	iot:UpdateProductTopic	*	Update a topic category.
CreateTopicRouteTable	iot:CreateTopicRouteTable	*	Create message routing relationships between topics.
DeleteTopicRouteTable	iot>DeleteTopicRouteTable	*	Delete message routing relationships between topics.
QueryTopicReverseRouteTable	iot:QueryTopicReverseRouteTable	*	Query the source topic of a target topic.
QueryTopicRouteTable	iot:QueryTopicRouteTable	*	Query the target topics of a source topic.
GetDeviceShadow	iot:GetDeviceShadow	*	Query the shadow information of a device.
UpdateDeviceShadow	iot:UpdateDeviceShadow	*	Update the shadow information of a device.
SetDeviceDesiredProperty	iot:SetDeviceDesiredProperty	*	Set desired property values for a device.
QueryDeviceDesiredProperty	iot:QueryDeviceDesiredProperty	*	Query the desired property values of a specified device.
BatchUpdateDeviceNickname	iot:BatchUpdateDeviceNickname	*	Update nicknames for multiple devices.
QueryDeviceFileList	iot:QueryDeviceFileList	*	Query the files that a specified device has uploaded to IoT Platform.

Operations	RAM action	Resource	Description
QueryDeviceFile	iot:QueryDeviceFile	*	Query a specified file that a specified device has uploaded to IoT Platform.
DeleteDeviceFile	iot>DeleteDeviceFile	*	Delete a specified file that a specified device has uploaded to IoT Platform.

8.2.4 Use RAM users

RAM users (sub-accounts) can log on to the IOT Platform console to manage IoT resources, and use the corresponding AccessKeyId and AccessKeySecret to use IoT application programming interface (API).

You need to create a RAM user first, and assign the permissions for accessing IoT Platform to this RAM user by using authorization policies. For more information about customizing authorization policies, see [Custom permissions](#).

Create a RAM user

Skip this step if you already have a RAM user.

1. Log on to the [RAM console](#).
2. In the left-side navigation pane, click Users.
3. Click Create User.
4. Enter user information, select Automatically generate an AccessKey for this user., and then click OK.



Note:

The system prompts you to save the AccessKey after you click OK. You can download this AccessKey only at this moment. You need to save this AccessKey and secure it immediately. The system requires the AccessKey when the corresponding RAM user calls API operations.

5. Set the initial login password.
 - a. On the User Management page, click Manage of the created RAM user to enter the User Details page.
 - b. Click Enable Console Logon.
 - c. Set an initial password for this RAM user, select On your next logon you must reset the password., and then click OK.

6. Enable multi-factor authentication (MFA). (Optional)

On the User Details page, click Enable VMFA Device.

After you create the RAM user, the RAM user can log on to the official website and the IoT Platform console by using the Resource Access Management (RAM) user logon link. To obtain the RAM user logon link, go to the RAM Overview page in the RAM console.

However, the RAM user cannot access your Alibaba Cloud resources before you grant permissions to the RAM user. Therefore, you need to assign permissions for accessing IoT Platform to this RAM user.

Authorize the RAM user to access IoT Platform

In the RAM console, assign permissions to a RAM user on the User Management page, or assign the same permissions to a group on the Group Management page. To assign permissions to a RAM user, follow these steps:

1. Log on to the [RAM console](#) using the primary account.
2. In the left-side navigation pane, click Users.
3. Click Authorize next to the RAM user that you want to assign permissions to.
4. In the authorization dialog box, select the authorization policy that you want to apply to this RAM user, click the right arrow in the middle of the page to move the selected authorization policy to Selected Authorization Policy Name, and then click OK.



Note:

To assign custom permissions to the RAM user, you need to create an authorization policy first. For more information about customizing an authorization policy, see [Custom permissions](#).

Edit User-Level Authorization
✕

Members added to this group have all the permissions of this group. A member cannot be added to the same group more than once.

Available Authorization Policy Names	Type		Selected Authorization Policy Name	Type
iot				
AliyunIOTFullAccess 管理物联网套件(IOT)的权限	System	➤ ➤		
AliyunDyiotFullAccess Provides full acce...	System			
AliyunDyiotReadOnlyAccess Provides read-only...	System			
AdministratorAccess Provides full acce...	System			

OK

Close

The authorized RAM user can access the resources defined in the authorization policy , and perform the specified operations.

Logon to the console using a RAM user

The primary account user can log on to the console from the official website.

However, the RAM user needs to log on to the console on the RAM User Logon page.

1. Obtain the link for logging on to the RAM User Logon page.

Log on to the RAM console using the primary account, view the RAM User Logon Link on the RAM Overview page, and then send this logon link to the RAM user.

2. The RAM user accesses the RAM User Logon page, and logs on to the console using the RAM user name and password.



Note:

The RAM user follows this logon format: RAM user name@company alias, such as username@company-alias. The RAM user also needs to change the logon password after logon for the first time.

3. Click Console in the upper-right corner of the page to go to the Home page.
4. Click Products, and select IoT Platform to go to the IoT Platform console.

