Alibaba Cloud
IoT Platform

User Guide

Issue: 20190311

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 informatio n, supplementary instructions, and other content that the user must understand.	ⓘ Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	📋 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus , page names, and other UI elements.	Click OK.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *{stand | slave}*

Contents

1 Create products and devices

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

1.1 Create a product (Basic Edition)

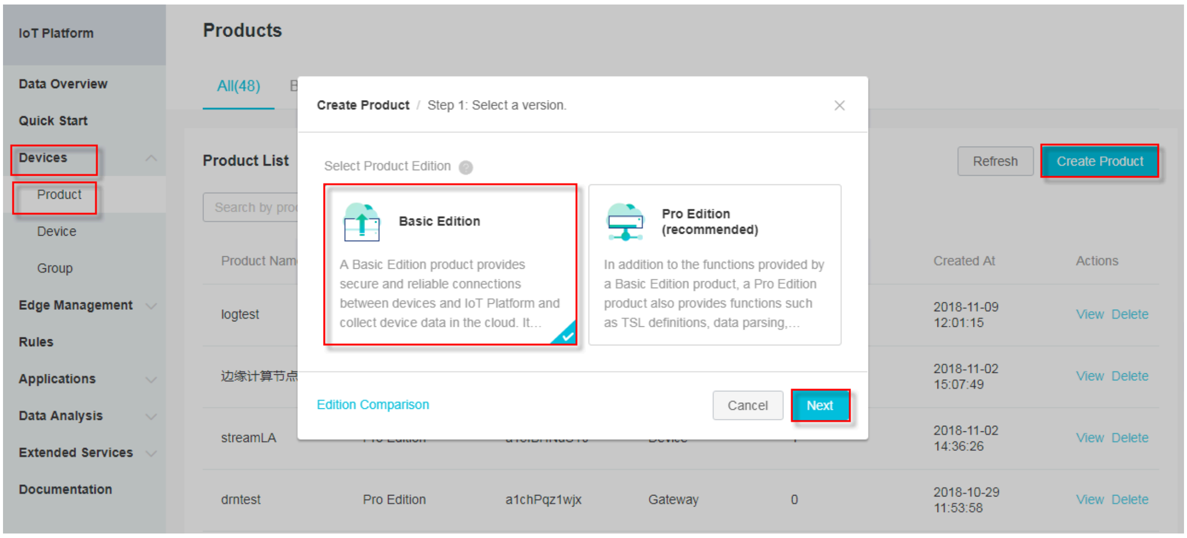
The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

Context

IoT Platform supports two editions of products: Basic Edition and Pro Edition. This article introduces how to create a Basic Edition product.

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. Select Basic Edition and click Next.



Create Product / Step 1: Select a version.

Select Product Edition

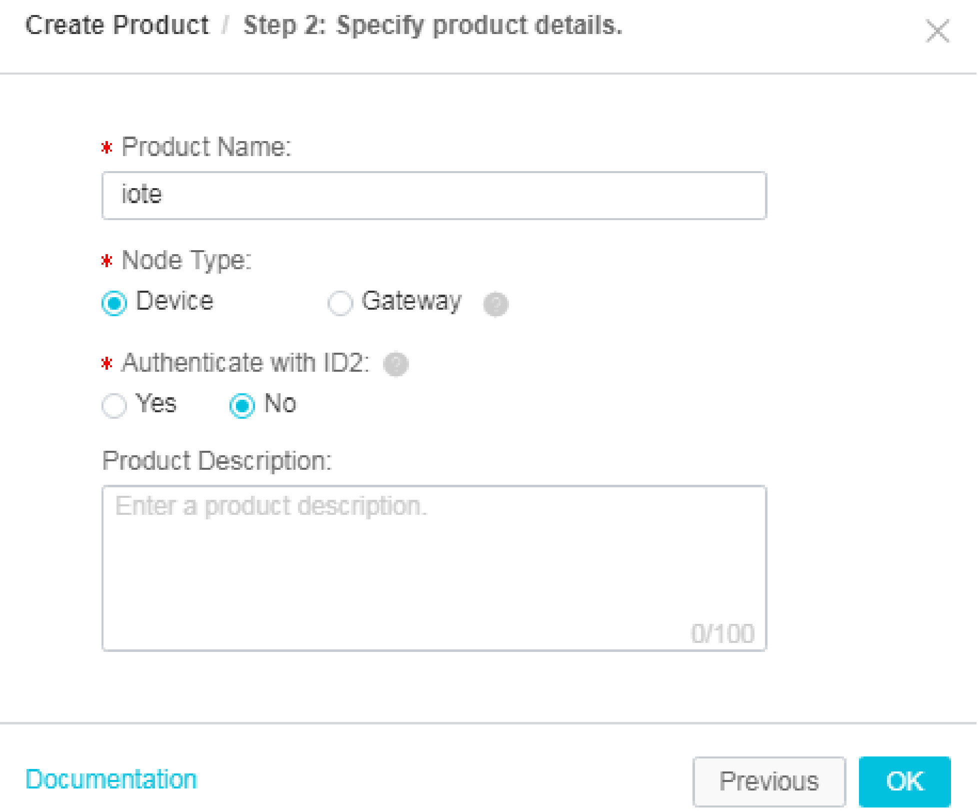
Basic Edition

A Basic Edition product provides secure and reliable connections between devices and IoT Platform and collect device data in the cloud. It...

Pro Edition (recommended)

In addition to the functions provided by a Basic Edition product, a Pro Edition product also provides functions such as TSL definitions, data parsing,...

4. Enter required information and click OK.



Create Product / Step 2: Specify product details.

* Product Name:

iote

* Node Type:

● Device ○ Gateway

* Authenticate with ID2:

○ Yes ● No

Product Description:

Enter a product description.

0/100

The parameters are described 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 English letters, digits and underscores.
Node Type	Options are Device and Gateway. · `Device` : Indicates that devices of this product cannot be mounted with sub-devices. This kind of devices can connect to IoT Platform directly or as sub-devices of gateway devices. · `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*.

Parameter	Description
Product Description	Describe the product information. You can enter up to 100 characters.

Result

After the product is created successfully, you are automatically redirected to the Products page. You can then view or edit the product information.

1.2 Create a product (Pro Edition)

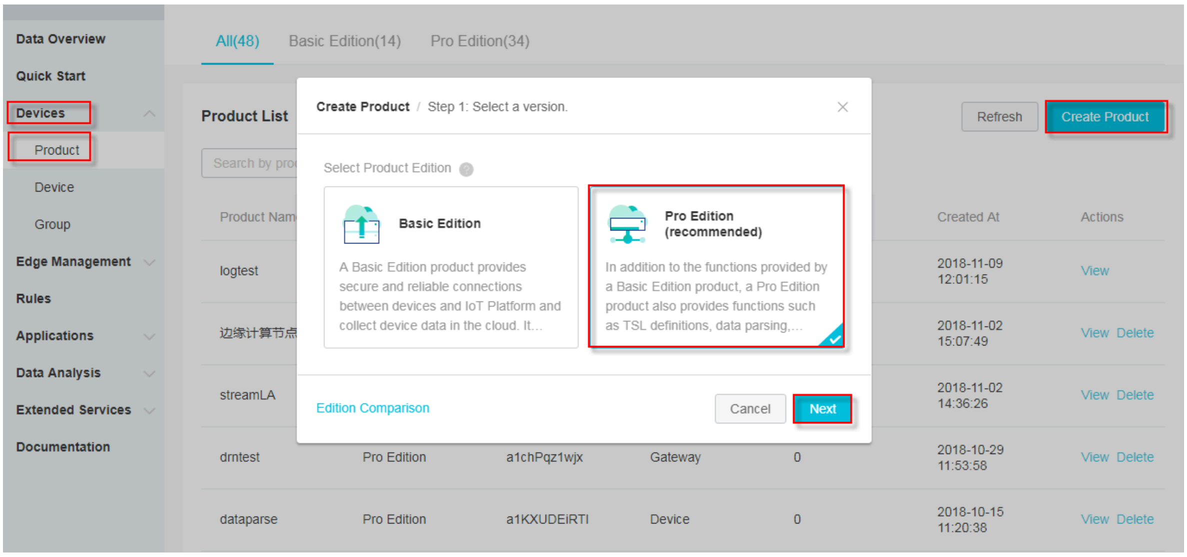
The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

Context

IoT Platform supports two editions of products: Basic Edition and Pro Edition. This topic describes how to create Pro Edition 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. Select Pro Edition and click Next.



Create Product / Step 1: Select a version.

Select Product Edition

Basic Edition

A Basic Edition product provides secure and reliable connections between devices and IoT Platform and collect device data in the cloud. It...

Pro Edition (recommended)

In addition to the functions provided by a Basic Edition product, a Pro Edition product also provides functions such as TSL definitions, data parsing,...

Edition Comparison

Product Nam					Created At	Actions
logtest					2018-11-09 12:01:15	View
边缘计算节点					2018-11-02 15:07:49	View Delete
streamLA					2018-11-02 14:36:26	View Delete
drntest	Pro Edition	a1chPqz1wjx	Gateway	0	2018-10-29 11:53:58	View Delete
dataparse	Pro Edition	a1KXUDEiRTI	Device	0	2018-10-15 11:20:38	View Delete

4. Enter all the required information and then click OK.

   The parameters are as follows.

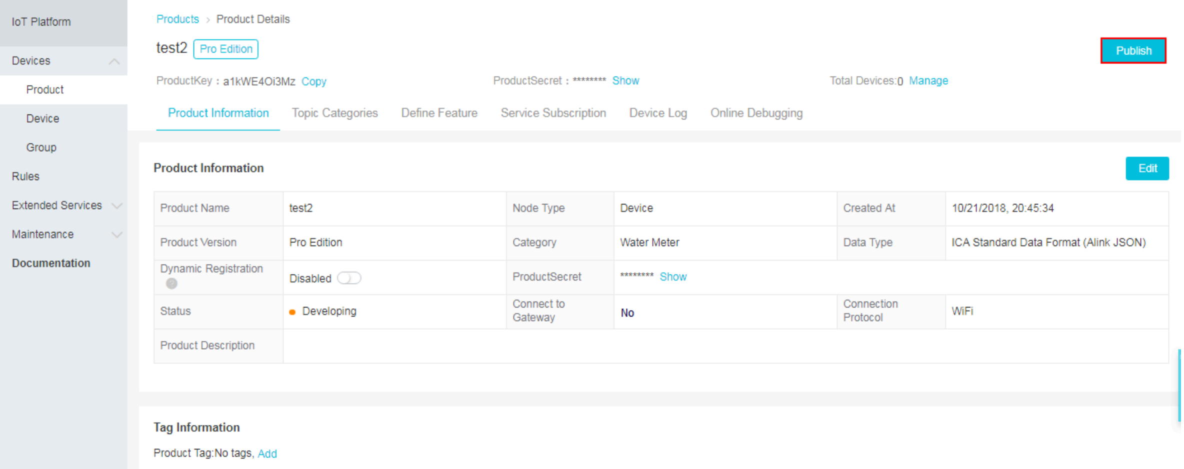
Parameter	Description
Product Name	The name of the product that you want to create. The product name must be unique within the account. For example, you can enter the product model as the product name. A product name is 4 to 30 characters in length, and can contain Chinese characters, English letters,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

Parameter	Description
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.
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 parsingso 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.



test2 Pro Edition

ProductKey : a1kWE4Oi3Mz Copy ProductSecret : ******** Show Total Devices:0 Manage

Product Information

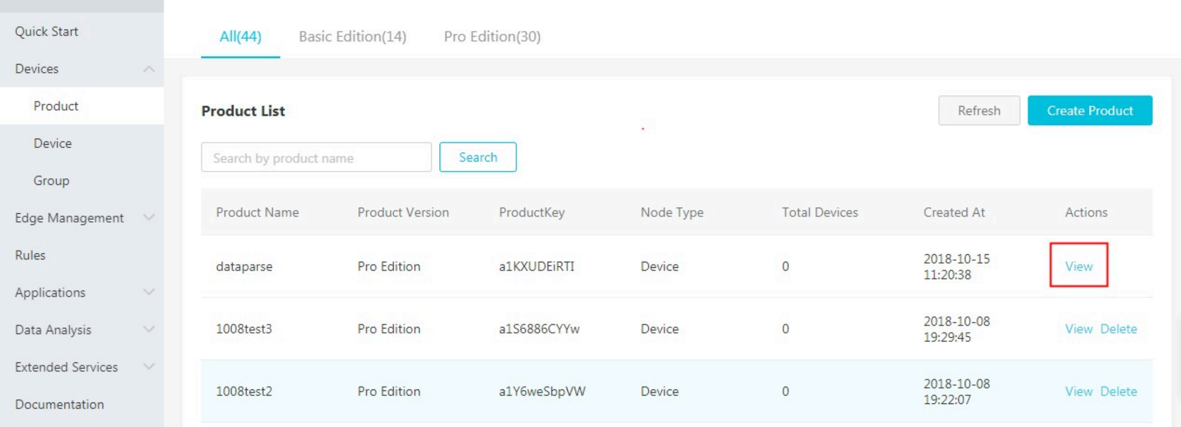
Product Name	test2	Node Type	Device	Created At	10/21/2018, 20:45:34
Product Version	Pro Edition	Category	Water Meter	Data Type	ICA Standard Data Format (Alink JSON)
Dynamic Registration	Disabled	ProductSecret	******** Show		
Status	Developing	Connect to Gateway	No	Connection Protocol	WiFi
Product Description					

Tag Information

Product Tag:No tags, Add

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.



Product List

Product Name	Product Version	ProductKey	Node Type	Total Devices	Created At	Actions
dataparse	Pro Edition	a1KXUDEiRTI	Device	0	2018-10-15 11:20:38	View
1008test3	Pro Edition	a1S6886CYYw	Device	0	2018-10-08 19:29:45	View Delete
1008test2	Pro Edition	a1Y6weSbpVW	Device	0	2018-10-08 19:22:07	View Delete

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

1.3 Create devices

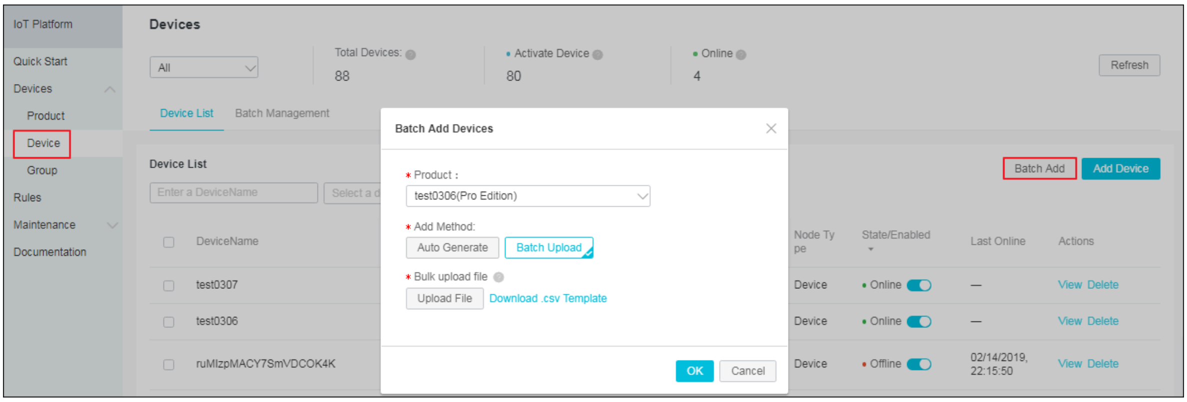
1.3.1 Create multiple devices at a time

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.



Total Devices: 88 | Activate Device 80 | Online 4

Batch Add Devices

* Product : test0306(Pro Edition)

* Add Method: Batch Upload

* Bulk upload file

DeviceName	Node Type	State/Enabled	Last Online	Actions
test0307	Device	Online	—	View Delete
test0306	Device	Online	—	View Delete
ruMIzpMACY7SmVDCOK4K	Device	Offline	02/14/2019, 22:15:50	View Delete

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.

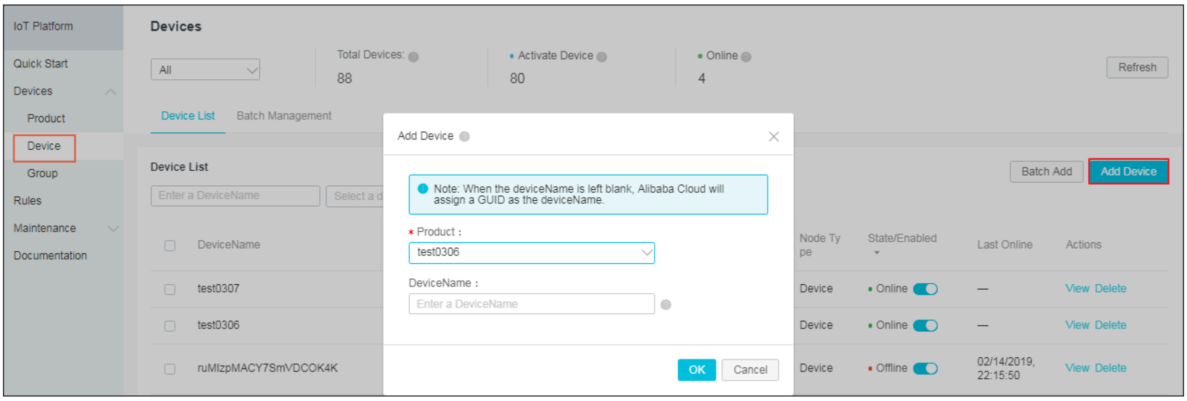
1.3.2 Create a device

A product is a collection of devices. After you have created products, you can create devices of the product models. You can create one device or multiple devices at a time. This article introduces how to create a single device.

Procedure

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click Devices > Device, and then click Add Device
3. Select a product that you have created. The device to be created will be assigned with the features of the selected product.
4. (Optional) Enter a name for the device. If you do not enter a device name for the device, the system will automatically generate one for the device.

> Note:
> A DeviceName (device name) must be unique within a product. It is used as a device identifier when the device communicates with IoT Platform.



Note: When the deviceName is left blank, Alibaba Cloud will assign a GUID as the deviceName.

* Product : test0306

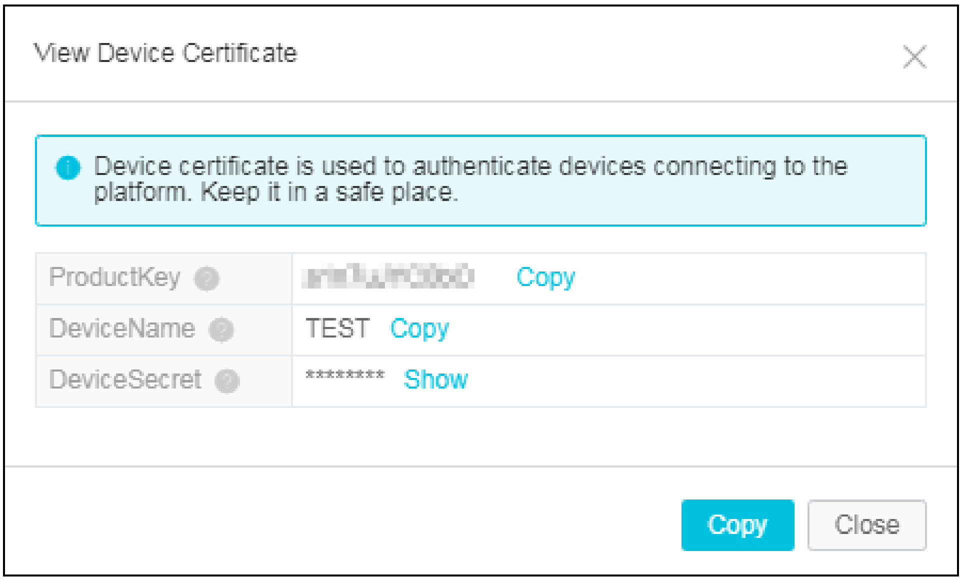
DeviceName : Enter a DeviceName

5. Click OK to create the device.

   After the device has been successfully created, the View Device Certificate box is displayed. There, 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.

- ProductKey: The globally unique identifier issued by IoT Platform for a product.
- DeviceName: The identifier of a device. It must be unique within a product and is used for device authentication and message communication.
- DeviceSecret: The secret key issued by IoT Platform for a device. It is used for authentication encryption and must be used in pairs with the DeviceName.



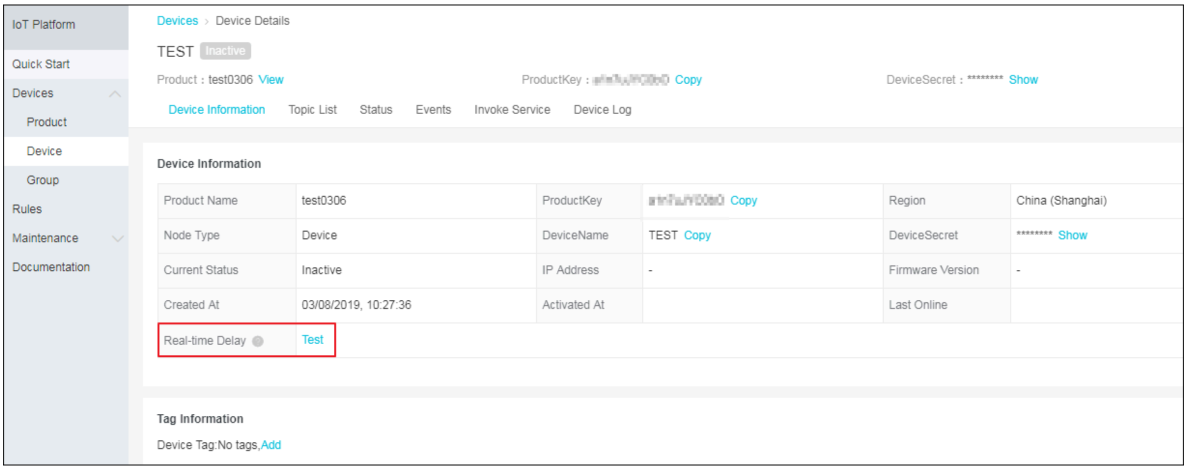
View Device Certificate

Device certificate is used to authenticate devices connecting to the platform. Keep it in a safe place.

ProductKey	[redacted] Copy
DeviceName	TEST Copy
DeviceSecret	******** Show

6. On the device list page, find the device and click View. On the Device Details page, you can view the information of this device.

   On the Device Details page, click Test to text the network latency for the device.



TEST Inactive

Product : test0306 View ProductKey : [redacted] Copy DeviceSecret : ******** Show

Device Information

Product Name	test0306	ProductKey	[redacted] Copy	Region	China (Shanghai)
Node Type	Device	DeviceName	TEST Copy	DeviceSecret	******** Show
Current Status	Inactive	IP Address	-	Firmware Version	-
Created At	03/08/2019, 10:27:36	Activated At		Last Online	
Real-time Delay	Test				

Tag Information

Device Tag:No tags,Add

1.4 TSL

1.4.1 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 or by using APIs.

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.
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.

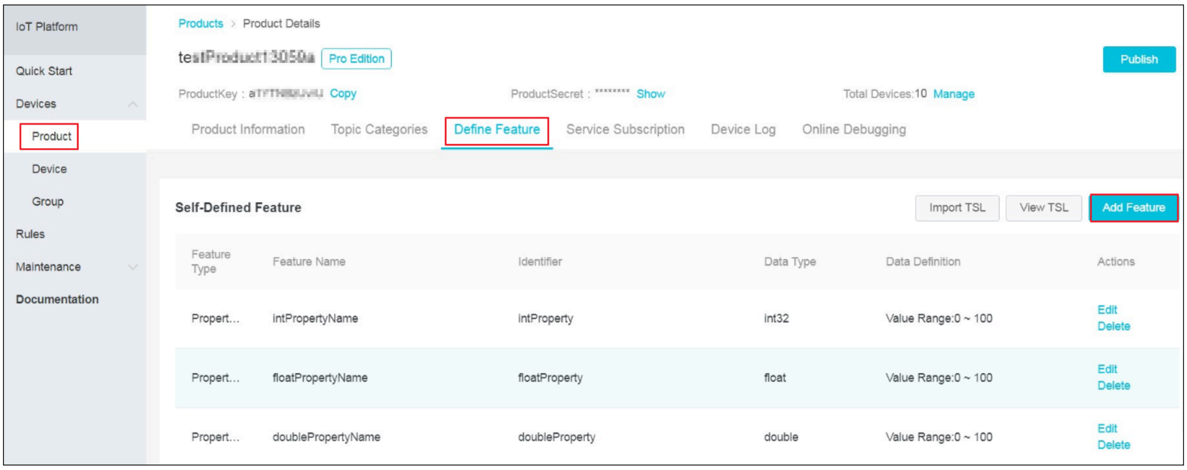
1.4.2 Define features

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

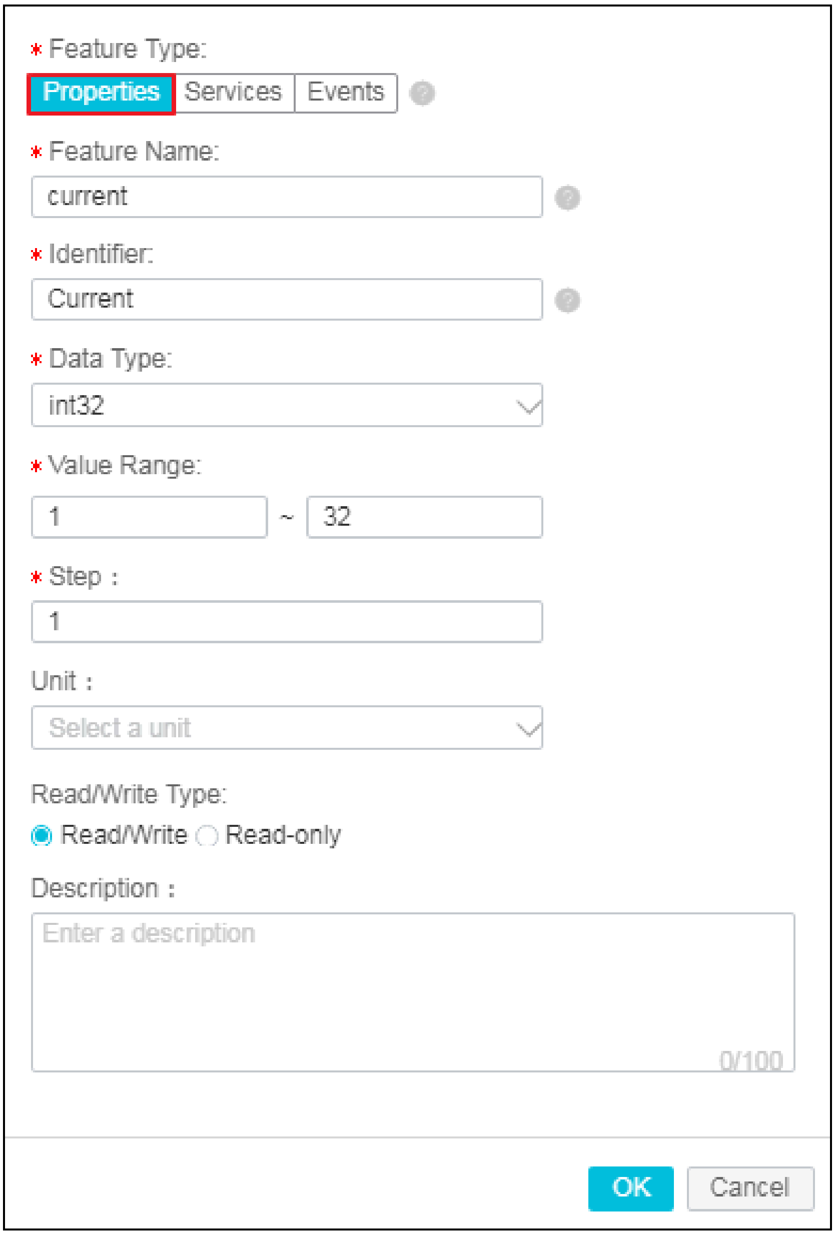
1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click Devices > Product.
3. On the Products page, find the product for which you want to define features and click View.
4. Click Define Feature.

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.



Feature Type	Feature Name	Identifier	Data Type	Data Definition	Actions
Propert...	intPropertyName	intProperty	int32	Value Range:0 ~ 100	Edit Delete
Propert...	floatPropertyName	floatProperty	float	Value Range:0 ~ 100	Edit Delete
Propert...	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 clickOK.



* Feature Type:

Properties | Services | Events

* Feature Name: current

* Identifier: Current

* Data Type: int32

* Value Range: 1 ~ 32

* Step : 1

Unit : Select a unit

Read/Write Type: ● Read/Write ○ Read-only

Description : Enter a description 0/100

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 `identifier` 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, PowerConsumption. **Note:** An identifier cannot be any one of the following words: set, get, post, time, and value, because they are system parameter names.

Parameter	Description
Data Type	- `int32` : 32-bit integer. If you select int32, you are required to define the value range, step, and unit. - `float` : Float. If you select float, you are required to define the value range, step, and unit. - `double` : Double float. If you select double, you are required to define the value range, step, and unit. - `enum` : Enumeration. You must specify enumeration items with values and descriptions. For example,1 indicates heating mode and 2 indicates cooling mode. - `bool` : 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. - `text` : Text string. You must specify the data length. The maximum value is 2048 bytes. - `date` : Timestamp. A UTC timestamp in string type, in milliseconds. - `struct` : 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. - `array` : Array. You must select a data type for the elements in the array from int32, float, double, text and struct. 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 int32, float, or double, step is required.
Unit	You can select None or a unit suitable.

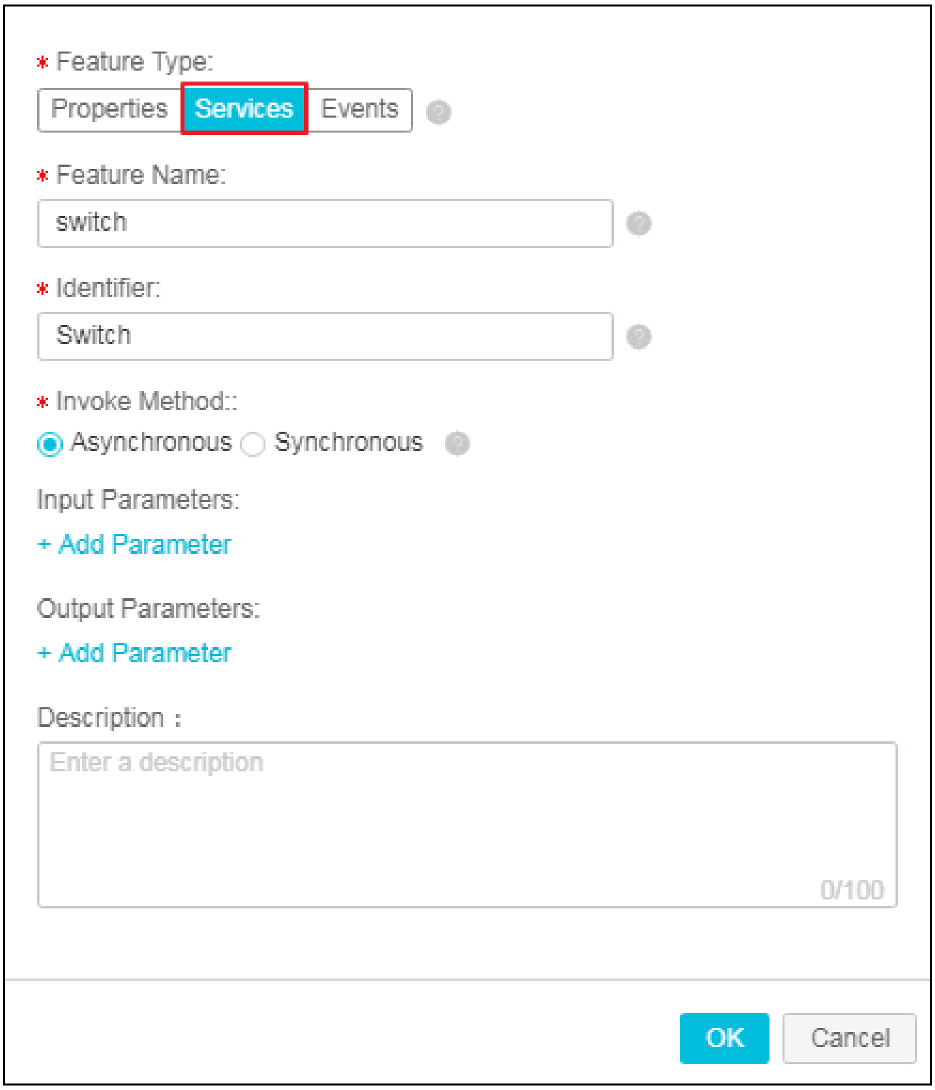
Parameter	Description
Read/Write Type	- `Read / Write` : GET and SET methods are supported for Read/Write requests. - `Read - only` : Only GET is supported for Read-only requests. Note: When the gateway connection protocol is Modbus, you do not set this parameter.
Description	Enter a description or remarks 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 `At Specific Time` or `Report Changes` . - 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:

Output Parameters:

Description :

Enter a description

0/100

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 `identifier` 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.

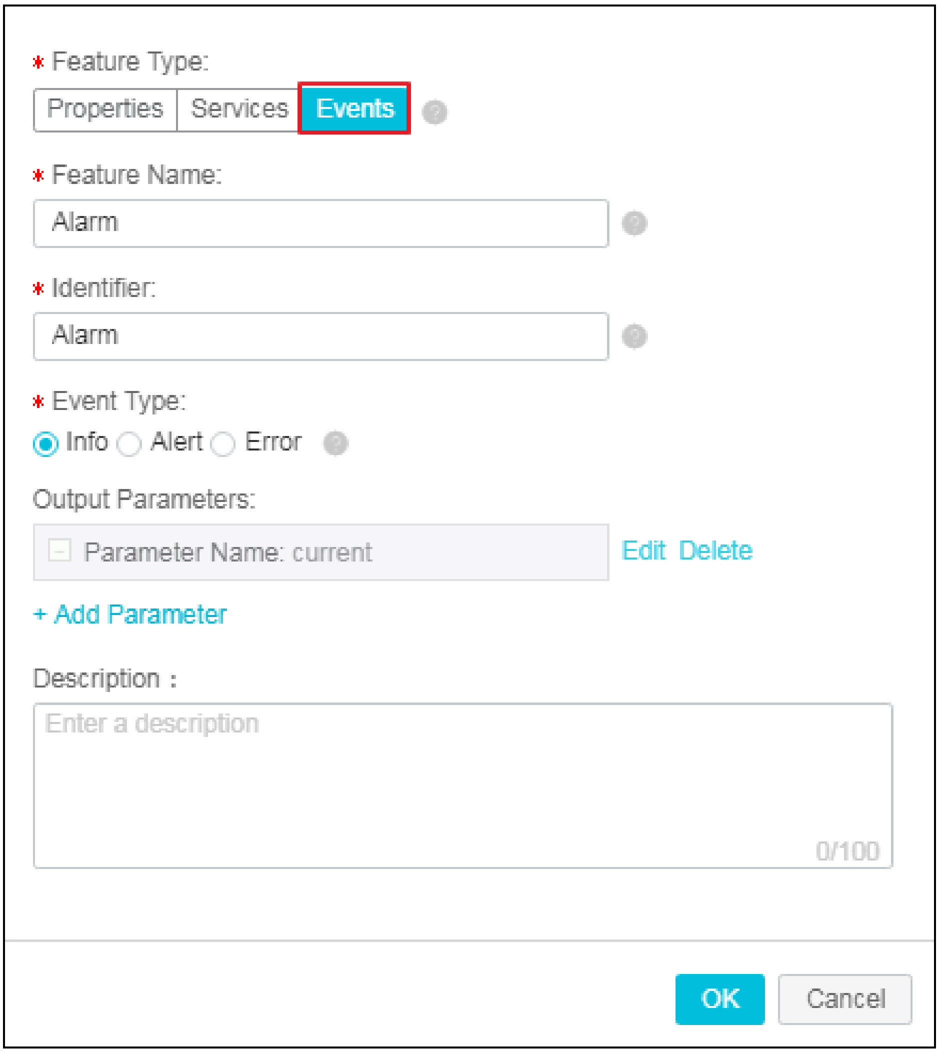
Parameter	Description
Input Parameters	(Optional) Set input parameters for the service. Click Add Parameter, and add an input parameter in the dialog box that appears. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. Note: - 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 `SoilHumidi ty` 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.
Extended Information	When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the service.

Parameter	Description
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

Description : Enter a description 0/100

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 `identifier` in Alink JSON TSL, and is used as the key when a device is reporting data of this event, for example, ErrorCode. Note: Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value.
Event Type	- `Info` : Indicates general notifications reported by devices, such as the completion of a specific task. - `Alert` : 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. - `Error` : 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 `Voltage` 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: set, get, post, time, and value. - You can add up to 20 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.

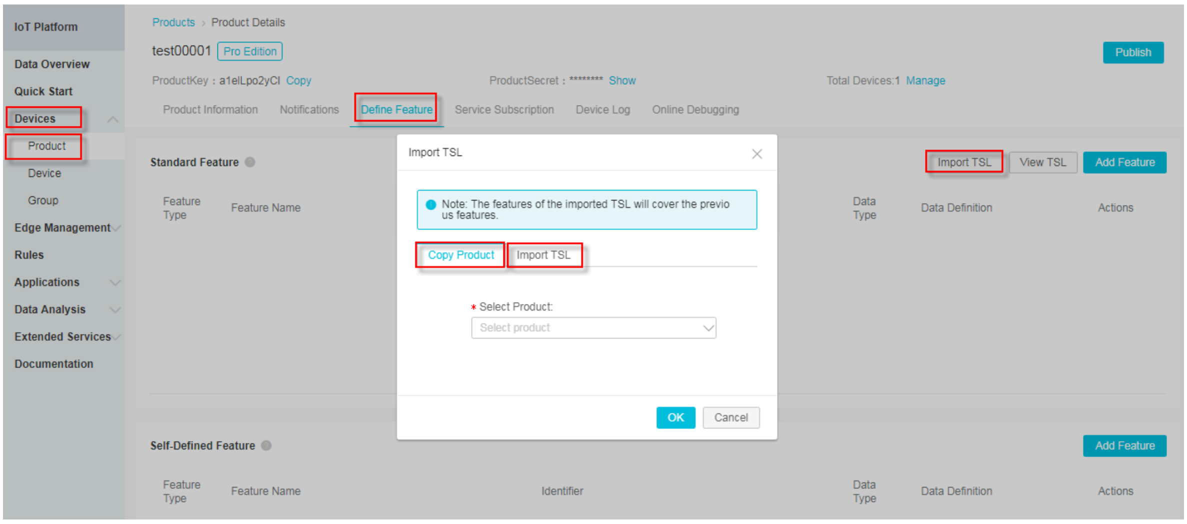
1.4.3 Import Thing Specification Language (TSL)

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.

1.4.4 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 Pro Edition 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 ",
" operateTyp  e ": "( 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 ": {
```

```
" registerCo  unt ": " 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 ",
" reverseReg  ister ": " 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 ",
" pollingTim  e ": " 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 "
    }
  ]
}
```

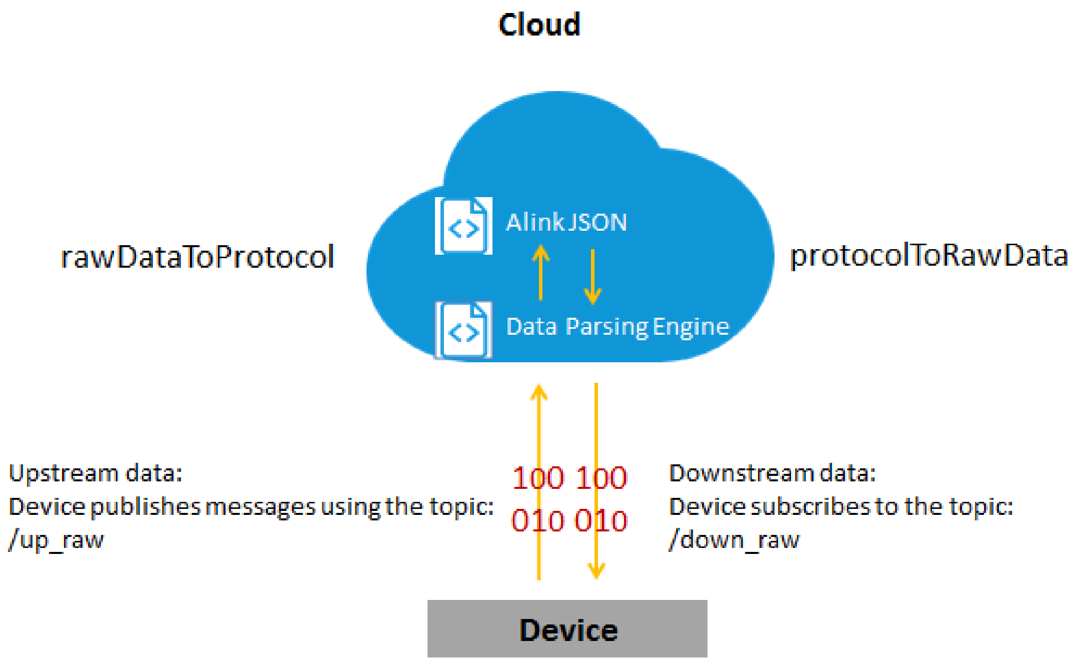
1.5 Data parsing

When you create a product on the IoT Platform console, if you select Do not parse/ Custom as the data type, you can write a script in the IoT Platform console to parse the original data into Alink JSON format.

What is data parsing?

Data parsing is a method that allows devices with limited storage space or bandwidth to avoid directly sending data to IoT Platform in Alink JSON format. Instead, devices pass original data to the cloud, whereby a script is run to convert the data into Alink JSON format. To allow devices to pass original data to the cloud, select `Do not parse / Custom` as the data type when creating the product, and then write a JavaScript file to parse the data. IoT Platform provides an online editor for you to edit and debug your data parsing script.

Data parsing process:

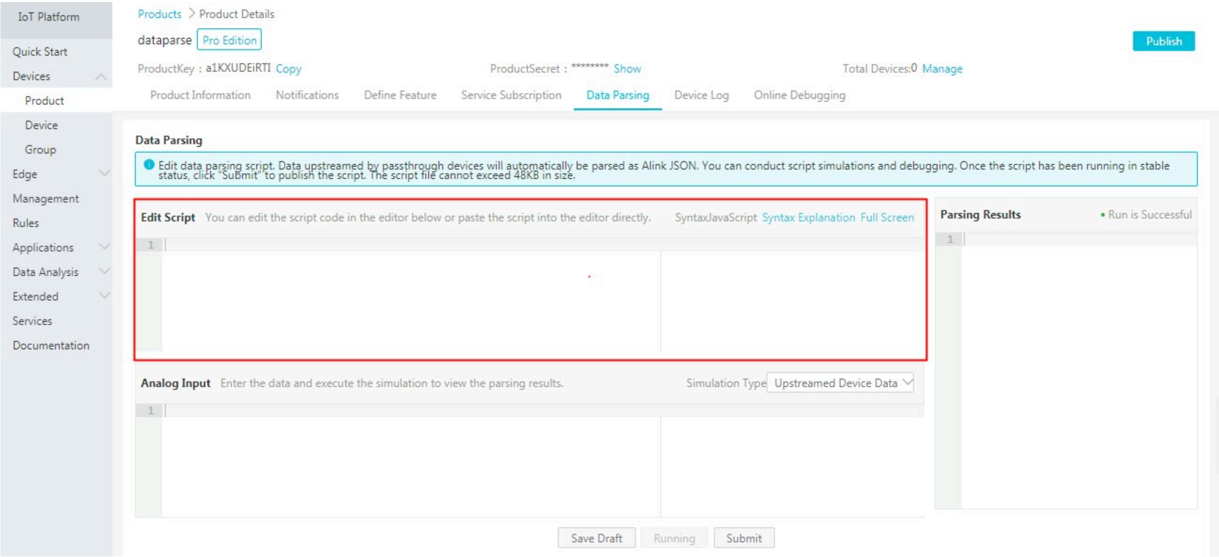


Using the data parsing script editor, you can:

- Edit your JavaScript data parsing file online.
- Save content as a draft, edit the draft, or delete it.
- Debug your script using analog data. You can enter upstream or downstream analog data, and run the script to check whether it works.
- Perform static syntax check (JavaScript syntax).
- Submit a verified script to the running platform for device data parsing.

Edit a script online

On the product details page, click Data Parsing and then enter your data parsing script in the edit box. Currently, only JavaScript is supported.

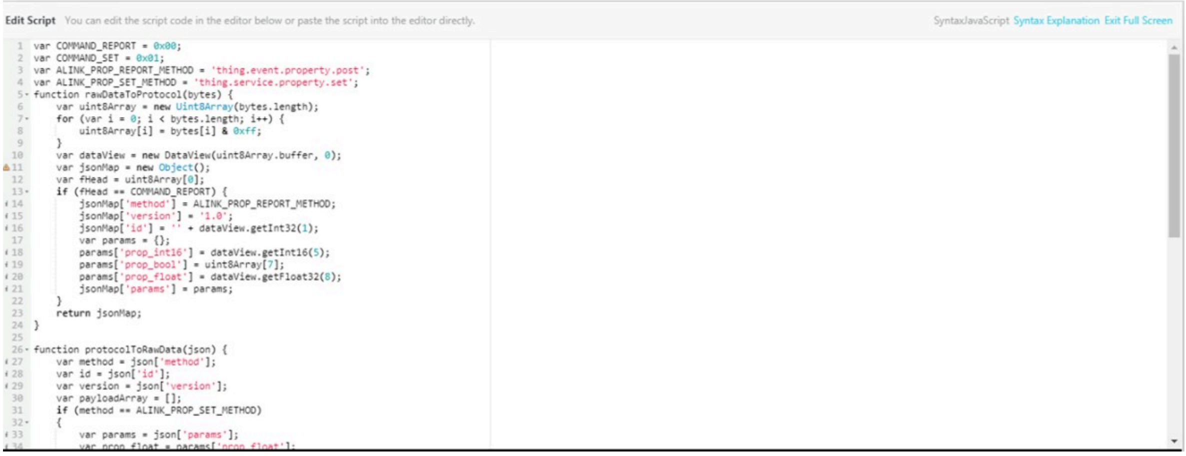


Edit data parsing script. Data upstreamed by passthrough devices will automatically be parsed as Alink JSON. You can conduct script simulations and debugging. Once the script has been running in stable status, click "Submit" to publish the script. The script file cannot exceed 48KB in size.

You can edit the script code in the editor below or paste the script into the editor directly.

Enter the data and execute the simulation to view the parsing results.

- Click Full Screen to view or edit a script in full screen. Click Exit Full Screen to exit the full screen mode.



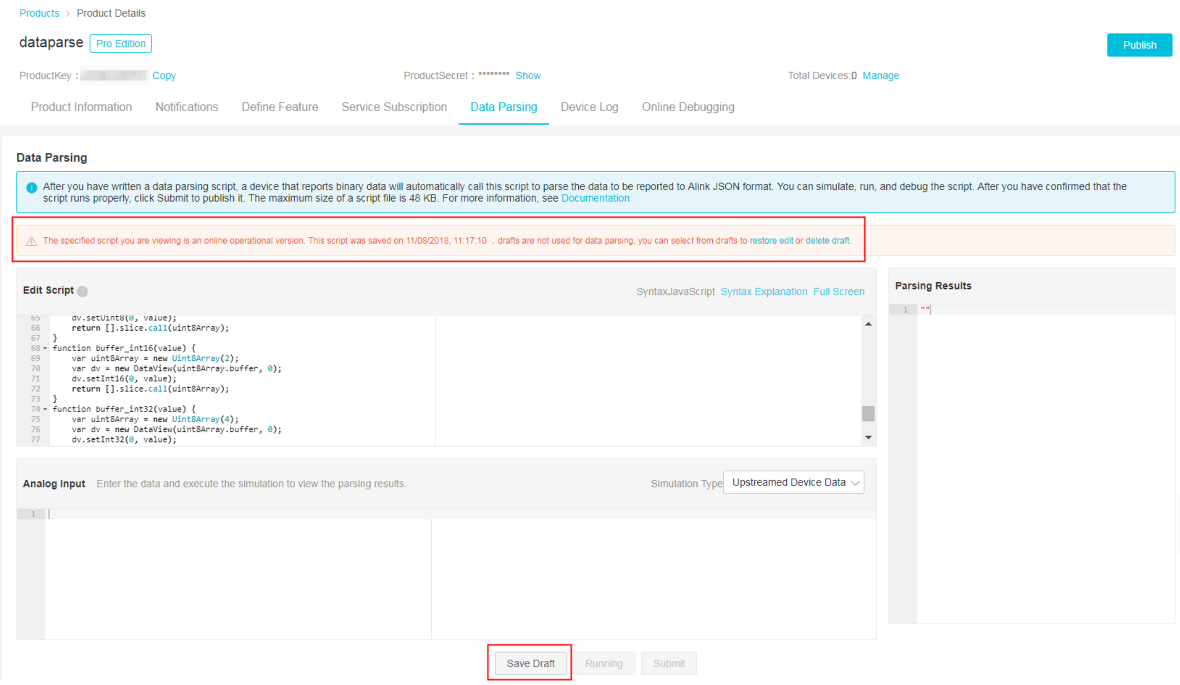
You can edit the script code in the editor below or paste the script into the editor directly.