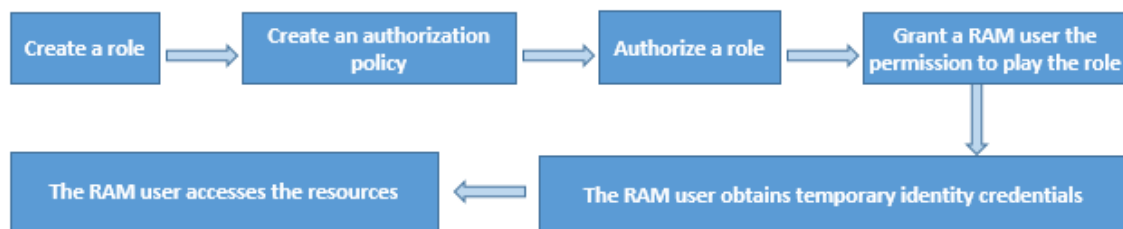
Then, the RAM user can perform authorized operations in the console.

8.2.5 Advanced guide to STS

Security Token Service (STS) enables more strict permission management than Resource Access Management (RAM). Using STS to implement resource access control involves a complicated authorization process. You can use STS to grant RAM users temporary permissions to access resources.

RAM users and the permissions granted to RAM users have long-term validity. You need to manually delete a RAM user or revoke permissions from RAM users. After the account information of a RAM user has been leaked, if you fail to timely delete this user or revoke related permissions, your Alibaba Cloud resources and important information may be compromised. Therefore, we recommend that you use STS to manage key permissions or permissions that do not require long-term validity.

Figure 8-1: Process for granting temporary permissions to RAM users.



Step 1: Create a role

A role is a virtual entity that represents a virtual user with a group of permissions.

1. Log on to the [RAM console](#).
2. Select Roles > Create Role to create a role.
3. Select User Role.
4. Use the default account information, and click Next.
5. Specify the role name and description, and click Create.

6. Click Close or Authorize.

If you have created the authorization policy that is to be granted to this role, click **Authorize** to authorize this user.

If you have not created the authorization policy, click **Close**. You can create an authorization policy for this role by clicking **Policies**.

Step 2: Create an authorization policy

An authorization policy defines the resource permissions that are to be granted to roles.

1. In the [RAM console](#), click **Policies > Create Authorization Policy**.
2. Select the blank template.
3. Specify the authorization policy name and policy content, and click **Create Authorization Policy**.

For more information about writing the policy content, click **Authorization Policy Format**.

Authorization policy example:Read-only permission of IoT resources.

```
{
  "Version ": " 1 ",
  "Statement ": [
    {
      "Action ": [
        " rds : DescribeDB   Instances ",
        " rds : DescribeDa  tabases ",
        " rds : DescribeAc   counts ",
        " rds : DescribeDB   InstanceNe  tInfo "
      ],
      "Resource ": "* ",
      "Effect ": " Allow "
    },
    {
      "Action ": " ram : ListRoles ",
      "Effect ": " Allow ",
      "Resource ": "*"
    },
    {
      "Action ":[
        " mns : ListTopic "
      ],
      "Resource ": "* ",
      "Effect ": " Allow "
    },
    {
      "Action ": [
        " dhs : ListProjec  t ",
        " dhs : ListTopic ",
        " dhs : GetTopic "
      ],

```

```

" Resource ": "*",
" Effect ": " Allow "
},
{
" Action ": [
" ots : ListInstance ",
" ots : ListTable ",
" ots : DescribeTable "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Action ":[
" log : ListShards ",
" log : ListLogStores ",
" log : ListProject "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Effect ": " Allow ",
" Action ": [
" iot : Query *",
" iot : List *",
" iot : Get *",
" iot : BatchGet *"
],
" Resource ": "*"
}
]
}

```

Authorization policy example:Read-write permission of IoT resources.

```

{
" Version ": " 1 ",
" Statement ": [
{
" Action ": [
" rds : DescribeDBInstances ",
" rds : DescribeDatabases ",
" rds : DescribeAccounts ",
" rds : DescribeDBInstanceNetInfo "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Action ": " ram : ListRoles ",
" Effect ": " Allow ",
" Resource ": "*"
},
{
" Action ":[
" mns : ListTopic "
],
" Resource ": "*",
" Effect ": " Allow "
},
{

```

```

" Action ": [
" dhs : ListProjec t ",
" dhs : ListTopic ",
" dhs : GetTopic "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Action ": [
" ots : ListInstan ce ",
" ots : ListTable ",
" ots : DescribeTa ble "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Action ":[
" log : ListShards ",
" log : ListLogSto res ",
" log : ListProjec t "
],
" Resource ": "*",
" Effect ": " Allow "
},
{
" Effect ": " Allow ",
" Action ": " iot :*",
" Resource ": "*"
}
]
}

```

After an authorization policy has been created, you can grant the permissions defined in this policy to roles.

Step 3: Authorize a role

A role can only have resource access permissions after it has been authorized.

1. In the [RAM console](#), click Roles.
2. Select the role that you want to authorize, and click Authorize.
3. In the dialog box that appears, select the custom authorization policy that you want to apply to the specified role, click the right arrow in the middle to move the

specified authorization policy to the Selected Authorization Policy Name list, and then click OK.