```
var COMMAND_REPORT = 0x00;
var COMMAND_SET = 0x01;
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post';
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';
function rawDataToProtocol(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];
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD;
        jsonMap['version'] = '1.0';
        jsonMap['id'] = '' + dataView.getInt32(1);
        var params = {};
        params['prop_int16'] = dataView.getInt16(5);
        params['prop_bool'] = uint8Array[7];
        params['prop_float'] = dataView.getFloat32(8);
        jsonMap['params'] = params;
    }
    return jsonMap;
}

function protocolToRawData(json) {
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD)
    {
        var params = json['params'];
        var prop_float = params['prop_float'];
```

- Click Save Draft at the bottom of the page to save the content you have edited. When you access the data parsing page next time, the system will prompt a notification saying that you have a draft. You can then choose to Restore Edit or Delete Draft.

- When saved, a draft script is not published to the running parsing platform, and does not affect a currently published script.
- A new draft will overwrite any previously saved draft.


Products > Product Details

dataparse Pro Edition

ProductKey : [redacted] Copy ProductSecret : ******** Show Total Devices:0 Manage

Data Parsing

After you have written a data parsing script, a device that reports binary data will automatically call this script to parse the data to be reported to Alink JSON format. You can simulate, run, and debug the script. After you have confirmed that the script runs properly, click Submit to publish it. The maximum size of a script file is 48 KB. For more information, see Documentation

The specified script you are viewing is an online operational version. This script was saved on 11/08/2018, 11:17:10 , drafts are not used for data parsing, you can select from drafts to restore edit or delete draft.

Edit Script

SyntaxJavaScript

```
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value) {
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
```

Analog Input Enter the data and execute the simulation to view the parsing results.

Simulation Type: Upstreamed Device Data

Parsing Results

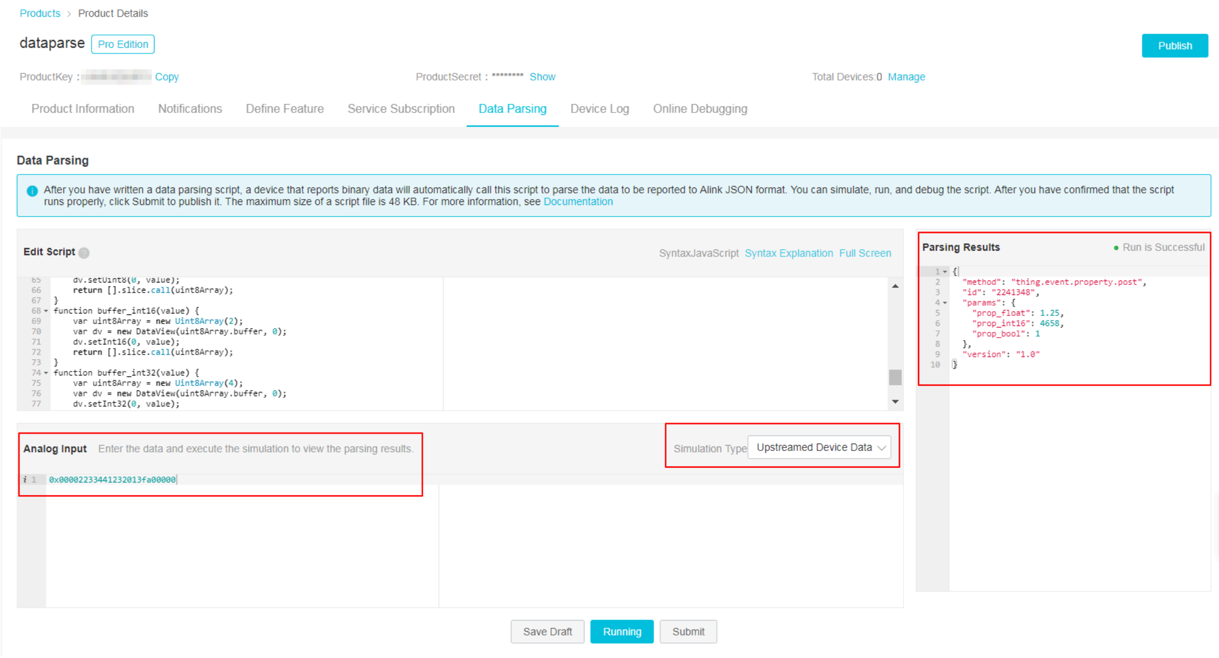
```
""
```

Verify the script using analog data

After the script is edited, you can enter analog data in the Analog Input box and click Running. The system will call this script to parse the analog data and the parsed result will be displayed in the Parsing Results box at the right side of the page. If the script is not correct, the message Failed to Run will be displayed next to Parsing Results, and an error message will be display in the box with information that you can use to to correct the script.

Parse upstream analog data

Select `Upstreamed Device Data` as the simulation type, enter the device's binary data which is to be passed through, and click Running. The system will convert the binary data to Alink JSON format, and the results are displayed in a box at the right side of the page.



Data Parsing

After you have written a data parsing script, a device that reports binary data will automatically call this script to parse the data to be reported to Alink JSON format. You can simulate, run, and debug the script. After you have confirmed that the script runs properly, click Submit to publish it. The maximum size of a script file is 48 KB. For more information, see Documentation

Edit Script

```
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value) {
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
```

Analog Input Enter the data and execute the simulation to view the parsing results.

Simulation Type: Upstreamed Device Data

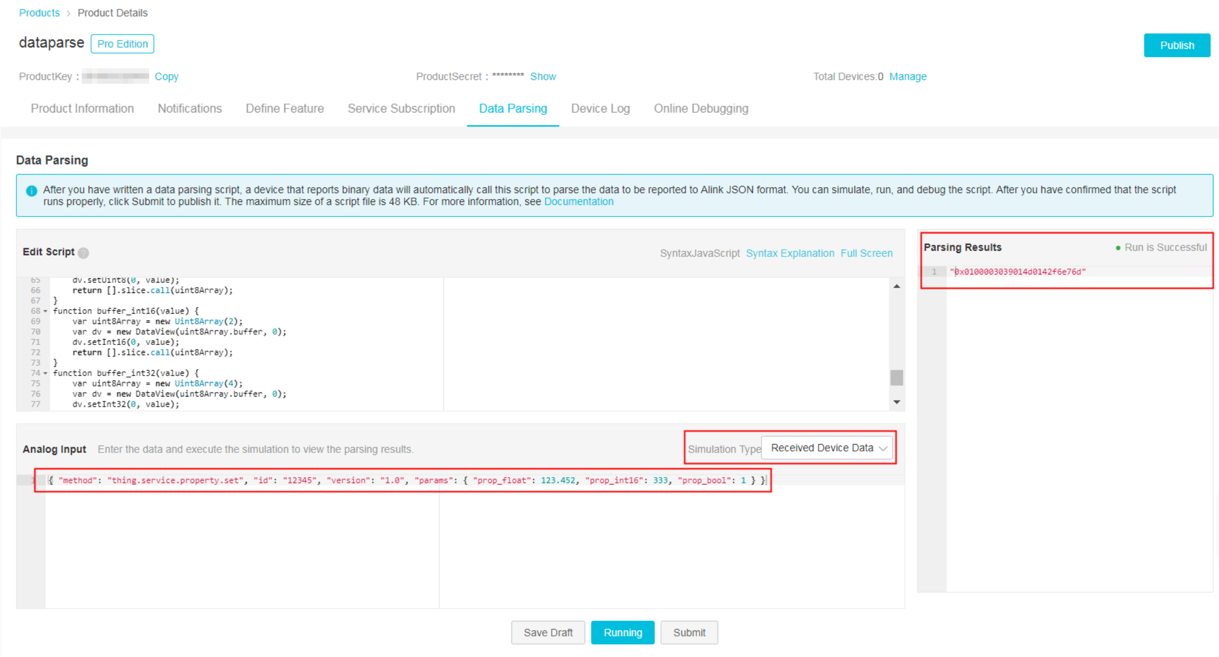
```
0x00002233441232013fa00000
```

Parsing Results ● Run is Successful

```
{
  "method": "thing.event.property.post",
  "id": "2241348",
  "params": {
    "prop_float": 1.25,
    "prop_int16": 4658,
    "prop_bool": 1
  },
  "version": "1.0"
}
```

Parse downstream analog data

Select `Receive Device Data` , enter Alink JSON formatted data, and click Running. The system will convert the ALink JSON data to binary data, and the results are displayed in a box at the right side of the page.



Data Parsing

After you have written a data parsing script, a device that reports binary data will automatically call this script to parse the data to be reported to Alink JSON format. You can simulate, run, and debug the script. After you have confirmed that the script runs properly, click Submit to publish it. The maximum size of a script file is 48 KB. For more information, see Documentation

Edit Script

```
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value) {
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
```

Analog Input Enter the data and execute the simulation to view the parsing results.

Simulation Type: Received Device Data

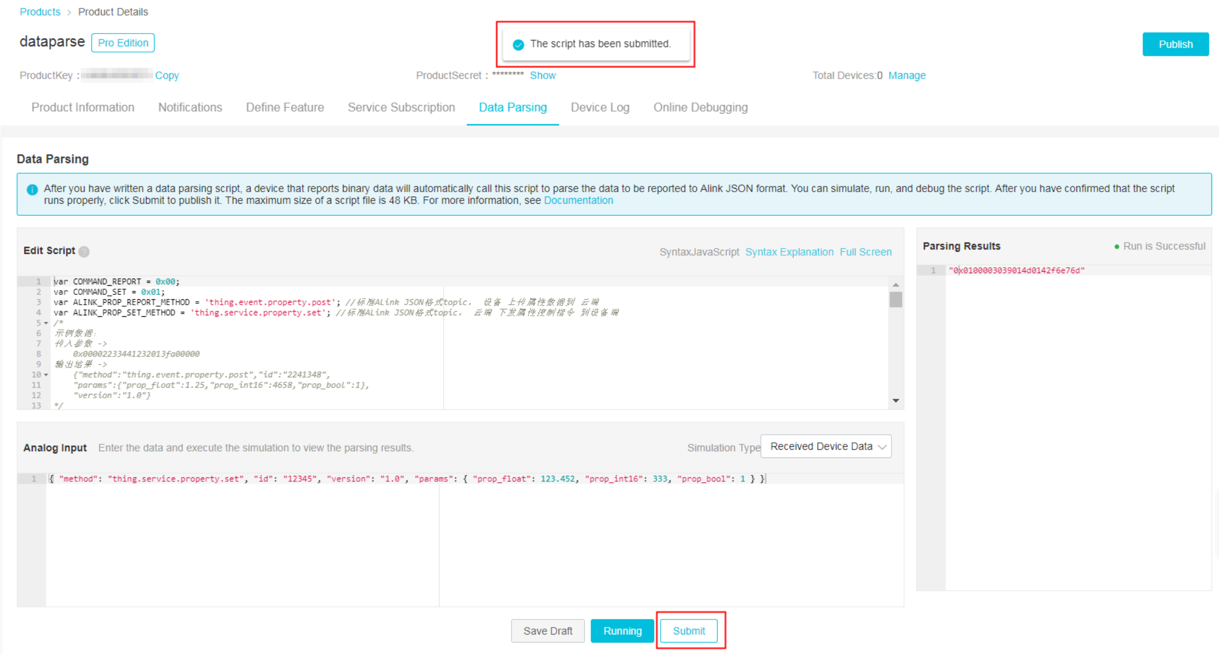
```
{ "method": "thing.service.property.set", "id": "12345", "version": "1.0", "params": { "prop_float": 123.452, "prop_int16": 333, "prop_bool": 1 } }
```

Parsing Results ● Run is Successful

```
"0x0100003039014d0142f6e76d"
```

Submit the script

In order to guarantee that submitted scripts are correct and run properly, only scripts that have passed parsing test can be submitted to the running platform. After a script is submitted, the system will automatically use it to convert the upstream data and downstream data of devices.



The script has been submitted.

ProductKey : [redacted] Copy ProductSecret : ******** Show Total Devices:0 Manage

Data Parsing

After you have written a data parsing script, a device that reports binary data will automatically call this script to parse the data to be reported to Alink JSON format. You can simulate, run, and debug the script. After you have confirmed that the script runs properly, click Submit to publish it. The maximum size of a script file is 48 KB. For more information, see Documentation

```
var COMMAND_REPORT = 0x00;
var COMMAND_SET = 0x01;
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //标准ALink JSON格式topic， 设备 上传属性数据到 云端
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'; //标准ALink JSON格式topic， 云端 下发属性控制指令 到设备端
/*
示例数据:
传入参数 ->
    0x00002233441232013fa00000
输出结果 ->
    {"method":"thing.event.property.post","id":"2241348",
    "params":{"prop_float":1.25,"prop_int16":4658,"prop_bool":1},
    "version":"1.0"}
*/
```

Enter the data and execute the simulation to view the parsing results.

Simulation Type: Received Device Data

```
{ "method": "thing.service.property.set", "id": "12345", "version": "1.0", "params": { "prop_float": 123.452, "prop_int16": 333, "prop_bool": 1 } }
```

```
"0x0100003039014d0142f6e76d"
```

Note:

A script must successfully parse analog data at least once before you can submit it.

Development framework

Overview

The following two methods must be defined in a script:

- `protocolToRawData` : Convert Alink JSON format data to binary data.
- `rawDataToProtocol` : Convert binary data to Alink JSON format data.

Language

Currently, only JavaScript that meets ECMAScript 5.1 is supported.

Define the methods

- Convert Alink JSON formatted data to binary data:

```
// Parses Alink JSON format data sent by the server
 and converts it to binary data
function protocolToRawData(jsonObj){
    return rawdata;
}
```

Parameter description: Input parameters (jsonObj) match with the Alink JSON format data in the TSL of the product.

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

Returned parameter: A binary byte array. For example:

```
 0x01000030  39014d0142  f6e76d
```

- Convert binary data to Alink JSON format data:

```
//  Parses   binary   data   sent   by   a   device   and   converts
   it   to   Alink   JSON   format   data
 function   rawDataToP  rotocol ( rawData ){
     return   jsonObj ;
}
```

Parameter description: Input parameter (rawData) is a binary byte array, for example,

```
 0x00002233  441232013f  a00000
```

Returned parameters: Data matches with the Alink JSON format data in the TSL of the product.

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

Script demo

1. Create a product and define features for the product.

   a. Create a Pro Edition product and select `Do not parse / Custom` as the data type.

   b. Define features (such as properties, services, and events) for the product. In this demo, the following three properties are defined:

Identifier	Data type
prop_float	float
prop_int16	int32
prop_bool	bool

2. Serial port protocol example.

Frame type	ID	prop_int16	prop_bool	prop_float
One byte. 0 - upstream; 1 - downstream.	Request sequence number.	Two bytes. Property value of prop_int16.	One byte. Property value of prop_bool.	Four bytes. Property value of prop_float.

3. Copy the script demo codes.

   Copy and paste the following demo codes into the script edit box:

```
 var   COMMAND_RE  PORT  =  0x00 ;
 var   COMMAND_SE  T  =  0x01 ;
 var   ALINK_PROP  _REPORT_ME  THOD  = ' thing . event . property .
 post '; // A   standard   ALink   JSON   formatted   topic   for
 devices   to   upload   property   data   to   the   cloud .
 var   ALINK_PROP  _SET_METHO  D  = ' thing . service . property .
 set '; // A   standard   ALink   JSON   formatted   topic   for
 the   cloud   to   send   property   management   commands   to
 devices .
/*
 Sample   data :
 Input   parameters  ->
     0x00002233  441232013f  a00000
 Output   parameters   ->
    {" method ":" thing . event . property . post "," id ":" 2241348
 ",
    " params ":{" prop_float ": 1 . 25 ," prop_int16 ": 4658 ,"
 prop_bool ": 1 },
    " 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 ; //
 ALink   JSON   formatted   topic   for   reporting   properties
         jsonMap [' version '] = ' 1 . 0 '; // Protocol   version
 in   ALink   JSON   format
         jsonMap [' id '] = '' +  dataView . getInt32 ( 1 ); // The
   request   ID   value   in   ALink   JSON   format
         var   params  = {};
         params [' prop_int16 '] =  dataView . getInt16 ( 5 ); //
 The   property   of   prop_int16   of   the   product
         params [' prop_bool '] =  uint8Array [ 7 ]; // The
 property   of   prop_bool
         params [' prop_float '] =  dataView . getFloat32 ( 8 ); //
 The   property   of   prop_float .
         jsonMap [' params '] =  params ; // Standard   fields   of
   params   in   ALink   JSON   format
    }
     return   jsonMap ;
}
/*
 Sample   data :
 Input   parameters   ->
    {" method ":" thing . service . property . set "," id ":" 12345
 "," version ":" 1 . 0 "," params ":{" prop_float ": 123 . 452 , "
 prop_int16 ": 333 , " prop_bool ": 1 }}
 Output   parameters  ->
     0x01000030  39014d0142  f6e76d
*/
 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 ) //  Property
 settings
    {
         var   params  =  json [' params '];
         var   prop_float  =  params [' prop_float '];
         var   prop_int16  =  params [' prop_int16 '];
         var   prop_bool  =  params [' prop_bool '];
        //  Raw   data   connected   according   to   the   custom
 protocol   format
         payloadArr  ay  =  payloadArr  ay . concat ( buffer_uin  t8
 ( COMMAND_SE  T )); //  command   field
         payloadArr  ay  =  payloadArr  ay . concat ( buffer_int  32
 ( parseInt ( id ))); //  ID   in   ALink   JSON   format
         payloadArr  ay  =  payloadArr  ay . concat ( buffer_int  16
 ( prop_int16 )); //  The   value   of   property  ' prop_int16 '
         payloadArr  ay  =  payloadArr  ay . concat ( buffer_uin  t8
 ( prop_bool )); //  The   value   of   property  ' prop_bool '
         payloadArr  ay  =  payloadArr  ay . concat ( buffer_flo
 at32 ( prop_float )); //  The   value   of   property  ' prop_float
 '
    }
     return   payloadArr  ay ;
}
//  The   followings   are   the   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 );
}
```

4. **Verify the script using analog data**

- **Parse analog upstream data**

  Select `Upstreamed   Device   Data` **and enter the following data:**

  ```
  0x00002233  441232013f  a00000
  ```

  click Running, and then view the outputs:

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

- **Select** `Received   Device   Data` **, and enter the following data:**

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

```
}
```

click Running, and then view the output:

```
 0x01000030  39014d0142  f6e76d
```

Appendix: Method for debugging scripts written in a local computer

Currently, IoT Platform Data Parsing does not support debugging on the running platform. Therefore, we recommend that you directly paste the finished script into the online editor and then test it. The following is example output of the test method.

```
//  Test   Demo
 function   Test ()
{
    // 0x00123201  3fa00000
     var   rawdata_re  port_prop  =  new   Buffer ([
         0x00 , // Fixed   command   header ,  0   indicates
 reporting   property
         0x00 ,  0x22 ,  0x33 ,  0x44 , //  Identify   the   request
 sequence   correspond  ing   to   the   ID   fields .
         0x12 ,  0x32 , // Two - byte   value   in   int16 ,
 correspond  ing   to   the   property   of   prop_int16
         0x01 , // One - byte   value   in   bool ,  correspond  ing
 to   the   property   of   prop_bool
         0x3f ,  0xa0 ,  0x00 ,  0x00  // Four - byte   value   in
 float ,  correspond  ing   to   the   property   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 ();
```

1.6 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.

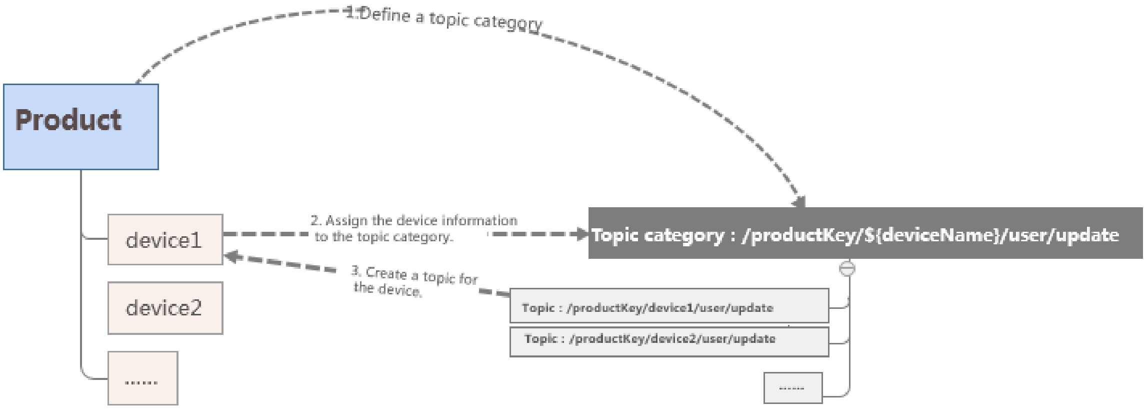
1.6.1 What is a topic?

Servers and devices communicate with each other in IoT Platform through topics. Topics are associated with devices, and topic categories are associated with products.

What is a topic category?

To simplify authorization operations and facilitate communication between devices and IoT Platform, topic categories were introduced. When you create a product, IoT Platform will create a default topic category for the product. In addition, when you create a device, the topic category will be automatically assigned to the device. You do not need to authorize each individual device to publish or subscribe to a topic.

Figure 1-1: The process of automatically creating a topic


1.Define a topic category
Product
device1
device2
......
2. Assign the device information to the topic category.
3. Create a topic for the device.
Topic category : /productKey/${deviceName}/user/update
Topic : /productKey/device1/user/update
Topic : /productKey/device2/user/update
......

When you create a product, IoT Platform automatically creates standard topic categories for the product. You can view all topic categories of the product on the Topic Categories tab page.

Description of topic categories:

- A topic category is a set of topics within the same product. For example, the topic category `/${ productKey }/${ deviceName }/ update` is a collection of the specific topics: `/${ productKey }/ device1 / update` and `/${ productKey }/ device2 / update`.

- The topic category must use a forward slash (/) to delimit the topic hierarchy. Two of the category levels are reserved: `${ productKey }` represents the product identifier, and `${ deviceName }` represents the device name.
- Each category level can only contain letters, numbers, and underscores (_). Topic category levels cannot be left empty.
- Operations available for devices: Publish indicate that the device can publish messages to a topic. Subscribe indicates that the device can subscribe to messages of a topic.
- IoT Platform Basic supports customized topic categories. Customizing topic categories allows for flexible communication to suit your business needs. Customizing topic categories and modifying category level names is not supported in IoT Platform Pro.
- The system-defined topic categories are pre-defined by IoT Platform Pro, do not support customization, and do not begin with `/${ productKey }`. For example, in IoT Platform Pro, topic categories provided for the Thing Special Language (TSL) begin with `/ sys /`, topic categories provided for firmware upgrades begin with `/ ota /`, and topic categories provided for device shadows begin with `/ shadow /`.

What is a topic?

A topic category is used for topic definition rather than communication. A topic is used for communication.

- Topics and topic categories use the same format. The difference is that in a topic category, the `${ deviceName }` is a variable, but in a topic it represents a specific device name.
- A topic is automatically derived from the device name and the topic category of the product. A topic contains a device name (`deviceName`), which can only be used in Pub/Sub communication. For example, the topic `/${ productKey }/ device1 / update` is owned by the device with name `device1` . Therefore, you can only publish or subscribe to messages to this topic for the device with name `device1` , and cannot use it for device with name `device2` to publish or subscribe to messages.

- When you configure the rules engine, the topic that you configure can contain one wildcard character.

Table 1-1: Wildcard characters in a topic

Wildcard character	Description
#	Must be the last character in the topic, and works as a wildcard by matching all topics in the current tree and all sub-trees of the topic hierarchy. For example, the topic `/ productKey / device1 /#` can represent `/ productKey / device1 / update` and `productKey / device1 / update / error`.
+	Matches all topics in the current tree of the topic hierarchy. For example, the topic `/ productKey /+/ update` can represent `/${ productKey }/ device1 / update` and `/${ productKey }/ device2 / update`.

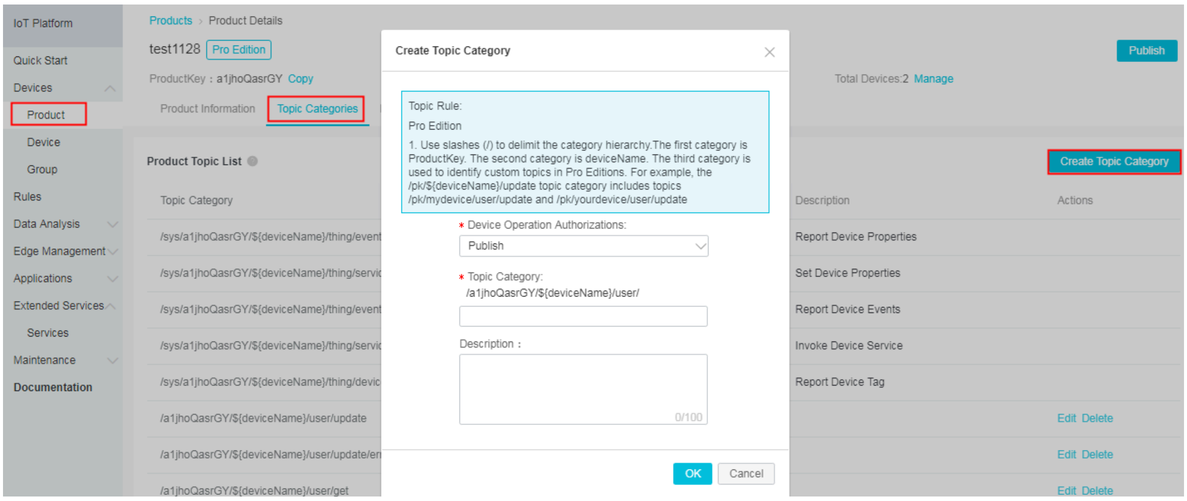
1.6.2 Create a topic category

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

Topic Rule:

Pro Edition

1. 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: /a1jhoQasrGY/${deviceName}/user/

Description : 0/100

- 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 Authorizat ions` as `Subscripti on` 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.

1.7 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

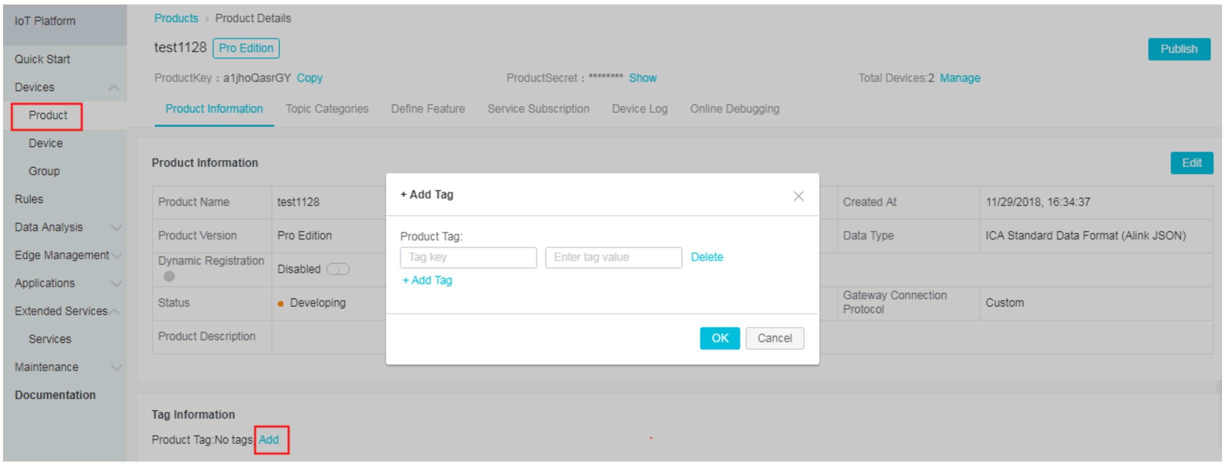
Product tags typically describe the information that is common to all devices of a product. For example, a tag can indicate a specific manufacturer, organization, physical size, or operating system. After a product has been created, you can create tags for it.

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.

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



test1128

ProductKey : a1jhoQasrGY ProductSecret : ******** Total Devices:2

Product Information

Product Name	test1128	Created At	11/29/2018, 16:34:37
Product Version	Pro Edition	Data Type	ICA Standard Data Format (Alink JSON)
Dynamic Registration	Disabled		
Status	Developing	Gateway Connection Protocol	Custom
Product Description			

Product Tag:

Product Tag:No tags

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.

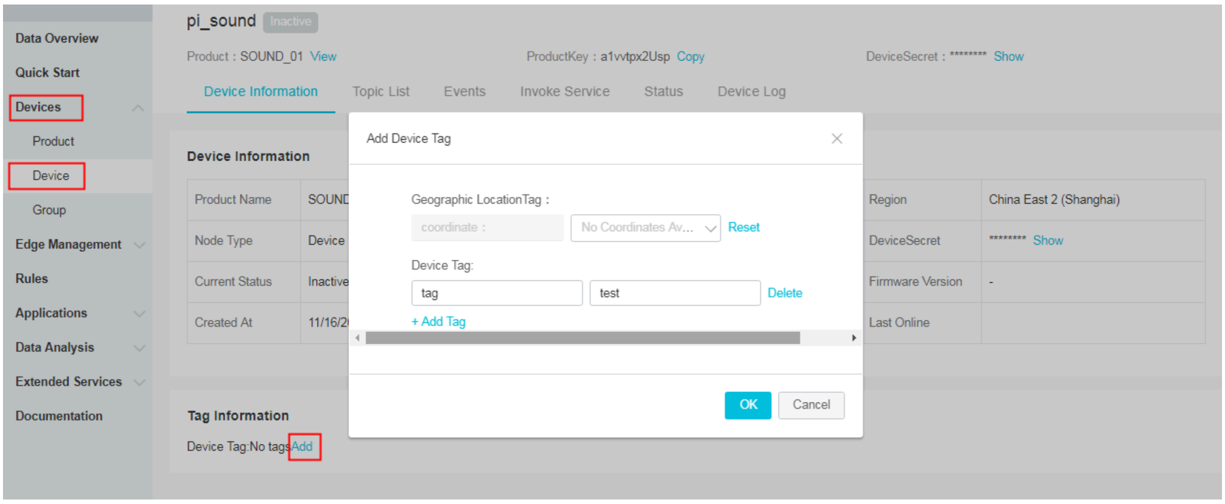
Device tags always follow the devices. You can include tag information in the messages reported to IoT Platform by devices. When you use the rules engine to forward these messages to other Alibaba Cloud services, the tag information is also forwarded to the targets.

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 (-) and dots (.), and cannot exceed 128 characters . A Chinese character is counted as 2 characters.



Product : SOUND_01

ProductKey : a1vvtpx2Usp

DeviceSecret : ********

Device Information

Product Name	SOUND	Region	China East 2 (Shanghai)
Node Type	Device	DeviceSecret	******** Show
Current Status	Inactive	Firmware Version	-
Created At	11/16/2	Last Online	

Geographic LocationTag : coordinate : No Coordinates Av...

Device Tag: tag test

Device Tag:No tags Add

Group tags

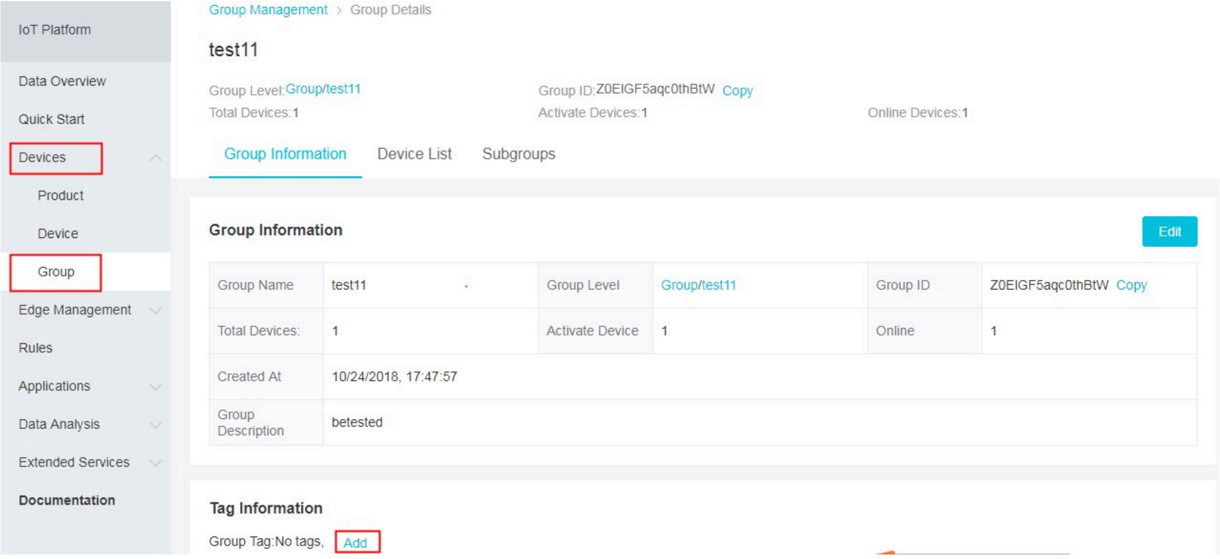
You can manage devices across products by grouping your devices. A group tag typically describe the general information of devices in the group and the sub-groups . For example, you can use region information as a group tag. After you have created a group, you can create tags for it.

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 (_) and hyphens (-), and cannot exceed 128 characters. A Chinese character is counted as 2 characters.



Group Management > Group Details

test11

Group Level:Group/test11 Group ID:Z0EIGF5aqc0thBtW Copy

Total Devices:1 Activate Devices:1 Online Devices:1

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.

1.8 Gateways and sub-devices

1.8.1 Gateways and sub-devices

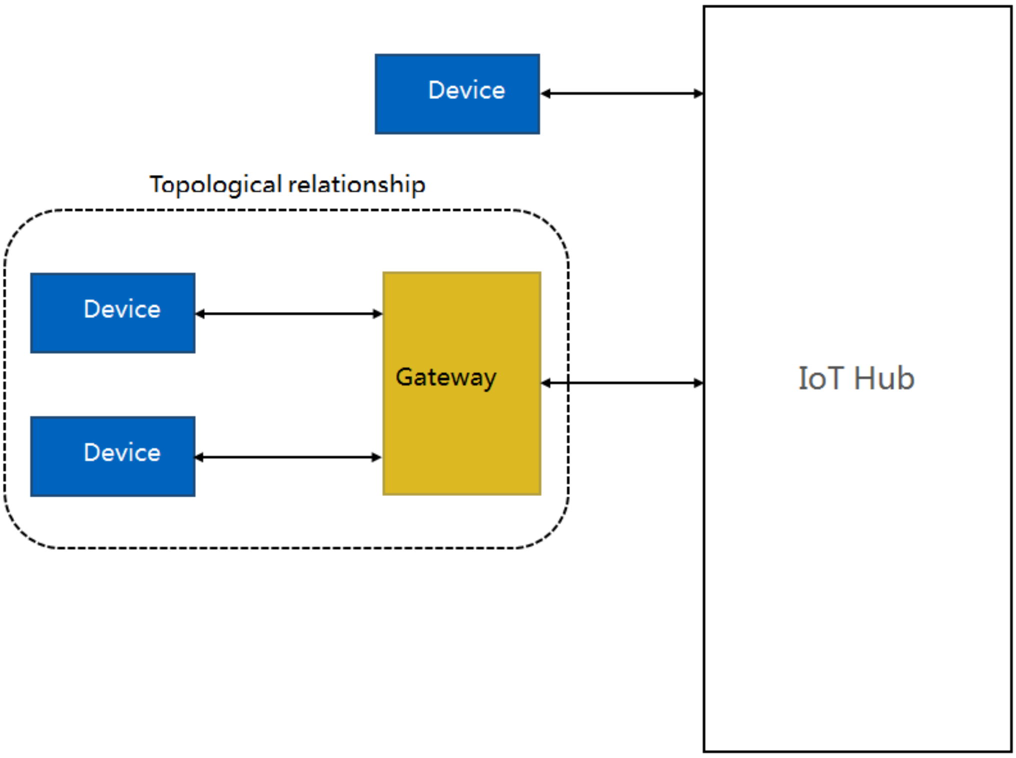
IoT Platform allows devices to connect to it directly, or be mounted as sub-devices to gateways that connect to IoT Platform.

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.

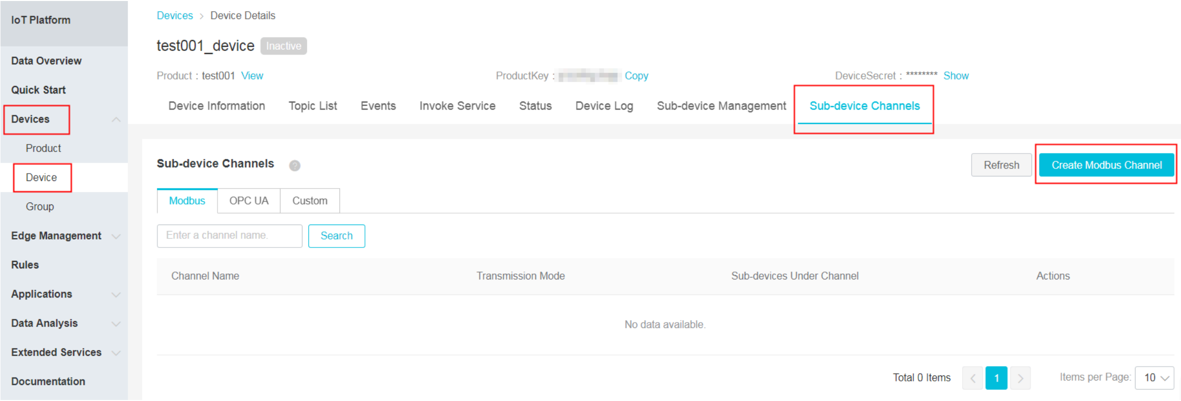
1. For more information about how to connect gateways to IoT Platform, see *Link Kit SDK*.
2. You can connect sub-devices to IoT Platform using either of the following two methods:
   - The *Unique-certificate-per-device authentication* method. This method requires you to install the device certificates (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.

1.8.2 Sub-device channels

You can create sub-device channels for Pro Edition gateway devices. Gateway devices can then use the management channels to manage sub-devices. Currently, IoT Platform supports three kinds of channels: Modbus protocol channels, OPC UA protocol channels, and custom protocol channels.

1. In the left-side navigation pane, click Devices > Device.
2. On the Devices page, find the gateway device for which you want to create channels, and click View next to it. You are directed to the Device Details page.

3. Click Sub-device Channels and then create sub-device management channels according to your required protocol.



test001_device Inactive

Product : test001 View ProductKey : [redacted] Copy DeviceSecret : ******** Show

Sub-device Channels

Enter a channel name.

Channel Name	Transmission Mode	Sub-devices Under Channel	Actions

No data available.

Total 0 Items < 1 > Items per Page: 10

- Modbus

  In the Modbus tab, click Create Modbus Channel and enter the required information in the dialog box.

Parameter	Description-
Channel Name	The channel identifier. It must be unique in the gateway device.
Transmission Mode	Supports RTU and TCP.
If you select RTU as the transmission mode, you must set the following parameters:	
Select Serial Port	For example, /dev/tty0 or /dev/tty1.
Baud Rate	Select a value from the drop-down list.
Data Bit	Supports the following data bit values: 5, 6, 7, and 8.
Check Bit	Supports no parity check, odd parity check, and even parity check.
Stop Bit	Support the following stop bit values: 1, 1.5, and 2.
If you select the transmission mode as TCP, you must set the following parameters:	
IP Address	Enter an IP address in dot-decimal notation.

Parameter	Description-
Port Number	Enter an integer in the range of 0-65535.

- OPC UA

  Click OPC UA > Create OCP UA Channel, and enter the required information in the dialog box.

Parameter	Description
Channel Name	The channel name must be unique in the gateway device.
Connection Address	For example, opc.tcp://localhost:4840
User Name	An optional parameter.
Password	An optional parameter.
Function Call Timeout	In seconds.

- Custom

  a. Click Custom > Create Customized Channel.

  b. Enter a channel name in the dialog box.

  c. Enter your customized configuration content.

  > Note:
  >
  > The configuration content must be in JSON format. We recommend that you prepare the JSON content in advance, and paste it in the box.

1.8.3 Sub-device management

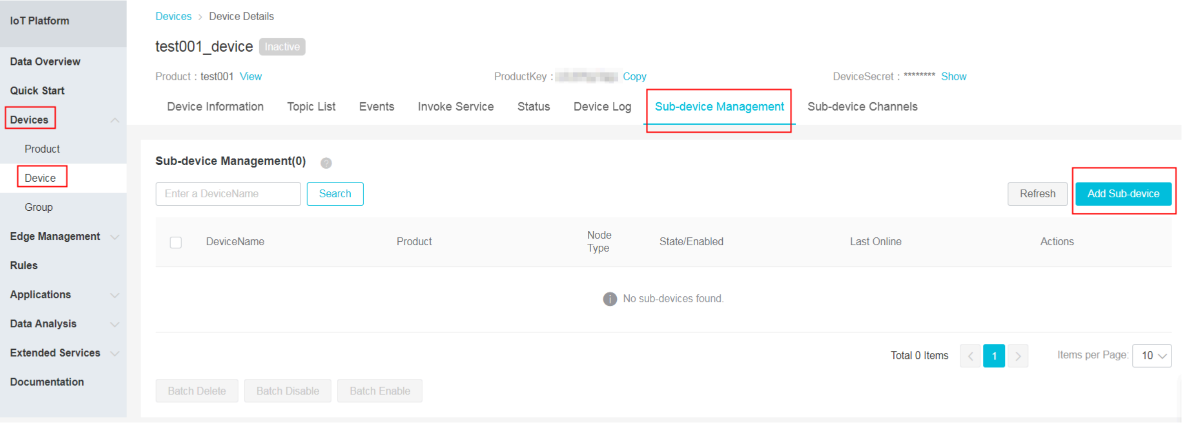
You can add sub-devices to a gateway device and send the TSL and the extended service information of the product (to which the sub-device belongs) to the gateway.

Prerequisites

- If the gateway connection protocol of a device is Modbus or OPC UA, before you connect the device to a gateway, you must create a corresponding sub-device channel for the gateway. For information about how to create sub-device channels , see the documentation about sub-device channels.
- Products and devices created before September 4, 2018 can be added to gateways as sub-devices. You can then build their topological relationships, but you cannot use sub-device channels or other custom configurations.

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.

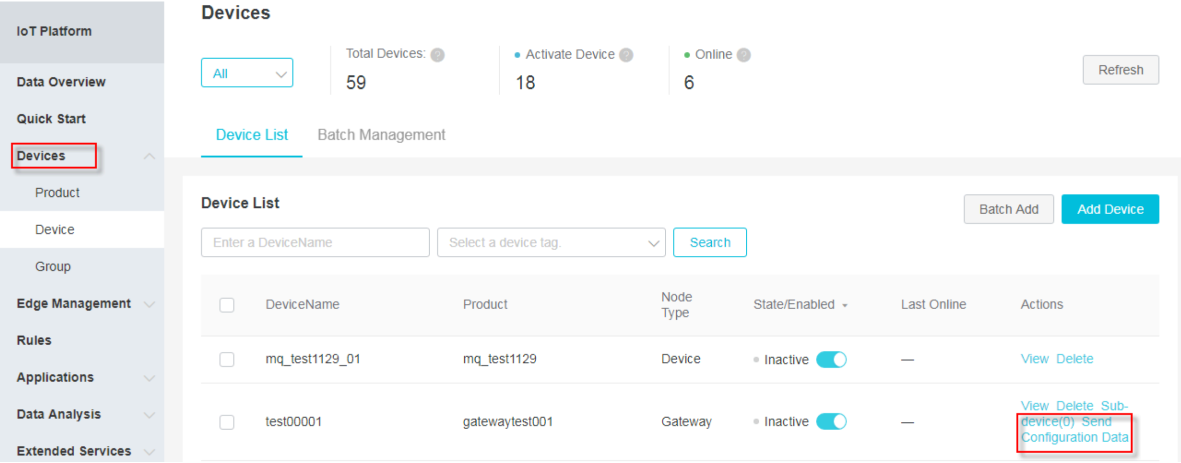


4. Enter the information of the sub-device in the dialog box.

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.
If the gateway connection protocol of the sub-device is Modbus, the following parameters are required.	
Associated Channel	Select a channel for the sub-device from the sub-device channels that you have created.
Slave Station Number	Enter an integer in the range of 1 - 247.
If the gateway connection protocol of the sub-device is OPC UA, the following parameters are required.	
Associated Channel	Select a channel for the sub-device from the sub-device channels that you have created.
Node Path	Enter a node path. For example, Objects/Device1. In this example, Objects is a fixed root node, and Device1 is the name of the device node path. Use / to separate node names.
If the gateway connection protocol of the sub-device is a custom protocol, you can set the following parameters.	

Parameter	Description
Associated Channel	Optional. Select a channel for the sub-device from the sub-device channels that you have created.
Custom Configuration	If you have selected an associated channel, you must customize the configuration. The custom configuration must be in JSON format.

5. After you have added sub-devices to a gateway, go back to the details page of the gateway device and click Send Configuration Data to assign the TSLs and extended service information of the products (to which the sub-devices belong) and the gateway connection configurations to the gateway.



Total Devices: 59

Activate Device 18

Online 6

Device List

	DeviceName	Product	Node Type	State/Enabled	Last Online	Actions
□	mq_test1129_01	mq_test1129	Device	Inactive	—	View Delete
□	test00001	gatewaytest001	Gateway	Inactive	—	View Delete Sub-device(0) Send Configuration Data

6. On the details page of the sub-device, you can view the gateway device information. Click Edit to modify the configuration information.

What's next

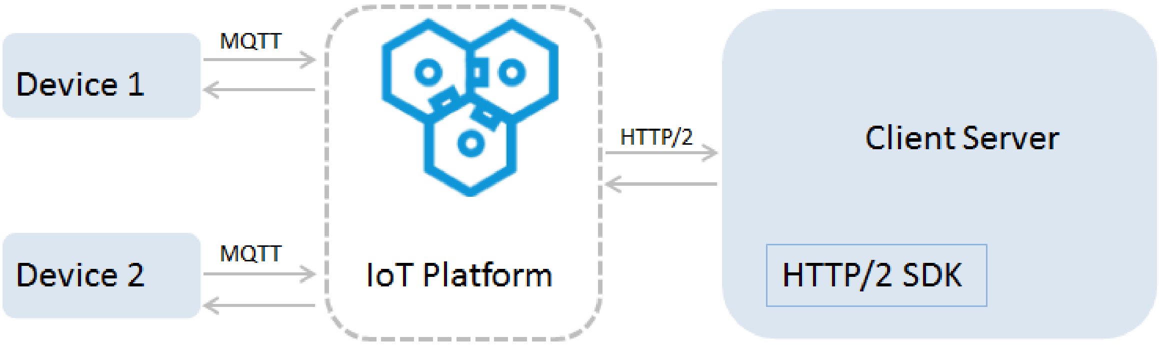
- If you want to develop your own devices and assign the configurations between the gateway device and the sub-device to the device client, see *Alink protocol*.

1.9 Service Subscription

1.9.1 What is Service Subscription?

Service clients can directly subscribe to device upload and status messages of products.

Currently, IoT Platform pushes messages through HTTP/2. After you configure the service subscription, IoT Platform pushes messages to your service client through HTTP/2. This means that you can use HTTP/2 SDKs to allow your enterprise server to directly receive messages from IoT Platform. HTTP/2 SDKs provide identity authentication, topic subscription, message sending and message receiving capabilities, and can be used to enable communication between devices and IoT Hub. Specifically, HTTP/2 SDKs allow you to transfer large numbers of messages between IoT Platform and your enterprise server, and support communication between devices and IoT Platform.



Note:

If you are using an old version of IoT Platform and Message Service is being used to transfer messages, you can upgrade your service subscription method to HTTP/2. If you want to continue using Message Service as your message transferring service, IoT Platform will push messages to Message Service, which means your clients must listen to your queues in Message Service in order to receive messages.

1.9.2 Development guide

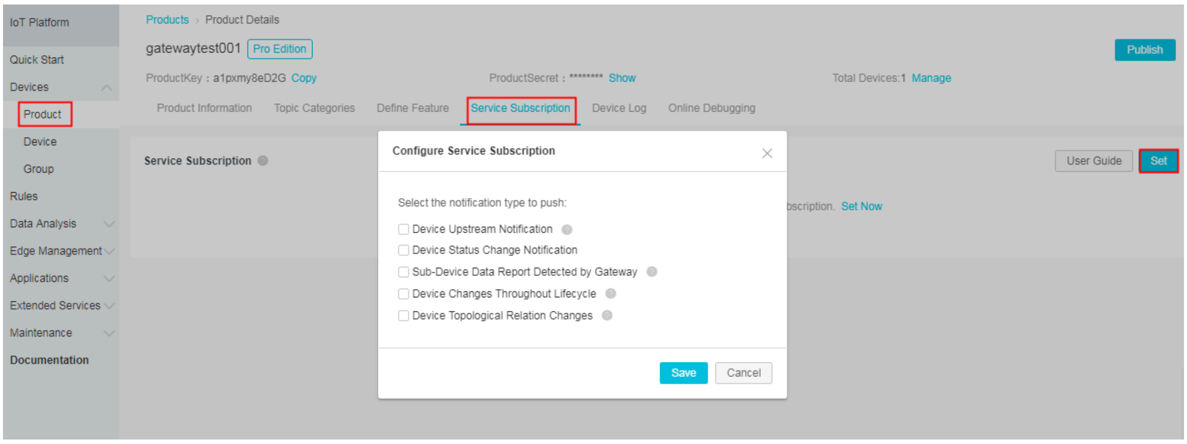
This article introduces how to configure the service subscription, connect to the HTTP/2 SDK, authenticate identity, and configure the message-receiving interface.

Specifically, this section details the development process of the service subscription. For more information, see 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.



Configure Service Subscription

Select the notification type to push:

- □ Device Upstream Notification
- □ Device Status Change Notification
- □ Sub-Device Data Report Detected by Gateway
- □ Device Changes Throughout Lifecycle
- □ Device Topological Relation Changes

Save Cancel

- *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.

  Pro Edition devices report custom data and TSL data of properties, events, responses to property setting requests, and responses to service calling requests . Basic Edition devices only report custom data.

  For example, a Pro Edition 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*: This is a specific notification type of Pro Edition products. 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*: This is a specific notification type of Pro Edition products. It includes notifications about creation and removal of the topological relation between a gateway and its sub-devices.
- *Device Changes Throughout Lifecycle*: This is a specific notification type of Pro Edition products. It includes notifications about device creation, deletion, disabling, and enabling.

Connect to the SDK

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

```
< dependency >
    < groupId > com . aliyun . openservic  es </ 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   accessKeyI  D
         String   accessKey  = " xxxxxxxxxx  xxxxx ";
        //  Your   account   AccessKeyS  ecret
         String   accessSecr  et  = " 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   configurat  ion
         Profile   profile  =  Profile . getAccessK  eyProfile (
 endPoint ,  regionId ,  accessKey ,  accessSecr  et );

        //  Construct   the   client
         MessageCli  ent   client  =  MessageCli  entFactory .
 messageCli  ent ( profile );

        //  Receive   data
         client . connect ( messageTok  en  -> {
             Message   m  =  messageTok  en . getMessage ();
             System . out . println (" receive   message   from  " +
 m );
             return   MessageCal  lback . Action . CommitSucc  ess ;
        });
```

The value of `accessKey` is the AccessKeyID of your account, and the value of `accessSecr et` 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 `setMessage Listener ()` of `messageCli ent` 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:

```
 MessageCal  lback   messageCal  lback  =  new   MessageCal  lback ()
 {
    @ Override
     public   Action   consume ( MessageTok  en   messageTok  en ) {
         Message   m  =  messageTok  en . getMessage ();
         log . info (" receive  : " +  new   String ( messageTok  en .
 getMessage (). getPayload ()));
         return   MessageCal  lback . Action . CommitSucc  ess ;
    }
};
```

```
messageClient . setMessageListener ("/${ 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 . setMessageListener ("/${ 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, `/${ YourProductKey }/${ YourDeviceName }/#`.