Edit User-Level Authorization
✕

Members added to this group have all the permissions of this group. A member cannot be added to the same group more than once.

Available Authorization Policy Names	Type		Selected Authorization Policy Name	Type
iot				
AliyunIOTFullAccess 管理物联网套件(IOT)的权限	System	➤ ➤		
AliyunDyiotFullAccess Provides full acce...	System			
AliyunDyiotReadOnlyAccess Provides read-only...	System			
AdministratorAccess Provides full acce...	System			

OK

Close

The role will have the permissions defined in the selected authorization policy after authorization is complete. You can click Manage to go to the Role Details page, and view basic information about this role and the permissions it has been granted.

Next, you need to grant a RAM user the permission to play this role.

Step 4: Grant a RAM user the permission to play the role

After authorization is complete, the role obtains the permissions that are defined in the authorization policy. However, the role is only a virtual user. You need a RAM user to play the role in order to perform the operations allowed by the permissions. If all RAM users are allowed to play the role, this causes security risks. You should only grant the permission to play this role to specified RAM users.

To grant a RAM user the permission to play this role, you need to create a custom authorization policy where the `Resource` parameter of this policy is set to the ID of the role. You then authorize the RAM user with this authorization policy.

1. In the [RAM console](#), click Policies > Create Authorization Policy .
2. Select the blank template.

3. Specify the authorization policy name and policy content, and click Create Authorization Policy.

**Note:**

In the policy content, set the `Resource` parameter to the Arn of the role. On the Roles page, find the specified role, click Manage to go to the Role Details page, and then view the Arn of the role .

Role authorization policy example:

```
{
  "Version ": " 1 ",
  "Statement ": [
    {
      "Effect ": " Allow ",
      "Action ": " iot : QueryProdu ct ",
      "Resource ": " Role Arn "
    }
  ]
}
```

4. After the authorization policy has been created, go to the home page of the RAM console.
5. Click Users in the left-side navigation pane to enter RAM user management page.
6. Select the RAM user you want to authorize and click Authorize.
7. In the dialog box that appears, select the authorization policy that you have just created, click the right arrow in the middle to move the specified authorization policy to the Selected Authorization Policy Name list, and then click OK.

After authorization is complete, the RAM user obtains the permission to play this role . You can then use STS to obtain the temporary identity credentials for accessing the resources.

Step 5: The RAM user obtains temporary identity credentials

Authorized RAM users can call the STS API operations or use the STS SDKs to obtain the temporary identity credentials for role play. The temporary credentials include an AccessKeyId, AccessKeySecret, and SecurityToken. For more information about the STS API and STS SDKs, see [API Reference \(STS\)](#) and [SDK Reference \(STS\)](#).

You need to specify the following parameters when using an STS API or SDK to obtain temporary identity credentials:

- **RoleArn:** The Arn of the role that the RAM user is to play.

- **RoleSessionName:** The name of the temporary credentials. This is a custom parameter.
- **Policy:** The authorization policy. This parameter adds a restriction to the permissions of the role. You can use this parameter to restrict the permissions of the token. If you do not specify this parameter, a token possessing all permissions of the specified role is created.
- **DurationSeconds:** The validity period of the temporary credentials. This parameter is measured in seconds. The default value is 3,600 and the value ranges from 900 to 3,600.
- **id and secret:** The `AccessKeyId` and `AccessKeySecret` of the RAM user.

Examples of obtaining temporary identity credentials

API example: The RAM user calls the STS `AssumeRole` operation to obtain the temporary identity credentials for role play.

```
https://sts.aliyuncs.com?Action=AssumeRole
&RoleArn=acs:ram::1234567890:role/iotstsrole
&RoleSessionName=iotreadonlyrole
&DurationSeconds=3600
&Policy=<url_encode_d_policy>
&<Common request parameters>
```

SDK example: The RAM user obtains the temporary identity credentials through the Python CLI interface for STS.

```
$ python ./sts.py AssumeRole RoleArn=acs:ram::
1234567890:role/iotstsrole RoleSessionName=
iotreadonlyrole Policy='{"Version":"1","Statement":
[{"Effect":"Allow","Action":"iot:*","Resource":"*"}]}'
DurationSeconds=3600 --id=id --secret=secret
```

After the request has been received, the temporary identity credentials that are required to play the role are returned. The credentials include an `AccessKeyId`, `AccessKeySecret`, and `SecurityToken`.

Step 6: The RAM user accesses the resources

After obtaining the temporary identity credentials, the RAM user can pass in the credentials in the SDK requests to play the specified role.

Java SDK example: The RAM user passes in the AccessKeyId, AccessKeySecret, and SecurityToken parameters that are contained in the temporary identity credentials in the request and creates the IAcsClient object.

```
IClientProfile profile = DefaultProfile.getProfile("cn -
hangzhou", AccessKeyId, AccessSecret);
RpcAcsRequest request = new RpcAcsRequest("SecurityToken",
"UTF-8", "JSON");
IAcsClient client = new DefaultAcsClient(profile);
AcsResponse response = client.getAcsResponse(request);
```