    Example:

    ```
    messageClient . setMessageListener ("/ alEddfaXXX X /
    device1 /#", messageCallback );
    ```

```
// When the received message matches with the
 specified topic, for example, "/alEddfaXXXX/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:

```
 messageClient.setMessageListener(messageCallback);
// 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 `MessageCallback.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 `MessageClient.setManualAcks` to configure for replying ACKs manually.**

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

    The method to manually reply ACKs is as follows:

```
 messageClient.ack(messageToken);
```

Message body format

- **Device status notification:**

```
{
    "status":"online|offline",
    "productKey":"12345565569",
    "deviceName":"deviceName1234",
    "time":"2018-08-31 15:32:28.205",
    "utcTime":"2018-08-31T07:32:28.205Z",
    "lastTime":"2018-08-31 15:32:28.195",
    " utcLastTim  e ":" 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 ",
" deviceSecr  et " : "",
" messageCre  ateTime ":  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 ",
" gwProductK  ey ": " 1234556554 ",
" gwDeviceNa  me ": " deviceName  1234 ",
" devices ": [
{
" iotId ": " 4z819VQHk6  VSLmmBJfrf  00107ee201 ",
" productKey ": " 1234556556  9 ",
" deviceName ": " deviceName  1234 "
}
],
" messageCre  ateTime ":  1510292739  881
}
```

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 ",
    " gwProductK  ey ":" 1234556554 ",
    " gwDeviceNa  me ":" 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.

1.9.3 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.
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.

Item	Limit description
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.

1.10 Device group

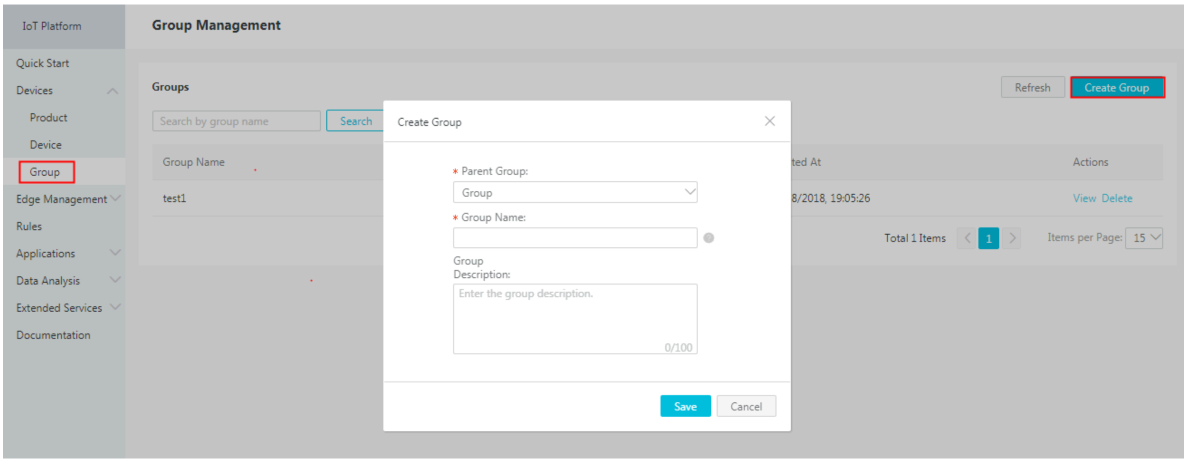
IoT Platform supports device groups. You can assign devices from different products to the same group. This article introduces how to create and manage device groups in the IoT Platform console.

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.

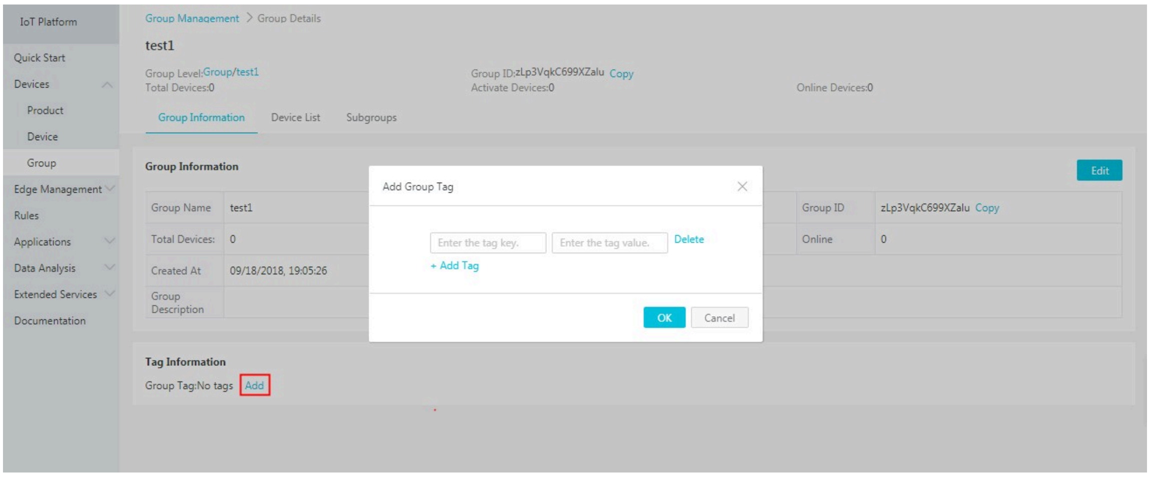
4. On the Group Management page, click View to view the Group Details page of the corresponding group.
5. (Optional) Add tags for the group. Tags can be used as group identifiers when you manage your groups.

   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.



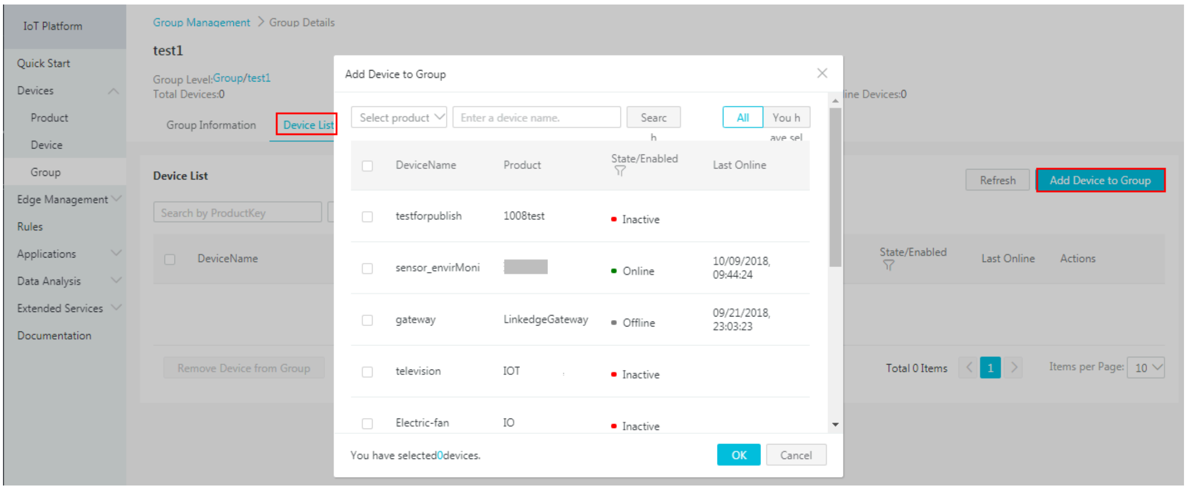
Group Information

Group Name	test1	Group ID	zLp3VqkC699XZalu Copy
Total Devices:	0	Online	0
Created At	09/18/2018, 19:05:26		
Group Description			

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.



□	DeviceName	Product	State/Enabled	Last Online
□	testforpublish	1008test	● Inactive	
□	sensor_envirMoni		● Online	10/09/2018, 09:44:24
□	gateway	LinkedgeGateway	● Offline	09/21/2018, 23:03:23
□	television	IOT	● Inactive	
□	Electric-fan	IO	● Inactive	

You have selected0devices.

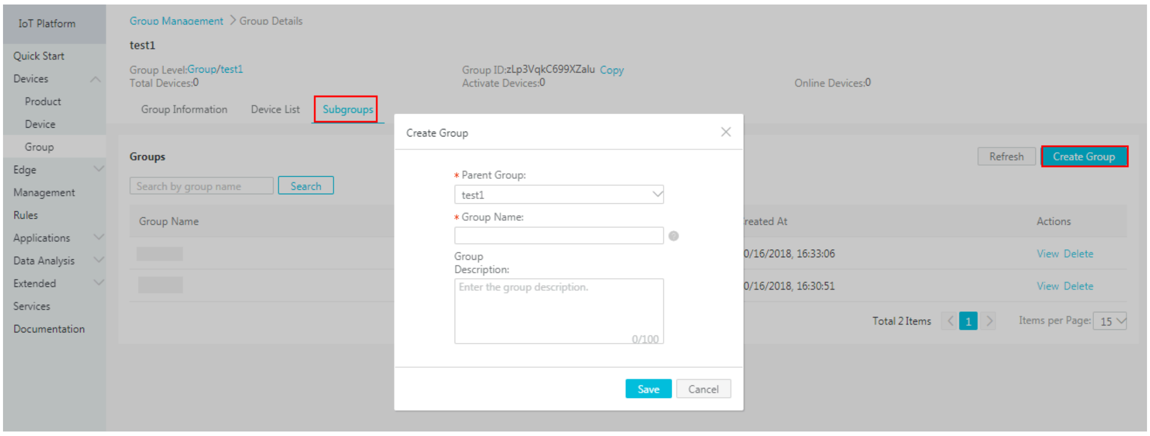
There are two buttons at the upper-right corner of the Add Device to Group page:.

- 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.



test1

Group Level:Group/test1
Total Devices:0

Group ID:zLp3VqkC699XZalu Copy
Activate Devices:0

Online Devices:0

0/16/2018, 16:33:06

0/16/2018, 16:30:51

* Parent Group: test1

* Group Name:

Group Description: Enter the group description. 0/100

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.

2 Rules

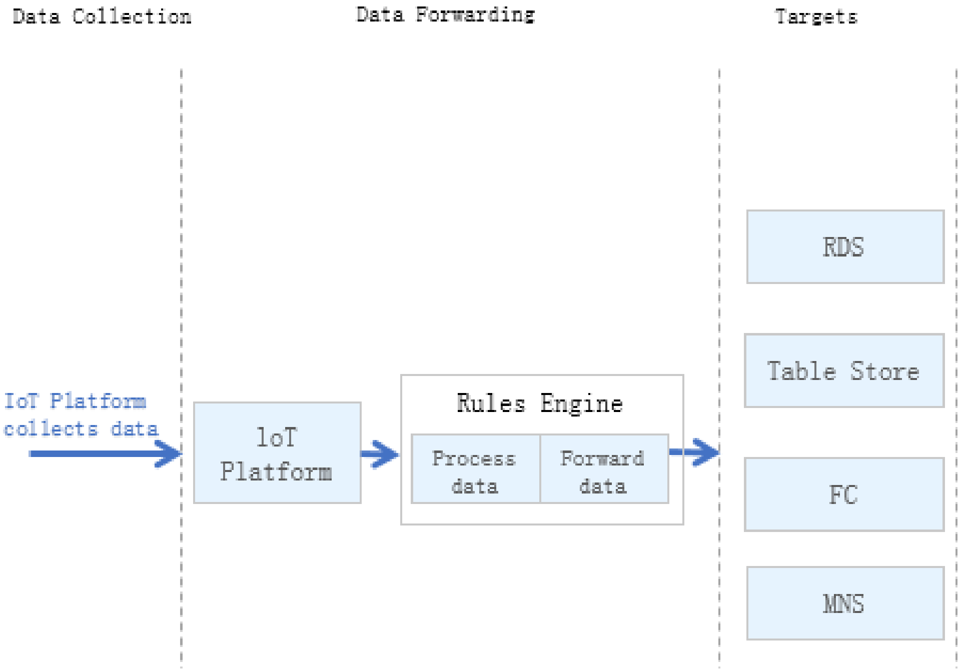
2.1 Data Forwarding

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 communicat ion.
- 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:

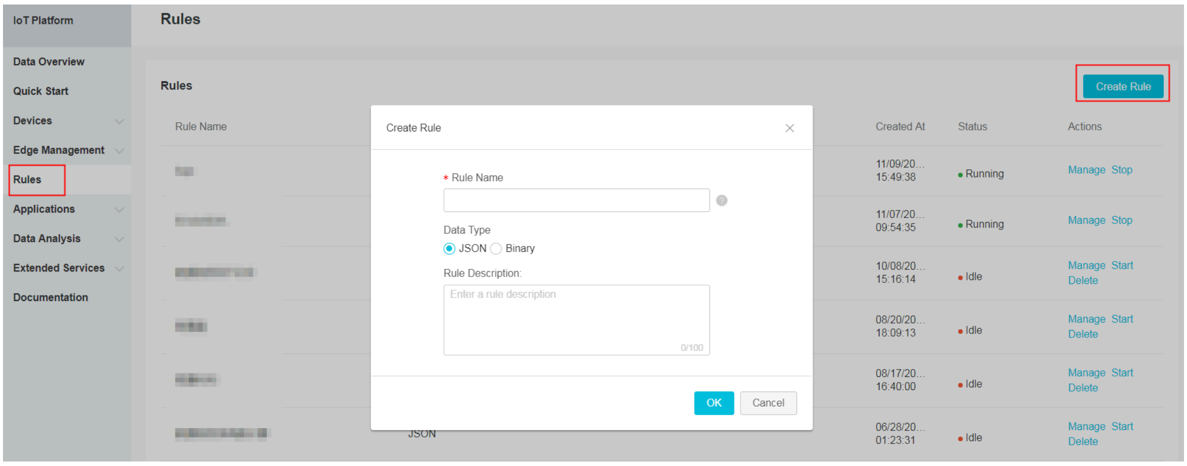
- The rule engine processes data based on topics. You can use the rule engine to process device data only when devices are communicating with each other by using topics.
- The rule engine processes the data in topics using SQL.
- SQL subqueries and the use of the LIKE operator are currently not supported.
- 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 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.



Rule Name		Created At	Status	Actions
[redacted]		11/09/20... 15:49:38	● Running	Manage Stop
[redacted]		11/07/20... 09:54:35	● Running	Manage Stop
[redacted]		10/08/20... 15:16:14	● Idle	Manage Start Delete
[redacted]		08/20/20... 18:09:13	● Idle	Manage Start Delete
[redacted]		08/17/20... 16:40:00	● Idle	Manage Start Delete
[redacted]	JSON	06/28/20... 01:23:31	● Idle	Manage Start Delete

Create Rule

* Rule Name

Data Type
● JSON ○ Binary

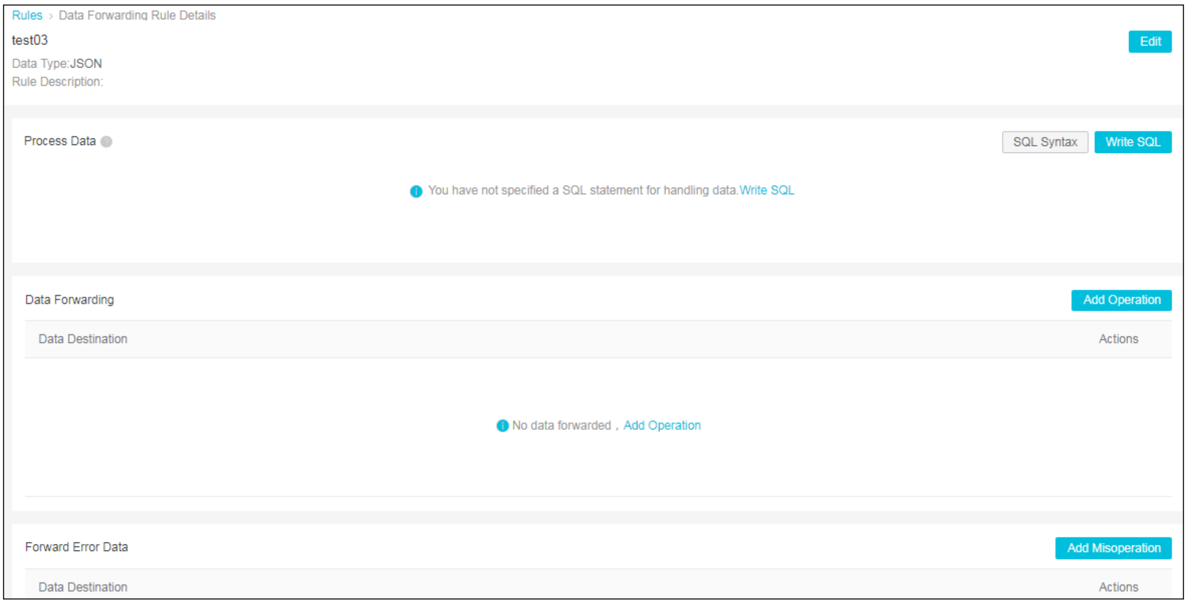
Rule Description:
Enter a rule description 0/100

OK Cancel

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.

Parameter	Description
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 and 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.



Rules > Data Forwarding Rule Details

test03

Data Type:JSON

Rule Description:

Process Data

You have not specified a SQL statement for handling data.Write SQL

Data Forwarding

Data Destination	Actions

No data forwarded , Add Operation

Forward Error Data

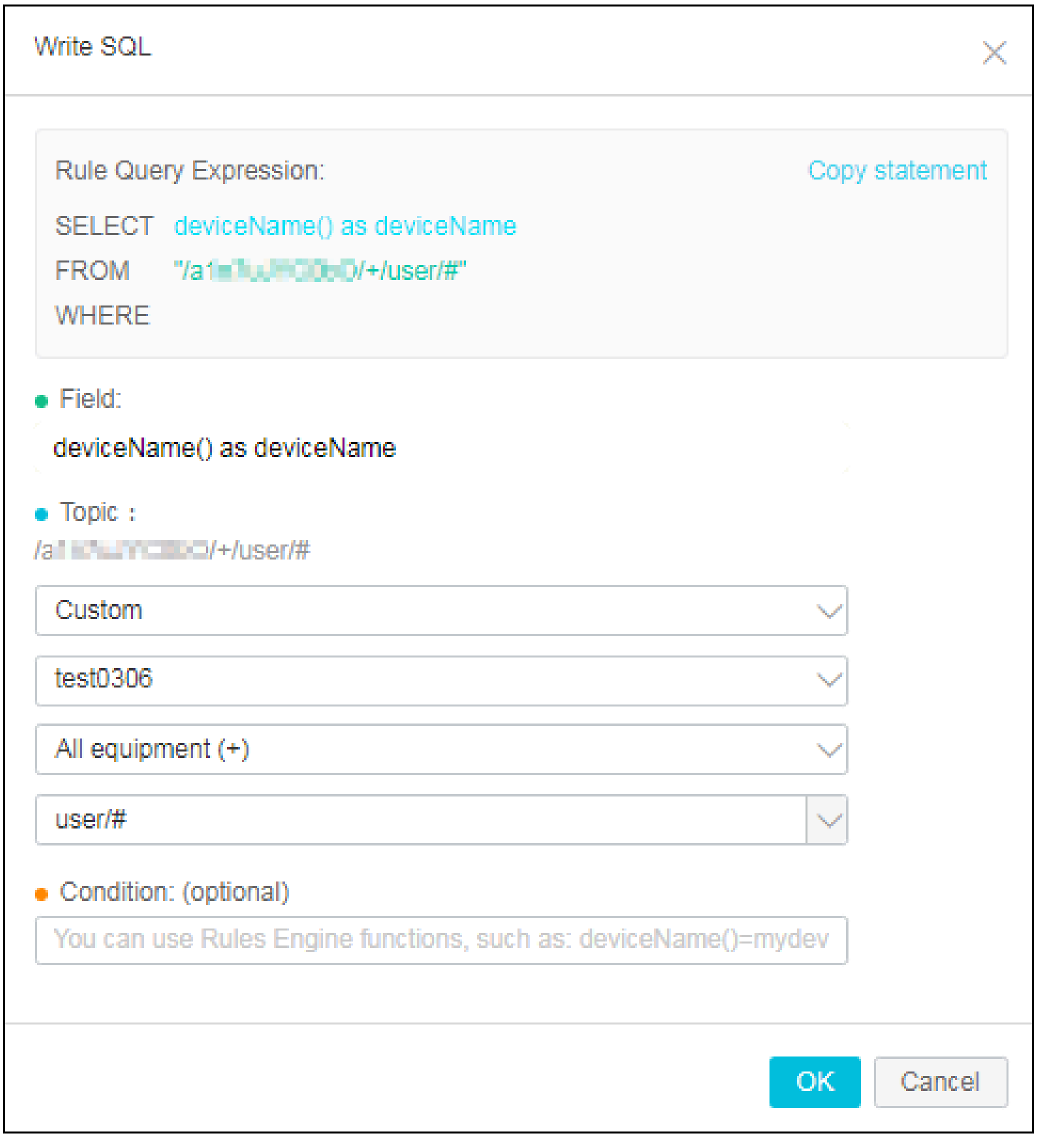
Data Destination	Actions

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:

SELECT deviceName() as deviceName
FROM "/a1[redacted]/+/user/#"
WHERE

● Field:

deviceName() as deviceName

● Topic :

/a[redacted]/+/user/#

Custom

test0306

All equipment (+)

user/#

● Condition: (optional)

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

The parameters to be configured are as follows. For more information, see *SQL statements* and *Functions*.

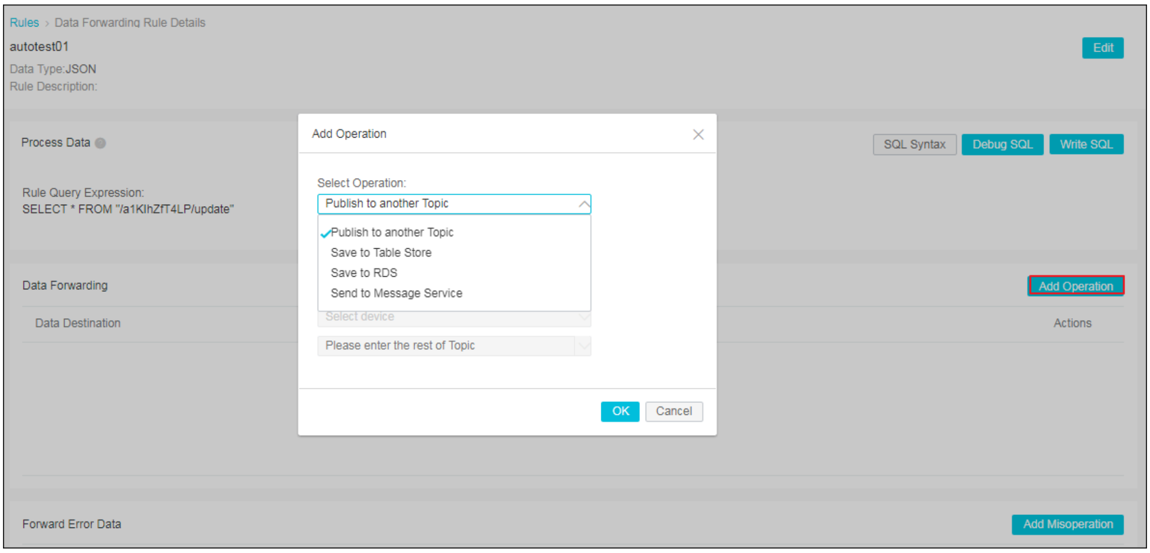
Parameter	Description
Rule Query Expression	The system will display the complete SQL statement here according to the values of `Field` , `Topic` , and `Condition` .
Field	Specify the message fields that this rule will retrieve from the message contents. For example, if you enter `deviceName () as deviceName` , the rule will retrieve the device names from the messages. For message content data, see *Data Format*.

Parameter	Description
Topic	Select the topics whose messages are to be processed by this rule. Topic types: · Custom: The messages are from custom topics. Wildcards `+` and `#` 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 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.



Rules > Data Forwarding Rule Details

autotest01

Data Type:JSON

Rule Description:

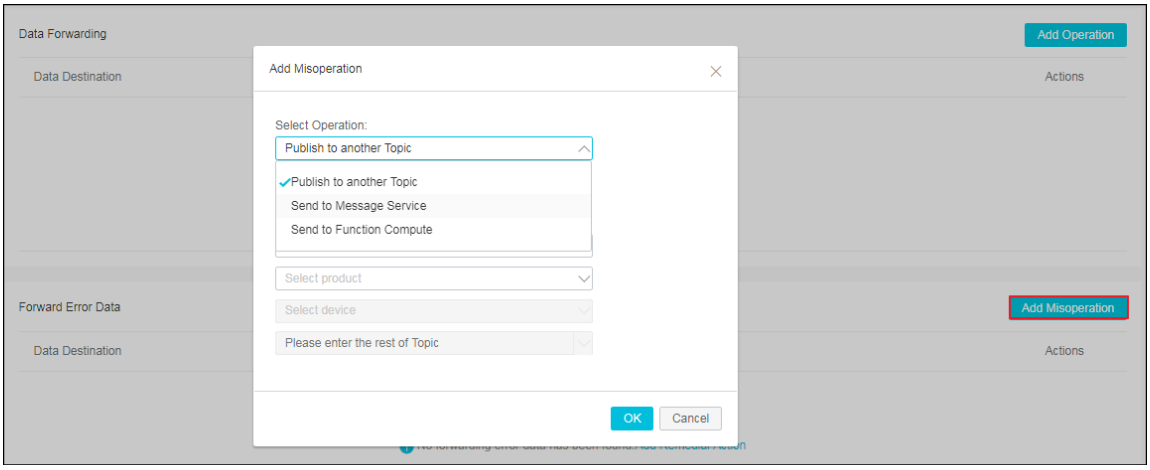
Rule Query Expression:
SELECT * FROM "/a1KIhZfT4LP/update"

Add Operation

Select Operation: Publish to another Topic

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.



Select Operation: Publish to another Topic

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 ":"",
" base64Orig  inalPayloa  d ":"",
" failures ":[
{
```

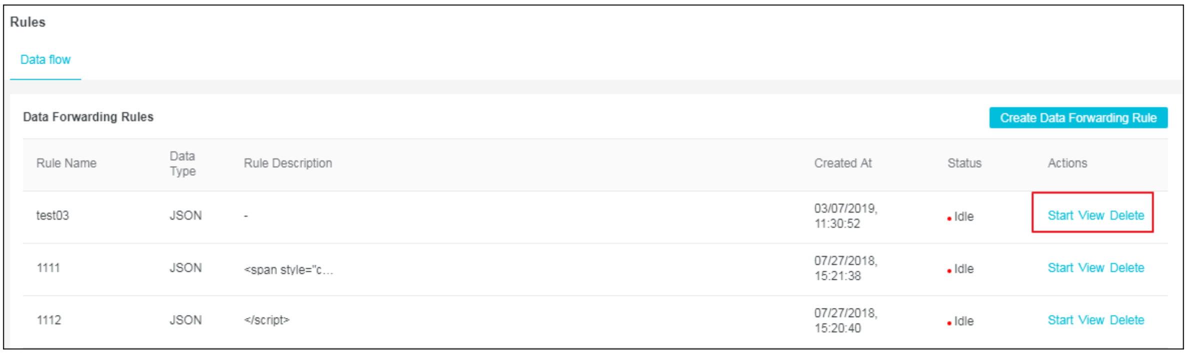
```
" actionType ":" OTS ",
" actionRegi  on ":" cn - shanghai ",
" actionReso  urce ":" table1 ",
" errorMessa  ge ":""
},
{
" actionType ":" RDS ",
" actionRegi  on ":" cn - shanghai ",
" actionReso  urce ":" instance1 / table1 ",
" errorMessa  ge ":""
}
]

}
```

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 .
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 Forwarding Rules

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:

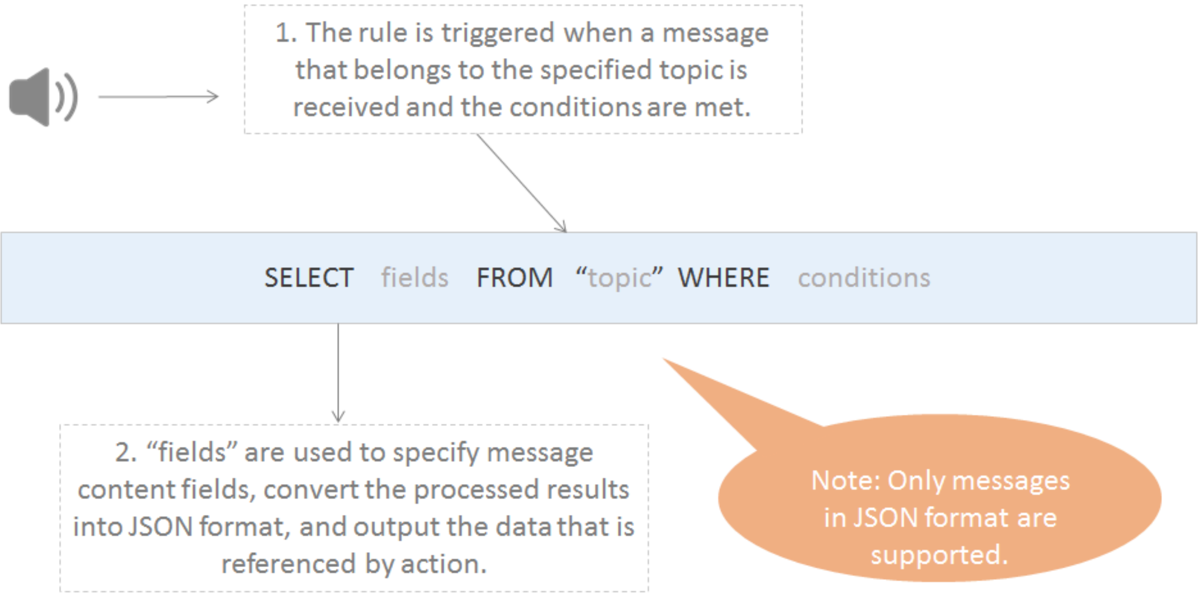
  Rules that are in a running state cannot be deleted.
- Click Stop to disable this rule.

2.1.3 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.



1. The rule is triggered when a message that belongs to the specified topic is received and the conditions are met.

SELECT fields FROM “topic” WHERE conditions

2. “fields” are used to specify message content fields, convert the processed results into JSON format, and output the data that is referenced by action.

Note: Only messages in JSON format are supported.

```
 For   example ,  an   environmen  tal   sensor   that   is
 typically   used   for   fire   detection   and   collecting
 temperatur  e ,  humidity ,  and   atmospheri  c   pressure   data ,
 reports   the   following   data :
{
" temperatur  e ": 25 . 1
" humidity ": 65
" pressure ": 101 . 5
" location ":" xxx , xxx "
}
 Assume   that   you   need   to   set   an   alarm   that   is
 triggered   when   the   temperatur  e   is   higher   than   38 °
 C   and   the   humidity   is   lower   than   40 %,  write   the
 following   SQL   statement   as   a   rule :
 SELECT   temperatur  e   as   t ,  deviceName ()  as   deviceName ,
 location   FROM  / ProductA /+/ update   WHERE   temperatur  e  >  38
   and   humidity  <  40
 If   the   reported   data   meets   the   rule   parameters ,  the
   rule   is   triggered   and   the   temperatur  e   data   is
 parsed   to   obtain   the   informatio  n   about   temperatur  e ,
 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   temperatur  e   as   t ,
  deviceName ()  as   deviceName ,  location "  that   is   provided
    in   the   previous   example ,  temperatur  e   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 expression s.

  ```
  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%’ .

2.1.4 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.

Function name	Description
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.
nanvl(value, default)	Returns the value of a property. If the value of the property is null, the function returns default.
newuuid()	Returns a random UUID.
payload(textEncoding)	Returns the string generated by encoding the message payload that is sent by a device. The default encoding is UTF-8, which means that payload() and payload(‘utf-8’) will return the same result.
power(n,m)	Raises number n to power m.

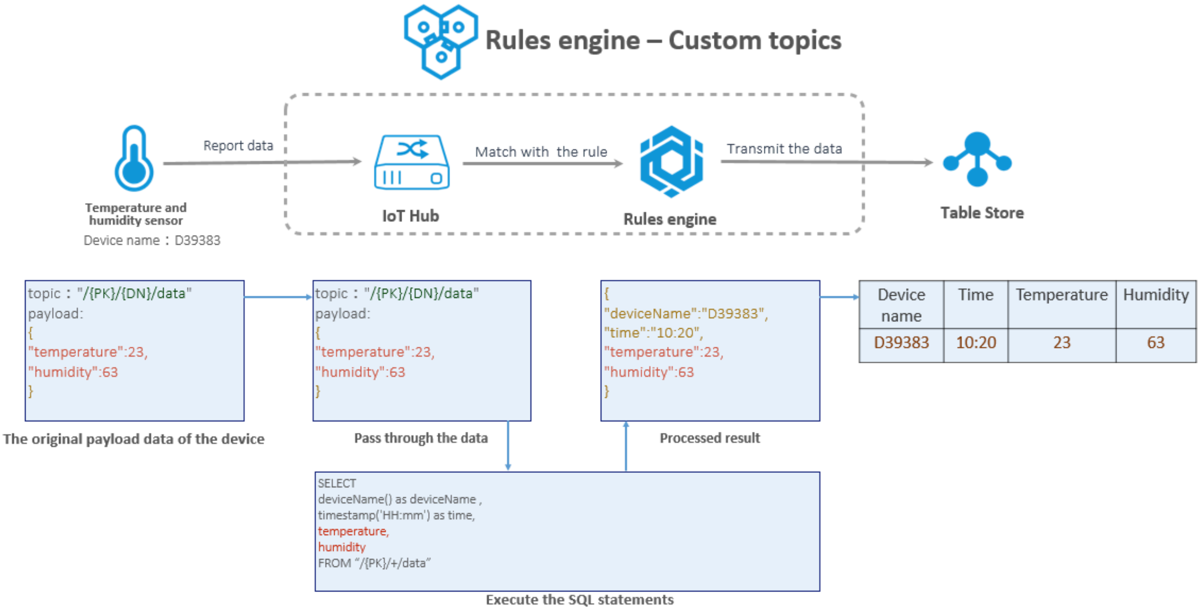
Function name	Description
rand()	Returns a random number greater than or equal to 0 and less than 1.
replace(source , substring, replacement)	Replaces a specific column. Example: replace(field,’ a’ ,’ 1’).
sin(n)	Returns the sine of n.
sinh(n)	Returns the hyperbolic sine of n.
tan(n)	Returns the tangent of n.
tanh(n)	Returns the hyperbolic tangent of n.
timestamp(format)	Returns the formatted 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: timestamp() = 1543373798943, timestamp('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'') = 2018-11-28T10:56:38Z.
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: timestamp_utc() = 1543373798943, timestamp_utc('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'') = 2018-11-28T02:56:38Z
topic(number)	Returns a segment of a topic. For example, for topic /abcdef/ghi, if you use the function topic(), “ /abcdef/ghi” will be returned; If you use the function topic(1), “ abcdef” will be returned; If you use the function topic(2), “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.

2.1.5 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:



```
topic："/{PK}/{DN}/data"
payload:
{
"temperature":23,
"humidity":63
}
```

```
topic："/{PK}/{DN}/data"
payload:
{
"temperature":23,
"humidity":63
}
```

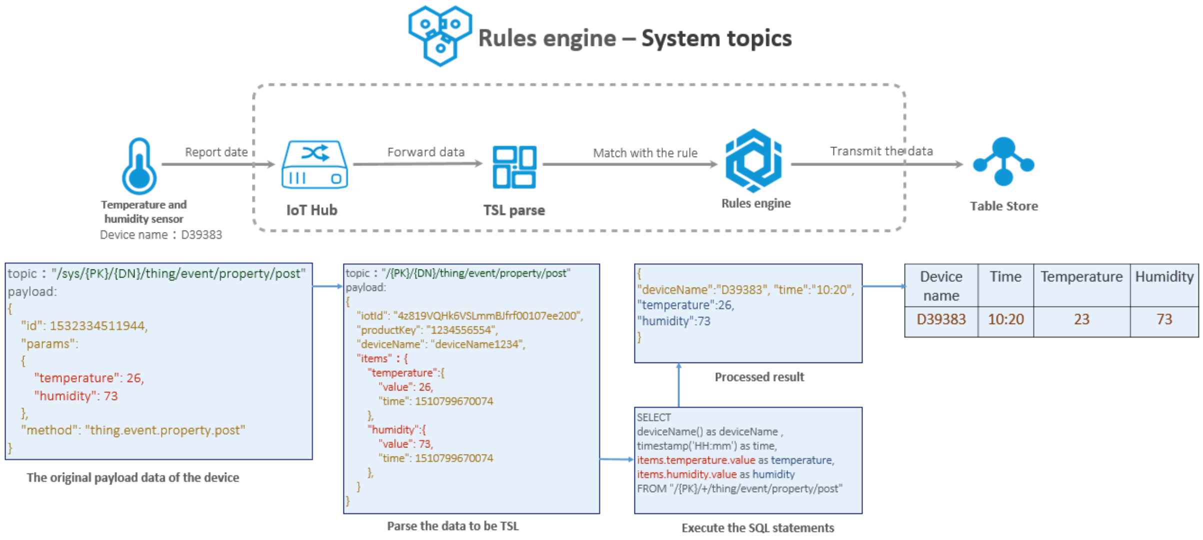
```
{
"deviceName":"D39383",
"time":"10:20",
"temperature":23,
"humidity":63
}
```

Device name	Time	Temperature	Humidity
D39383	10:20	23	63

```
SELECT
deviceName() as deviceName ,
timestamp('HH:mm') as time,
temperature,
humidity
FROM "/{PK}/+/data"
```

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 (Pro Edition)*. The following figure shows the data forwarding procedure:



```
topic："/sys/{PK}/{DN}/thing/event/property/post"
payload:
{
  "id": 1532334511944,
  "params":
  {
    "temperature": 26,
    "humidity": 73
  },
  "method": "thing.event.property.post"
}
```

The original payload data of the device

```
topic："/{PK}/{DN}/thing/event/property/post"
payload:
{
  "iotId": "4z819VQHk6VSLmmBJfrf00107ee200",
  "productKey": "1234556554",
  "deviceName": "deviceName1234",
  "items" : {
    "temperature":{
      "value": 26,
      "time": 1510799670074
    },
    "humidity":{
      "value": 73,
      "time": 1510799670074
    },
  }
}
```

Parse the data to be TSL

```
SELECT
deviceName() as deviceName ,
timestamp('HH:mm') as time,
items.temperature.value as temperature,
items.humidity.value as humidity
FROM "/{PK}/+/thing/event/property/post"
```

Execute the SQL statements

```
{
"deviceName":"D39383", "time":"10:20",
"temperature":26,
"humidity":73
}
```

Processed result

Device name	Time	Temperature	Humidity
D39383	10:20	23	73

Note:

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

2.1.6 Data format (Pro Edition)

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 Pro edition 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 ",
" deviceSecr  et " : "",
" messageCre  ateTime ":  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 ",
" gwProductK  ey ": " 1234556554 ",
" gwDeviceNa  me ": " deviceName  1234 ",
" devices ": [
        {
" iotId ": " 4z819VQHk6  VSLmmBJfrf  00107ee201 ",
" productKey ": " 1234556556  9 ",
" deviceName ": " deviceName  1234 "
       }
    ],

" messageCre  ateTime ":  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 ",
    " gwProductK  ey ":" 1234556554 ",
    " gwDeviceNa  me ":" 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.

2.1.7 Regions and zones

Before you a create 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 2-1: 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

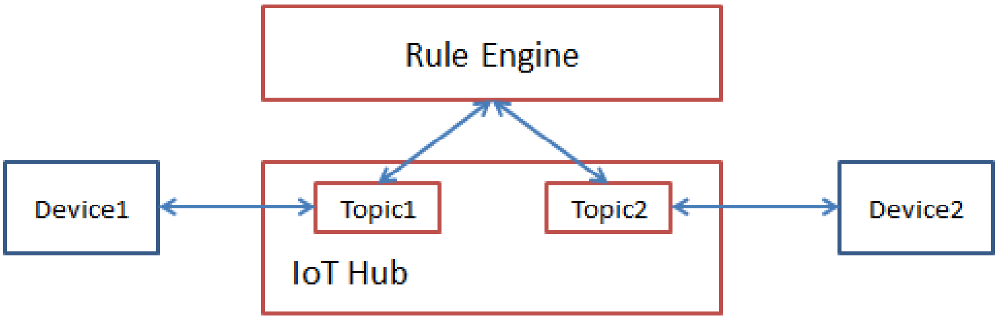
You can forward the data that is processed based on SQL rules to another topic for machine-to-machine (M2M) communication and other applications.

Prerequisites

Before configuring forwarding, follow the instructions in *Create and configure a rule* to write a SQL script and filter the data.

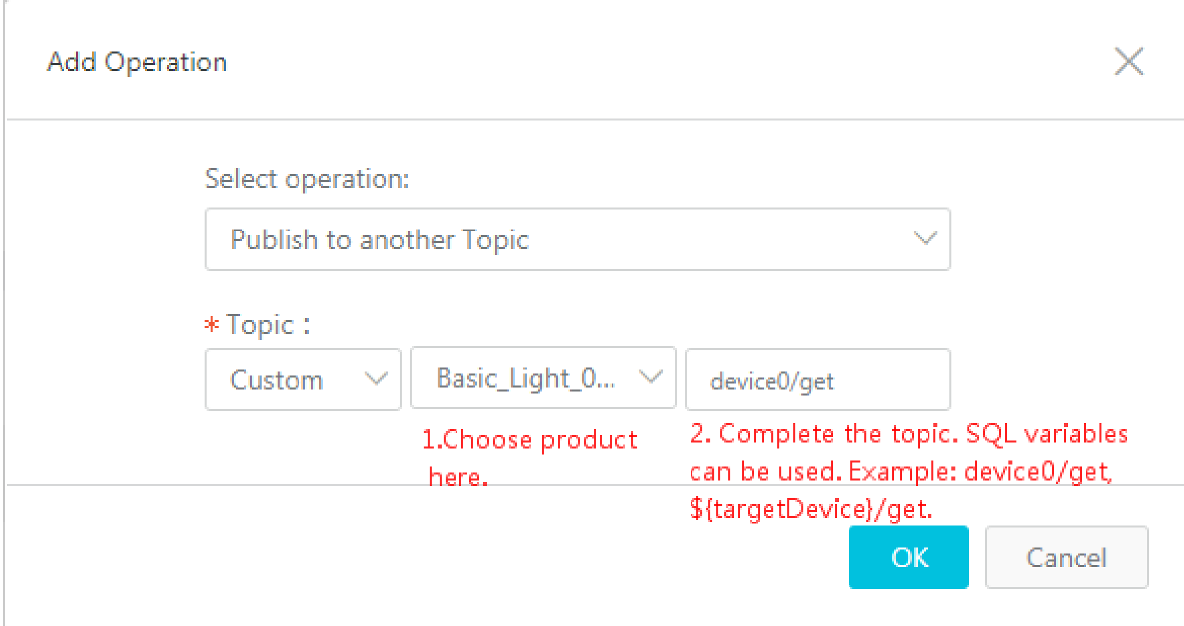
Context

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 : Custom | Basic_Light_0... | device0/get

1.Choose product here.

2. Complete the topic. SQL variables can be used. Example: device0/get, ${targetDevice}/get.

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

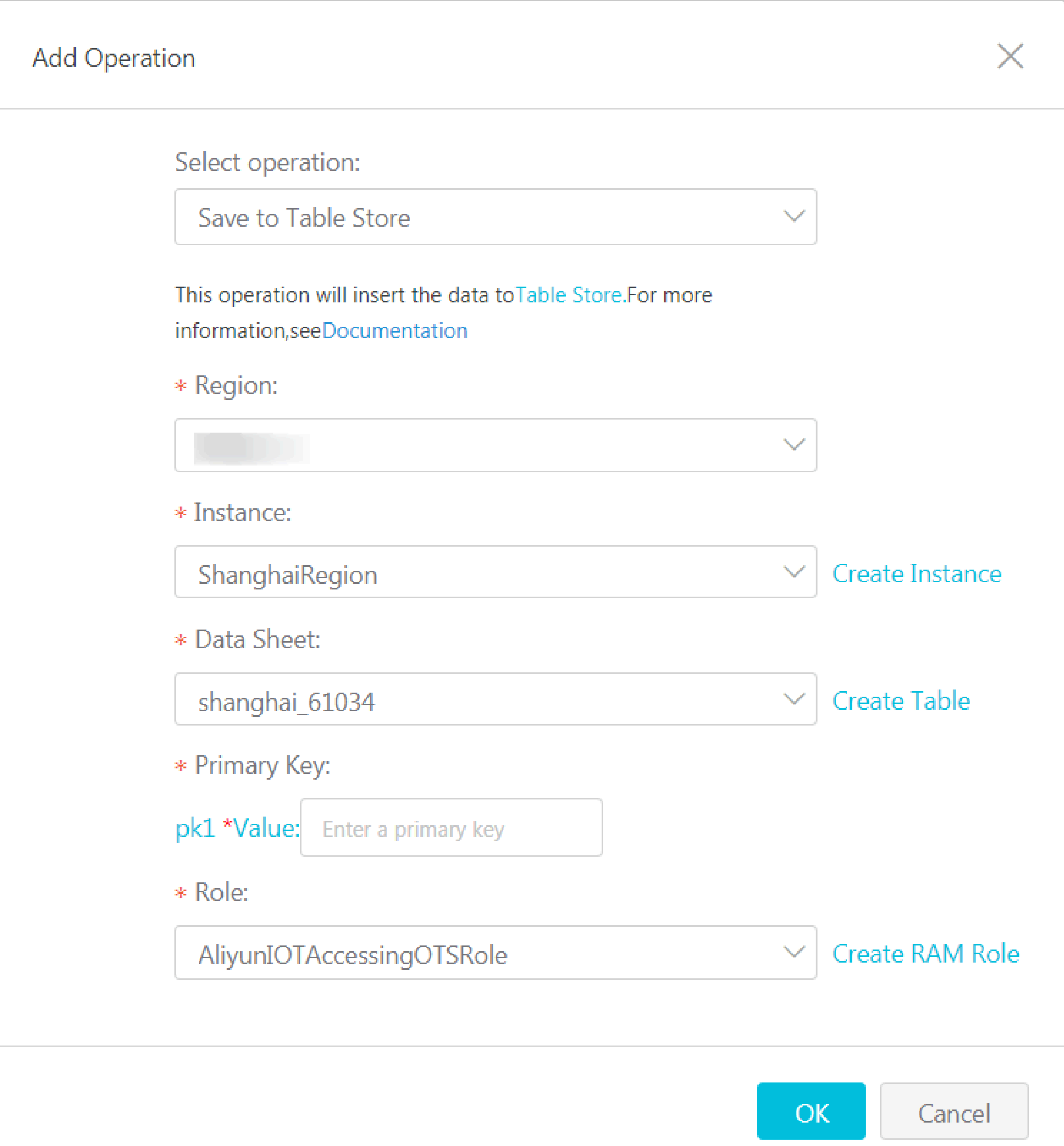
You can configure the rules engine to forward the processed data to Table Store.

Prerequisites

Before configuring forwarding, follow the instructions in *Create and configure a rule* to write a SQL script to filter the data.

Procedure

1. Click Add Operation next to Data Forwarding to open the Add Operation page. Select Save to Table Store .



Add Operation

Select operation: Save to Table Store

This operation will insert the data toTable Store.For more information,seeDocumentation

* Region: [redacted]

* Instance: ShanghaiRegion

* Data Sheet: shanghai_61034

* Primary Key:

pk1 *Value: Enter a primary key

* Role: AliyunIOTAccessingOTSRole

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

- Select Operation: Select Table Store.
- Region, Instance, and Table: Specify each of these fields for the table to which you want to forward data.
- Primary Key Field: All tables in Table Store have primary key columns. After you have selected the table to forward data, the console automatically reads the

primary key fields of this table. You need to configure the values of the primary key fields.

- Role: Grant IoT Platform permission to write data to Table Store. First create a role that includes permission to write data to Table Store and assign this role to the rules engine. The rules engine can now write processed data to a table.

What's next

Example

The JSON data record `{" device ":" bike "," product ":" xxx "," data2 ": [{...}]}` is extracted using SQL. This JSON data record needs to be stored in Table Store. The primary key columns in the destination table are `device` , `product` , and `id` .

Configuration and effects:

1. Set the value of the primary key field "device" to `${ device }` in the console. When a message arrives that triggers the forwarding rule, the value of the device field in the JSON data record will be saved under the device column in the destination table. The preceding configuration and effects resemble those for the primary key field "product".

   Note:

   `${}` is an escape character. If you do not use this escape character, the constant you specify as the value of the primary key field will be saved to the primary key column.

2. The forwarding rule will automatically detect the auto-increment column. The auto-increment column will be automatically assigned a unique value every time a new record is inserted into the table. The values in this column cannot be edited.
3. IoT Platform can automatically parse values of the non-primary key fields included in the JSON data record and create corresponding columns for the destination table . In this example, two columns, data1 and data2, will automatically be created, and the corresponding values will be saved under each column.

   Note:

   Currently, only top-level JSON structure can be parsed. Parsing of nested JSON structures is not supported. Therefore, in this example, the entire JSON object with its nested structure will be saved under the data2 column. The nested JSON structure will not be further parsed. No additional columns will be created to save the nested elements.

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

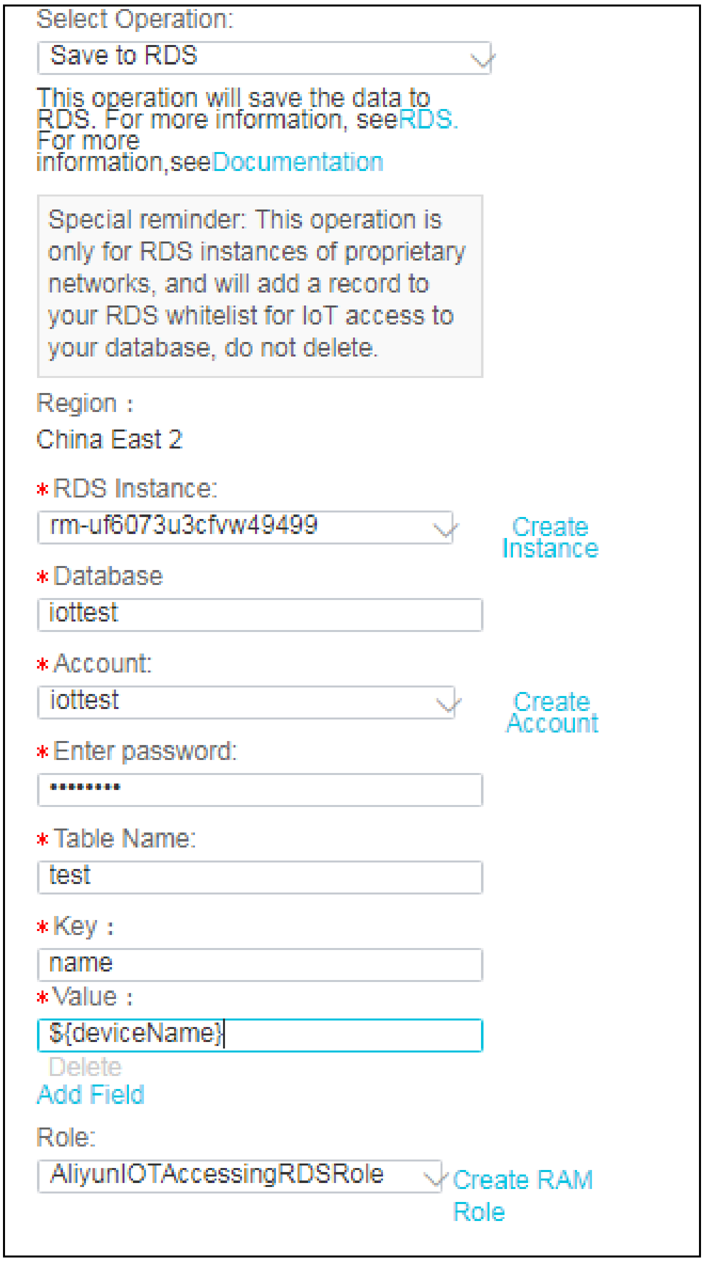
- The ApsaraDB for RDS instances and your IoT Platform service must be in the same region. For example, if your devices are in cn-shanghai region, the data can only be forwarded to RDS instances in the cn-shanghai region.
- Only RDS instances in VPCs are supported.
- Only MySQL instances and SQL Server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

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

*Database: iottest

*Account: iottest

*Enter password: ••••••••

*Table Name: test

*Key : name

*Value : ${deviceName}

Role: AliyunIOTAccessingRDSRole

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.

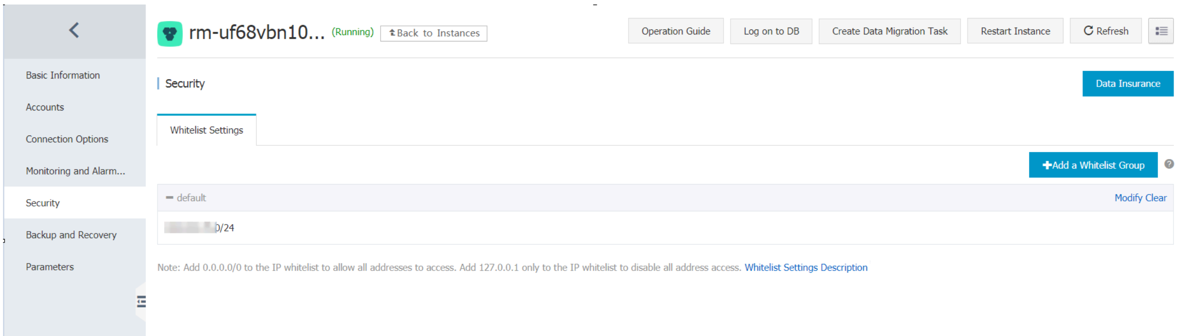
Parameter	Action
Database	Enter the name of the target database. Note: If your database is in the master mode, you need to manually enter the database name.
Account	Enter the account of the RDS database. The account requires the permissions to read and write data to the database. Otherwise, rules engine 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 `${ deviceName }`, 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.



[redacted]0/24

Note: Add 0.0.0.0/0 to the IP whitelist to allow all addresses to access. Add 127.0.0.1 only to the IP whitelist to disable all address access. Whitelist Settings Description

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

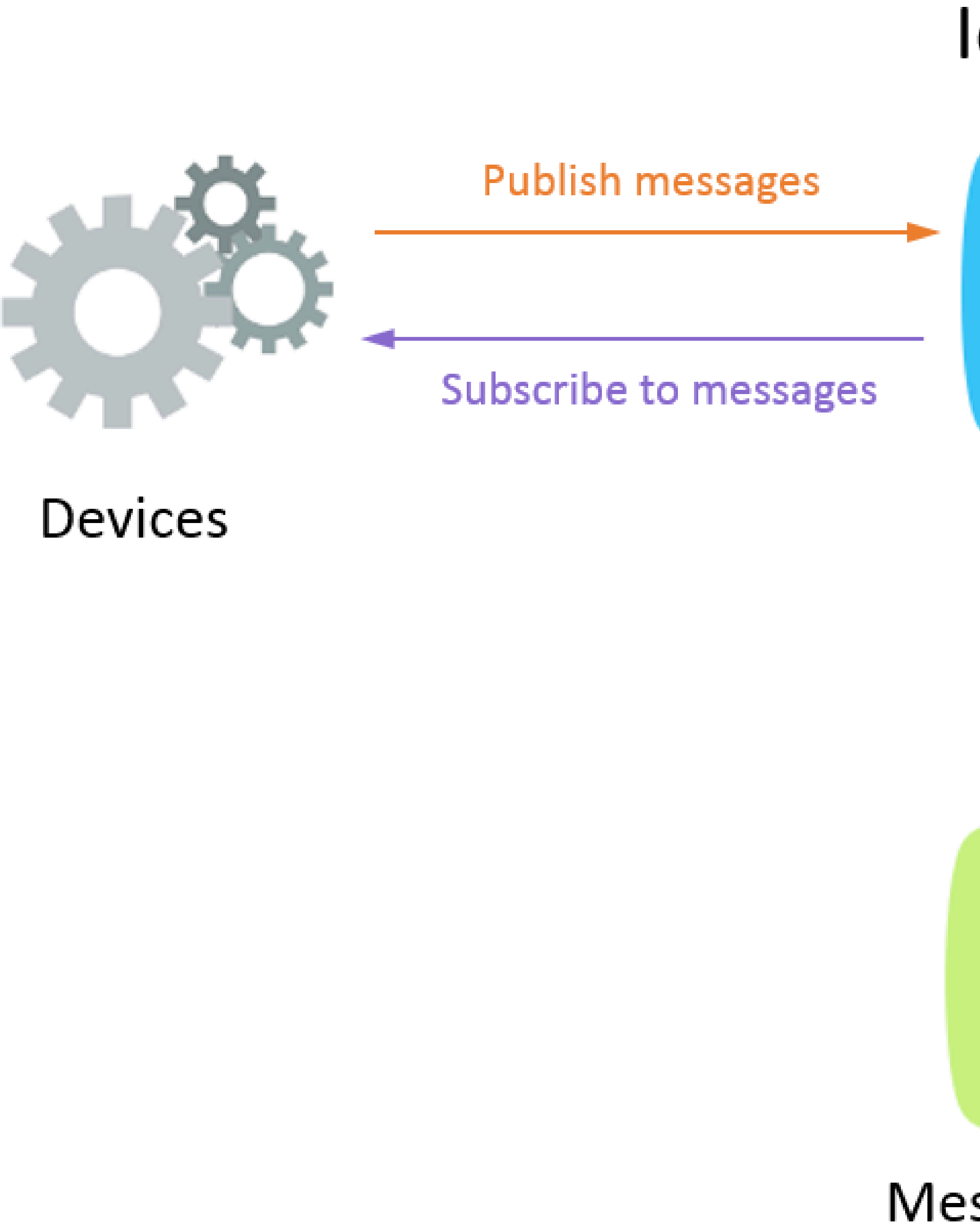
- Devices send data to application servers

  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.


Publish messages
Subscribe to messages
Devices
Io
Mes

Procedure

1. Log on to the *RAM console*, and create a role with the permission to write messages from IoT Platform into MNS.

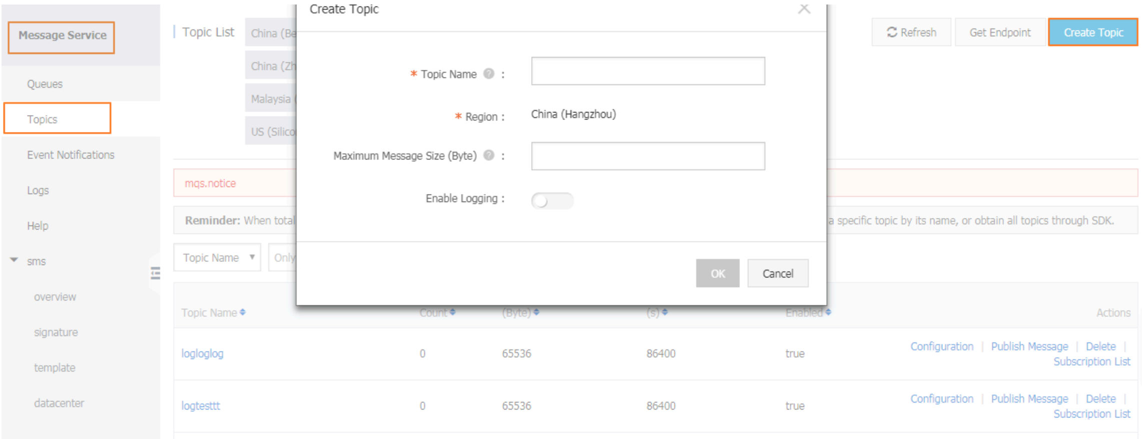
   Then, when you are configuring the data forwarding rule in IoT Platform, you can apply this role to allow IoT Platform to write data into MNS. Without applying such a role, IoT Platform cannot forward data to MNS.

   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.



Create Topic

* Topic Name :

* Region : China (Hangzhou)

Maximum Message Size (Byte) :

Enable Logging :

mqs.notice

Reminder: When total ... a specific topic by its name, or obtain all topics through SDK.

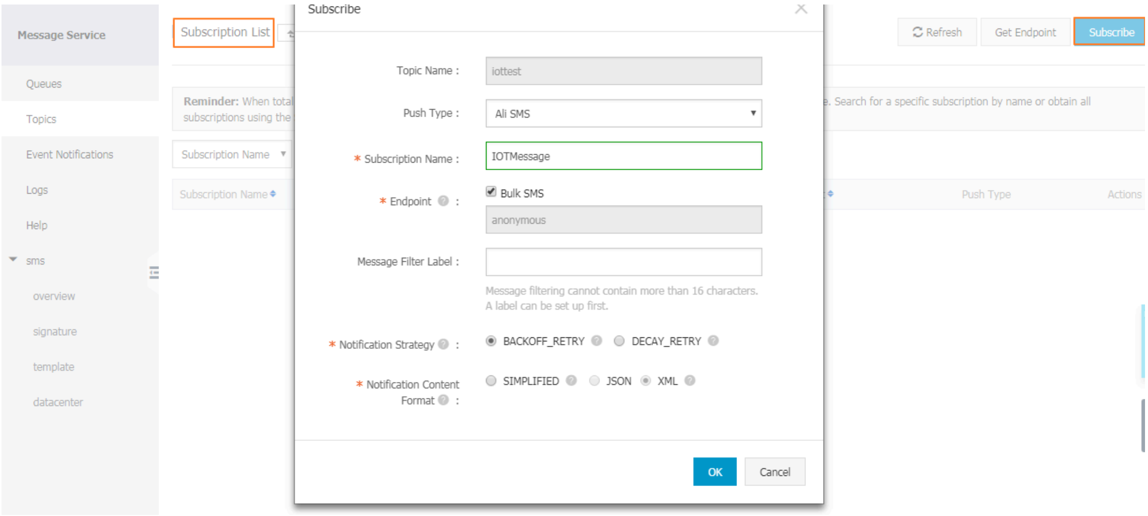
Topic Name	Count	(Byte)	(s)	Enabled	Actions
loglogloq	0	65536	86400	true	Configuration | Publish Message | Delete | Subscription List
logtesttt	0	65536	86400	true	Configuration | Publish Message | Delete | Subscription List

   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.



Subscribe

Topic Name : iottest

Push Type : Ali SMS

* Subscription Name : IOTMessage

* Endpoint : ☑ Bulk SMS
anonymous

Message Filter Label :

Message filtering cannot contain more than 16 characters.
A label can be set up first.

* Notification Strategy : ● BACKOFF_RETRY ○ DECAY_RETRY

* Notification Content Format : ○ SIMPLIFIED ○ JSON ● XML

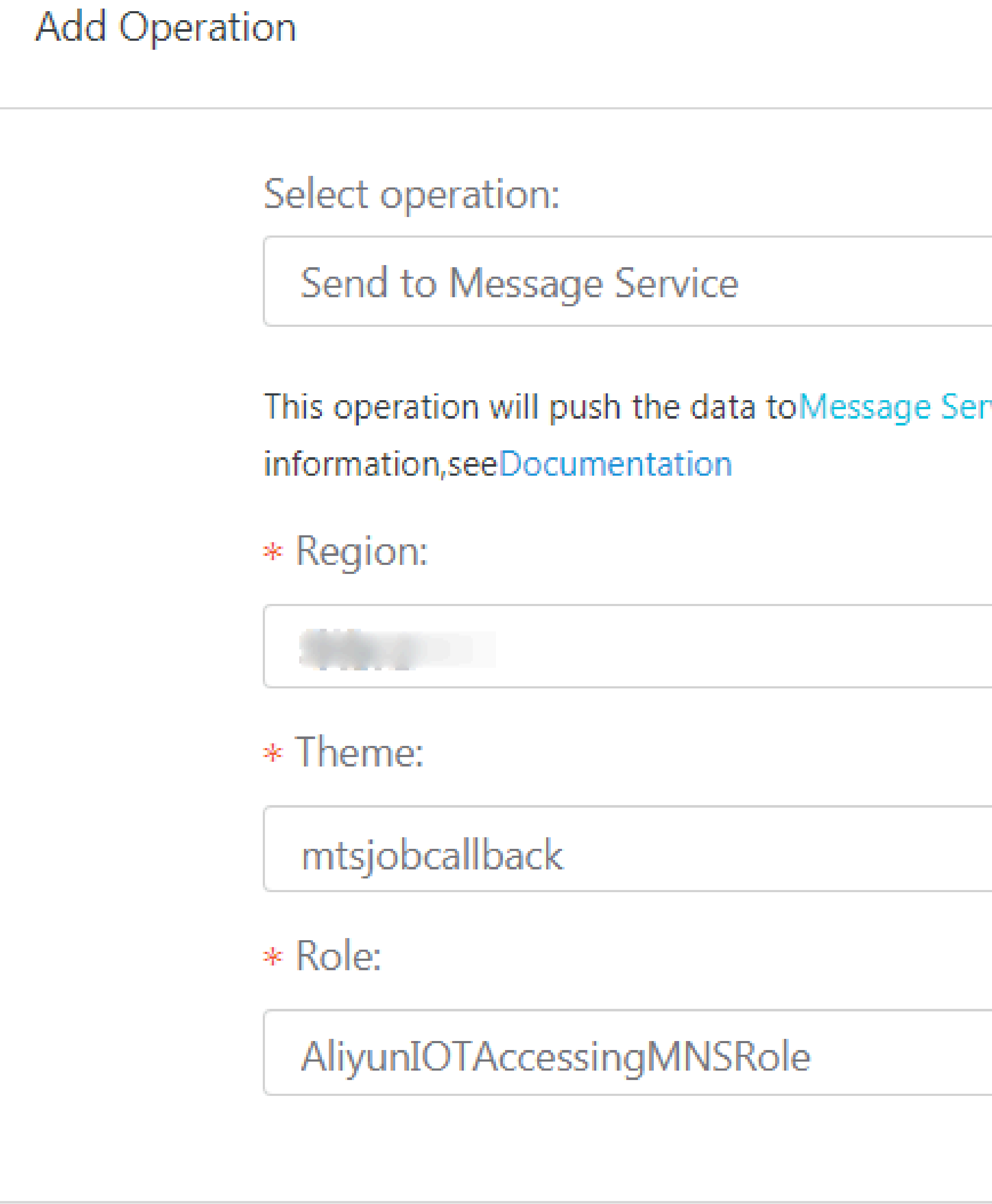
Reminder: When total ... subscriptions using the ... Search for a specific subscription by name or obtain all

      For more information, see the *MNS documentations*.

3. Go to the *IoT Platform console* and, on the Rules page, click Create Ruleand 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 toMessage Ser

information,seeDocumentation

* Region:

* Theme:

mtsjobcallback

* Role:

AliyunIOTAccessingMNSRole

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

   Parameter description:

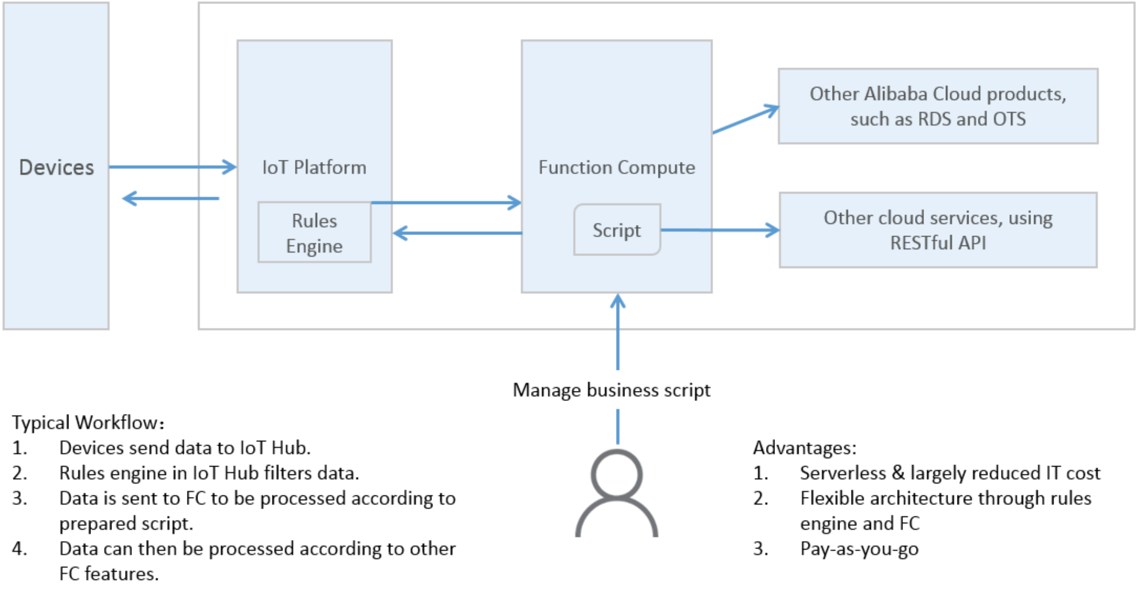
Parameter	Description
Select Operation	Select the Alibaba Cloud product which will be the data forwarding target. Here, select Send to Message Service.
Region	Select the region where the MNS topic is.
Theme	Select the MNS topic that is to receive data from IoT Platform.
Role	The role with the permission 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 Hub to Function Compute (FC).


Devices
IoT Platform
Rules Engine
Function Compute
Script
Other Alibaba Cloud products, such as RDS and OTS
Other cloud services, using RESTful API
Manage business script

Typical Workflow:
1. Devices send data to IoT Hub.
2. Rules engine in IoT Hub filters data.
3. Data is sent to FC to be processed according to prepared script.
4. Data can then be processed according to other FC features.

Advantages:
1. Serverless & largely reduced IT cost
2. Flexible architecture through rules engine and FC
3. Pay-as-you-go

Procedure:

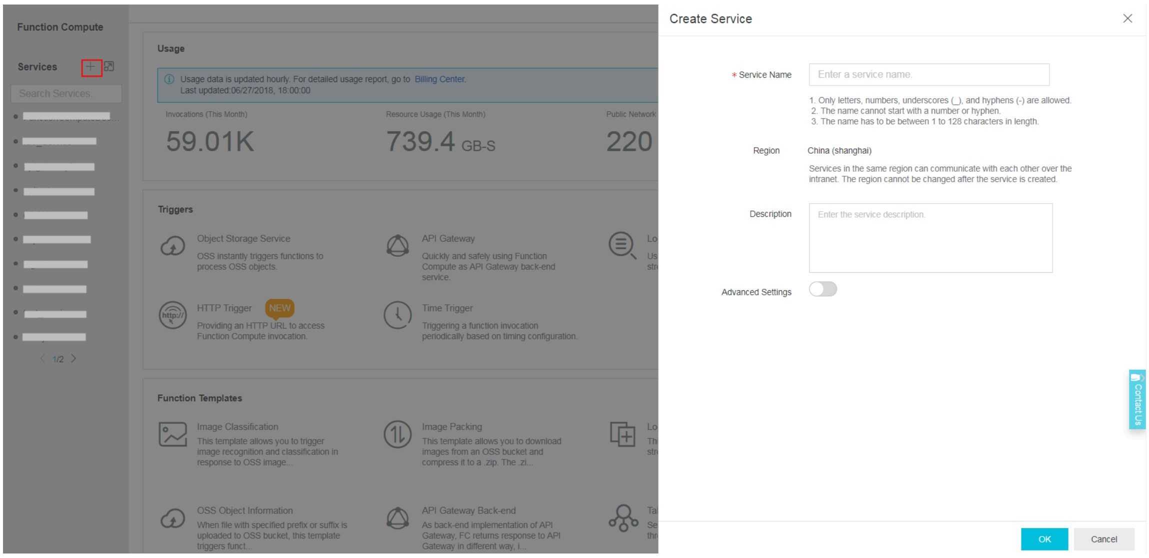
1. On the Function Compute console, create a service and function.
2. Create a rule to send data processed on IoT Platform to FC, and then enable the rule.
3. Send a message to the topic that has rules engine configured.

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.



Usage data is updated hourly. For detailed usage report, go to Billing Center.
Last updated:06/27/2018, 18:00:00

Invocations (This Month) 59.01K
Resource Usage (This Month) 739.4 GB-S
Public Network 220

Object Storage Service: OSS instantly triggers functions to process OSS objects.
API Gateway: Quickly and safely using Function Compute as API Gateway back-end service.
HTTP Trigger NEW: Providing an HTTP URL to access Function Compute invocation.
Time Trigger: Triggering a function invocation periodically based on timing configuration.

Image Classification: This template allows you to trigger image recognition and classification in response to OSS image...
Image Packing: This template allows you to download images from an OSS bucket and compress it to a .zip. The .zi...
OSS Object Information: When file with specified prefix or suffix is uploaded to OSS bucket, this template triggers funct...
API Gateway Back-end: As back-end implementation of API Gateway, FC returns response to API Gateway in different way, i...

Create Service

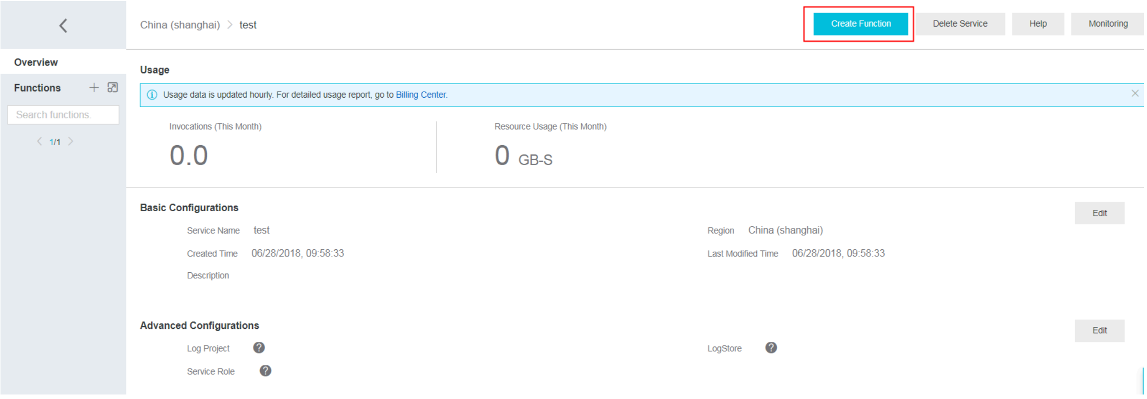
* Service Name: Enter a service name.
1. Only letters, numbers, underscores (_), and hyphens (-) are allowed.
2. The name cannot start with a number or hyphen.
3. The name has to be between 1 to 128 characters in length.

Region: China (shanghai)
Services in the same region can communicate with each other over the intranet. The region cannot be changed after the service is created.

Description: Enter the service description.

Advanced Settings

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



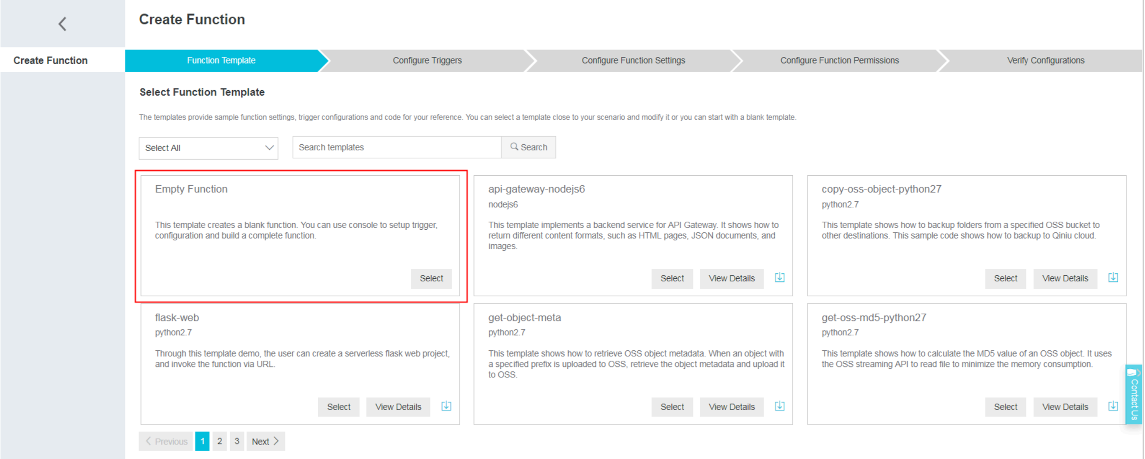
Usage
Usage data is updated hourly. For detailed usage report, go to Billing Center.

Invocations (This Month) 0.0
Resource Usage (This Month) 0 GB-S

Basic Configurations
Service Name: test
Region: China (shanghai)
Created Time: 06/28/2018, 09:58:33
Last Modified Time: 06/28/2018, 09:58:33
Description

Advanced Configurations
Log Project
LogStore
Service Role

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



Select Function Template
The templates provide sample function settings, trigger configurations and code for your reference. You can select a template close to your scenario and modify it or you can start with a blank template.

Empty Function: This template creates a blank function. You can use console to setup trigger, configuration and build a complete function.

api-gateway-nodejs6 (nodejs6): This template implements a backend service for API Gateway. It shows how to return different content formats, such as HTML pages, JSON documents, and images.

copy-oss-object-python27 (python2.7): This template shows how to backup folders from a specified OSS bucket to other destinations. This sample code shows how to backup to Qiniu cloud.

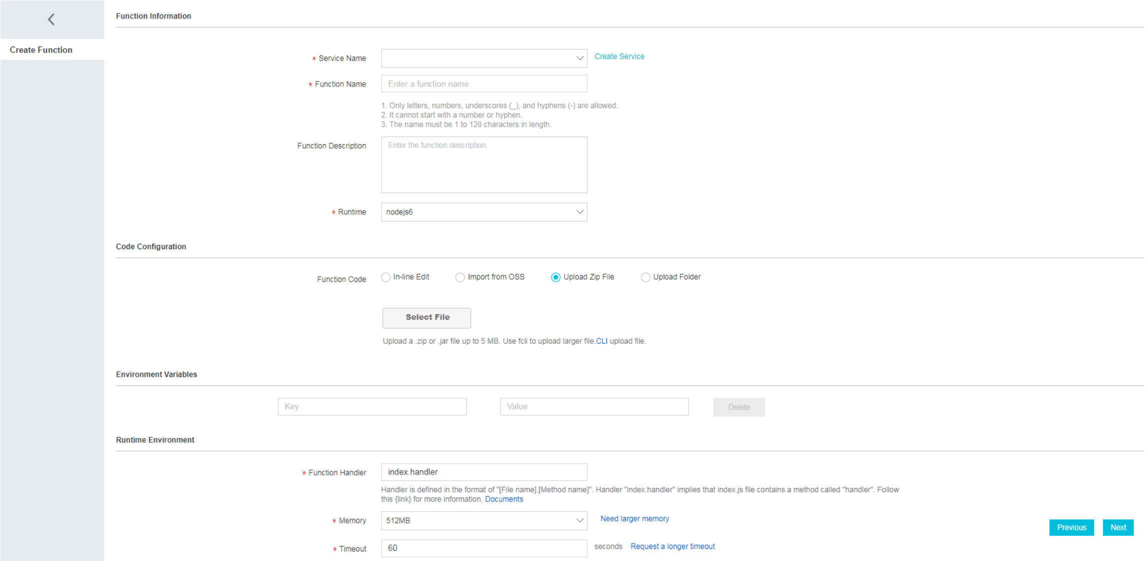
flask-web (python2.7): Through this template demo, the user can create a serverless flask web project, and invoke the function via URL.

get-object-meta (python2.7): This template shows how to retrieve OSS object metadata. When an object with a specified prefix is uploaded to OSS, retrieve the object metadata and upload it to OSS.

get-oss-md5-python27 (python2.7): This template shows how to calculate the MD5 value of an OSS object. It uses the OSS streaming API to read file to minimize the memory consumption.

d. Set parameters for the function.

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



Function Information

* Service Name:

* Function Name: Enter a function name

1. Only letters, numbers, underscores (_), and hyphens (-) are allowed.
2. It cannot start with a number or hyphen.
3. The name must be 1 to 128 characters in length.

Function Description: Enter the function description.

* Runtime: nodejs6

Code Configuration

Function Code: ○ In-line Edit ○ Import from OSS ● Upload Zip File ○ Upload Folder

Upload a .zip or .jar file up to 5 MB. Use fcli to upload larger file.CLI upload file.

Environment Variables

Key Value

Runtime Environment

* Function Handler: index.handler

Handler is defined in the format of "[File name].[Method name]". Handler "index.handler" implies that index.js file contains a method called "handler". Follow this {link} for more information. Documents

* Memory: 512MB

* Timeout: 60 seconds

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.

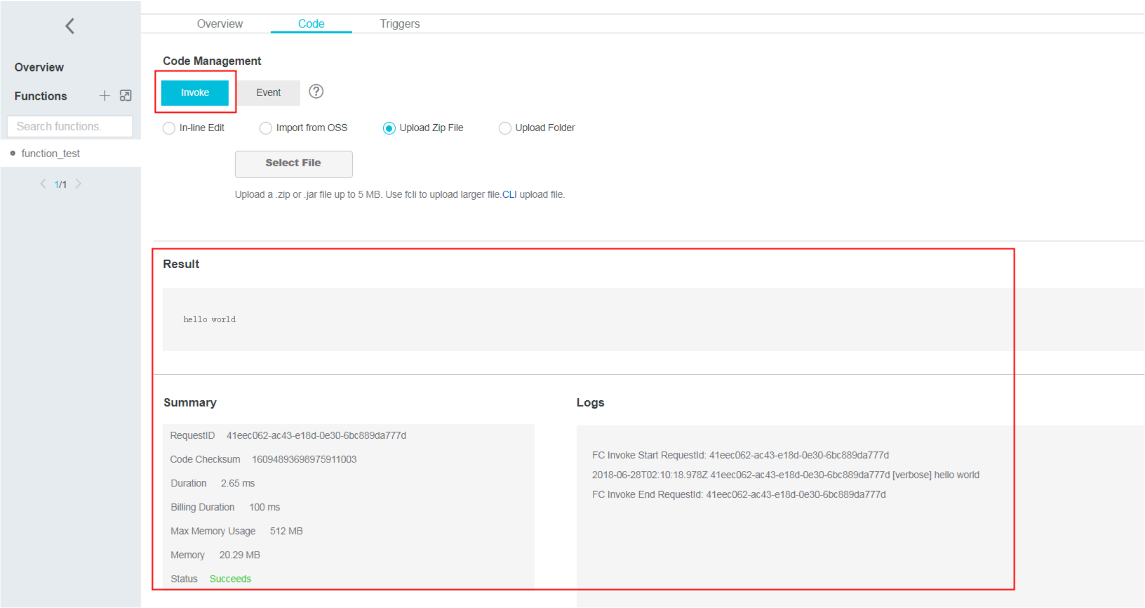
Code Configuration: Upload your code.

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

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.



○ In-line Edit ○ Import from OSS ● Upload Zip File ○ Upload Folder

Upload a .zip or .jar file up to 5 MB. Use fcli to upload larger file.CLI upload file.

Result

```
hello world
```

Summary

RequestID	41eec062-ac43-e18d-0e30-6bc889da777d
Code Checksum	16094893698975911003
Duration	2.65 ms
Billing Duration	100 ms
Max Memory Usage	512 MB
Memory	20.29 MB
Status	Succeeds

Logs

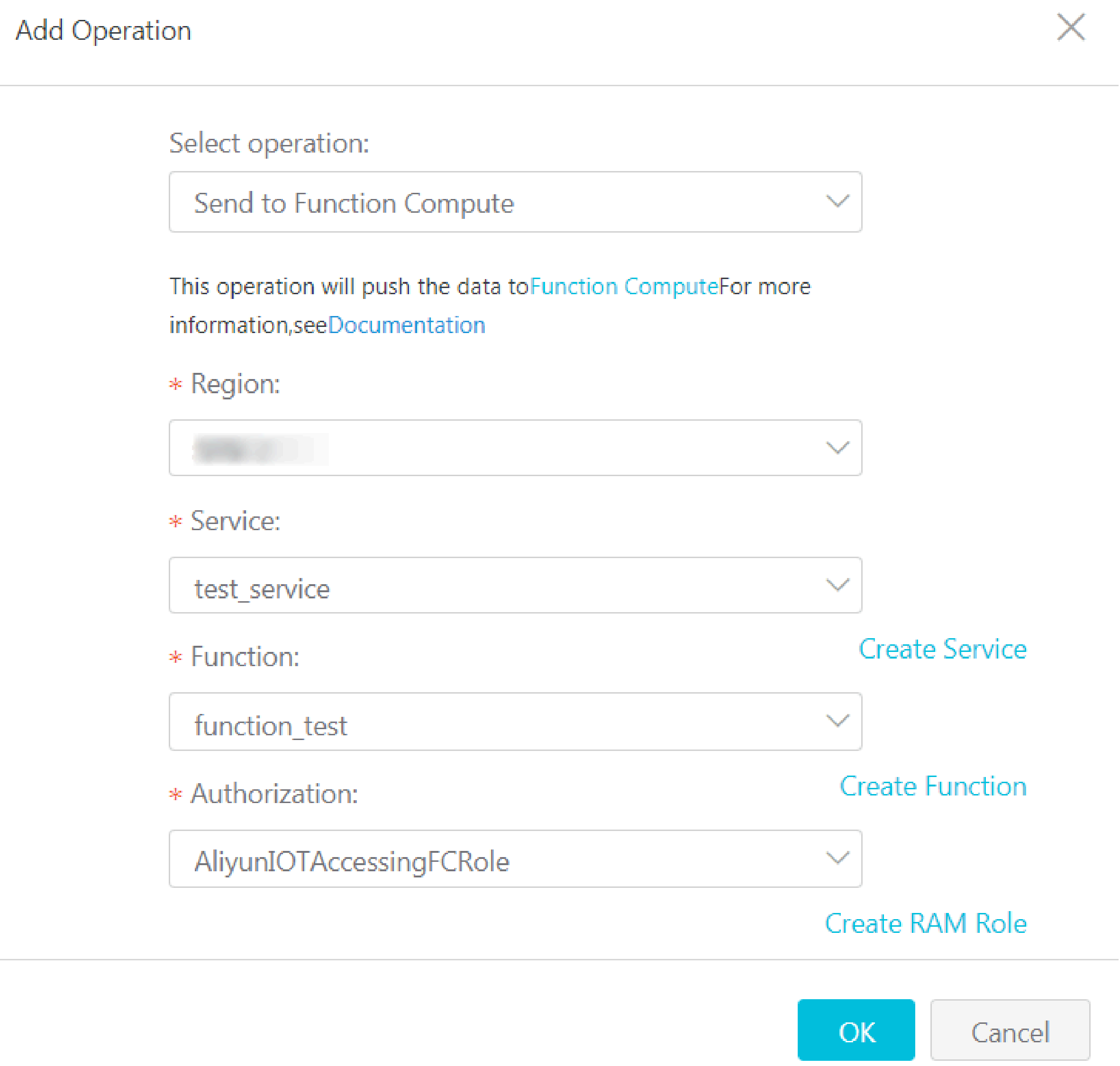
```
FC Invoke Start RequestId: 41eec062-ac43-e18d-0e30-6bc889da777d
2018-06-28T02:10:18.978Z 41eec062-ac43-e18d-0e30-6bc889da777d [verbose] hello world
FC Invoke End RequestId: 41eec062-ac43-e18d-0e30-6bc889da777d
```

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 toFunction ComputeFor more information,seeDocumentation

* Region:

* Service: test_service

* Function: function_test

* Authorization: AliyunIOTAccessingFCRole

- 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:

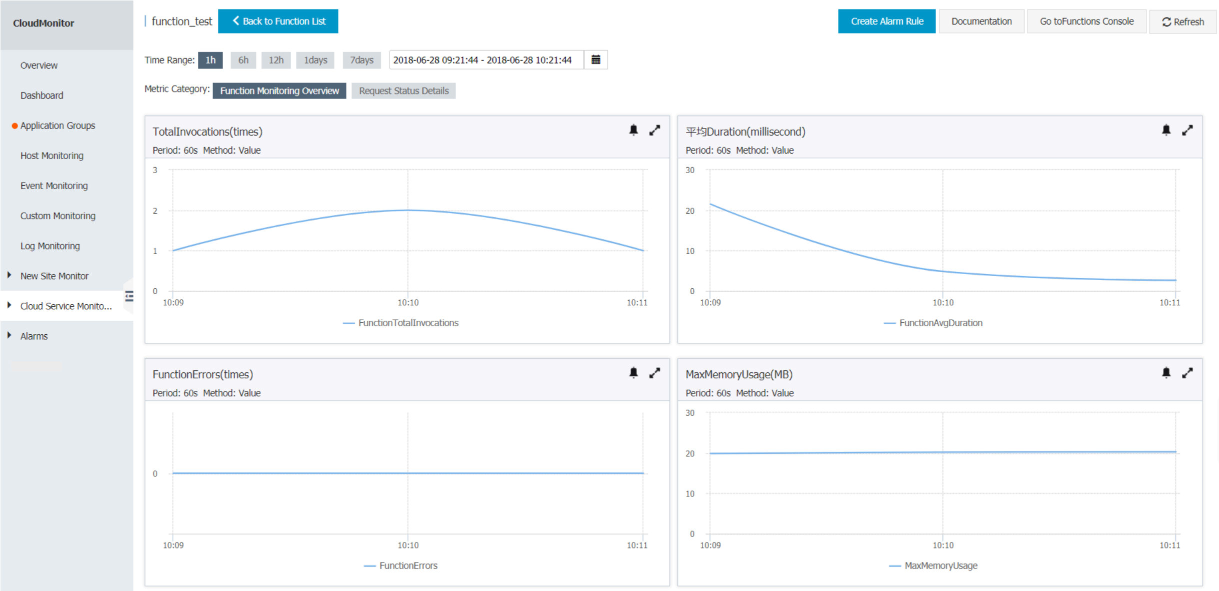
Data forwarding to FC is supported in regions including China (Shanghai), Singapore, and Japan (Tokyo).

- 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 Hub 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 Online debug

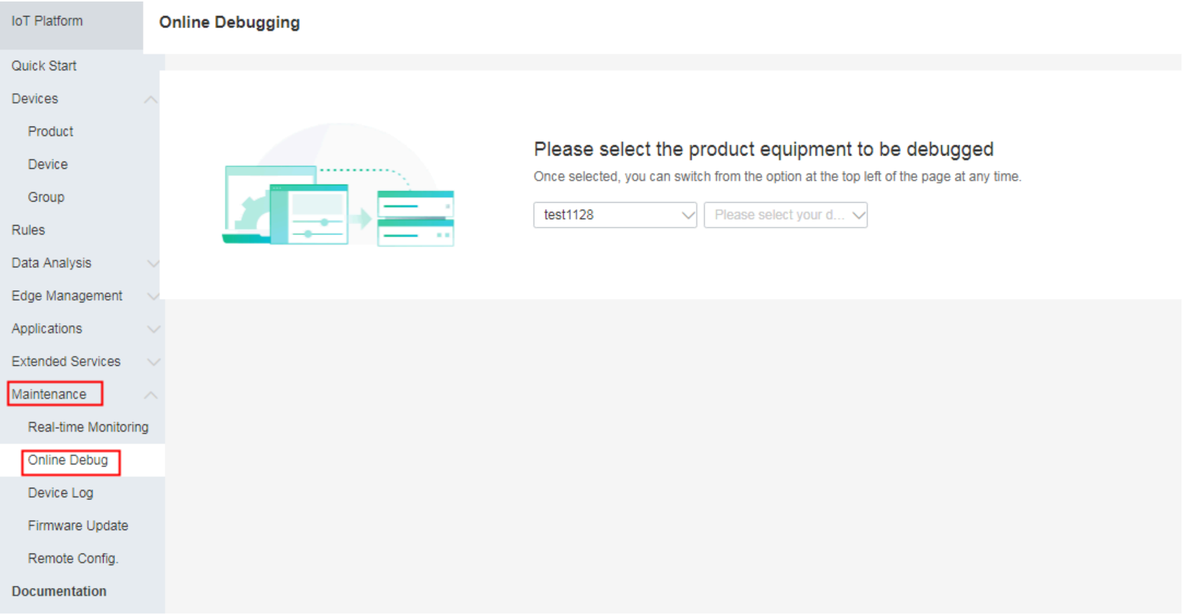
3.1.1 Online debugging

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

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.

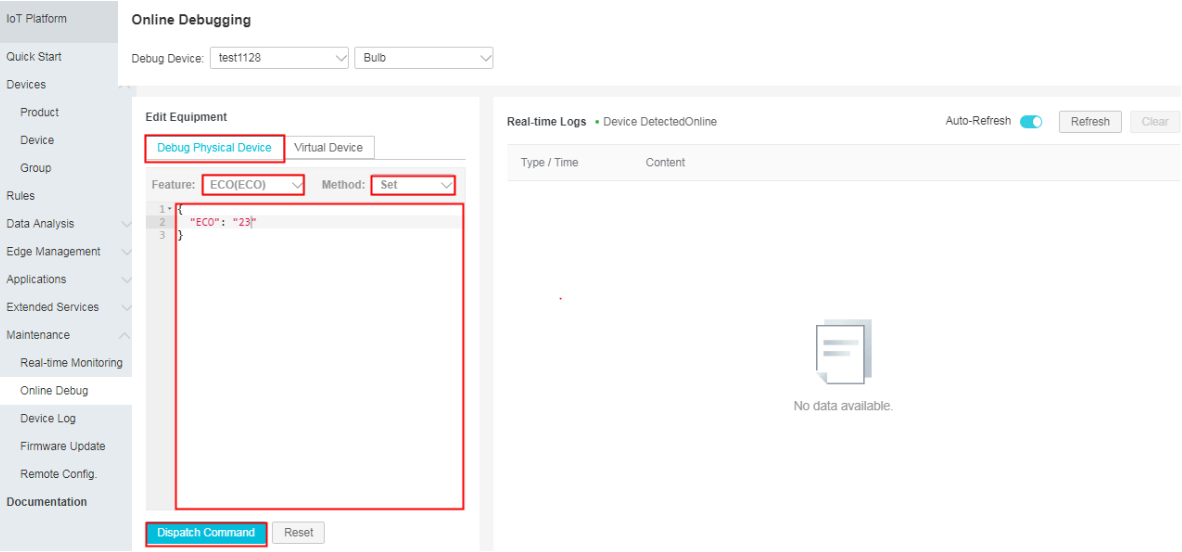
   After you select a device, you are automatically directed to the debugging page.



3. Select Debug Physical Device.
4. Select the feature that you want to test.

> Note:
>
> - If you select a property, you need to select an operation method from Set and Get.

- If you select an event, select Get as the operation method.



Debug Device: test1128 Bulb

Feature: ECO(ECO) Method: Set

```
{
  "ECO": "23"
}
```

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 `{"YourServiceInputParam": Value}`, and then click Dispatch Command. You can then see the operation result from the device log.
   - Get an event: Click Dispatch Command. Then, the latest event information reported by the device is displayed in the box.

3.1.2 Debug applications using virtual devices

IoT Platform provides virtual devices to help developers debug applications. Currently, only IoT Platform Pro Edition supports the online debugging feature.

Context

A typical IoT development process is as follows: a device client 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.

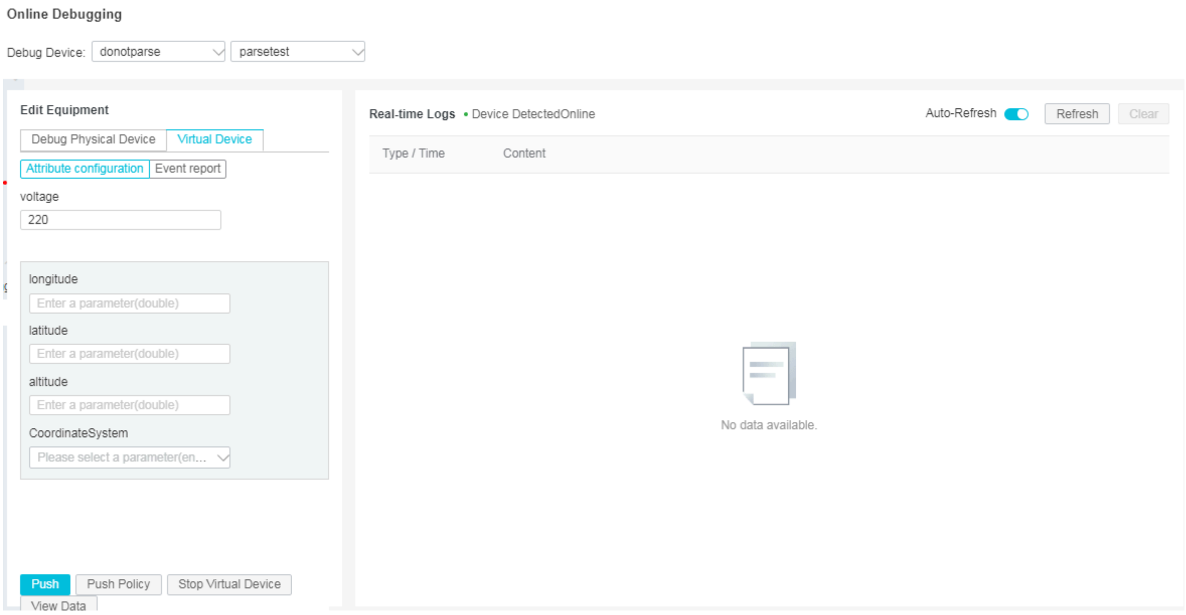
   After you select a device, you are automatically directed to the debugging page.
4. Choose Virtual Device > Start Virtual Device.

   > Note:
   >
   > If the physical device is active or disabled, you cannot start the corresponding virtual device.
5. Set the content for the simulated push.

   > Note:
   >
   > - You can push properties and events. If you have not defined any properties or events for this product, click Editing model, and then define properties and events on the Define Feature page of the product.

- 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

voltage: 220

longitude: Enter a parameter(double)

latitude: Enter a parameter(double)

altitude: Enter a parameter(double)

CoordinateSystem: Please select a parameter(en...

Real-time Logs • Device DetectedOnline

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.

3.2 Device log

IoT Platform provides device logs that you can use to monitor your devices. On the Device Log page of the IoT Platform console, you can search for specific device logs to quickly troubleshoot any errors. This topic describes the device log querying methods, log types, and reasons for errors found in logs.

Query device logs

Device logs can be of the following four types:

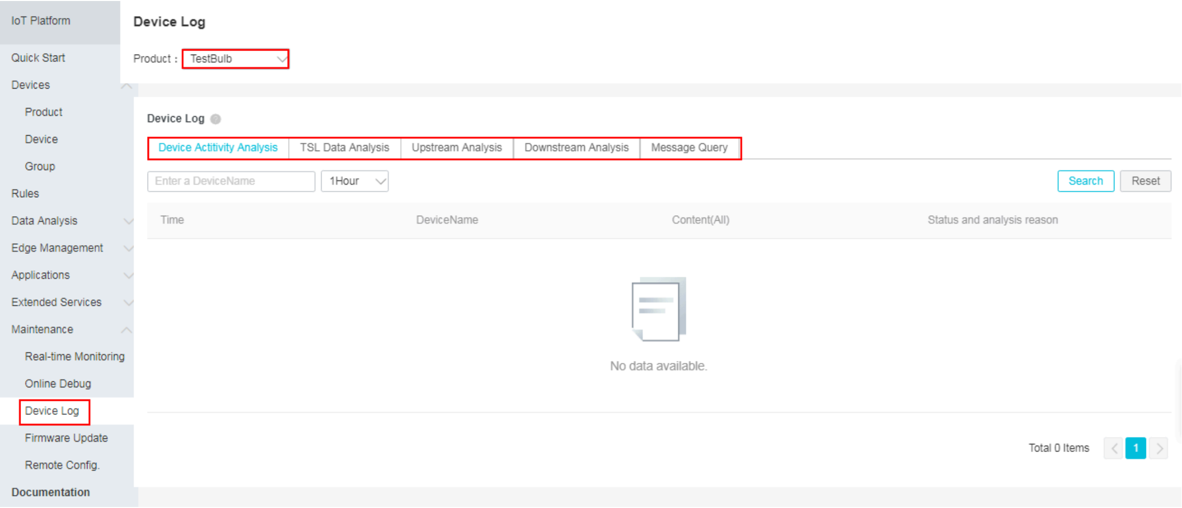
- *Device activity analysis logs*
- *Upstream data analysis logs*
- *Downstream data analysis logs*
- *TSL data analysis logs*

Note that Basic Edition products only support the following three types of logs: device activity analysis logs, upstream data analysis logs, and downstream data analysis logs . Pro Edition products support all four types of logs.

Query device logs:

1. In the left-side navigation pane of the IoT Platform console, click Maintenance > Device Log.

2. Enter the target items to filter, such as product name, log type, device name, and time range, and then click Search.



Filters for device logs:

Filter	Description
DeviceName	The device name, which is a unique identifier of a device in a product. You can query logs of a device by using the device name as the filter.
MessageID	The message ID, which is the unique identifier of a message in IoT Platform. You can enter a message ID to search for the corresponding message forwarding process.
Status	The logs that display operation results. The value can be either successful or failed. Options: · All · Successful · Failed
Time range	A specific time range you can specify for querying logs in that period.

Note:

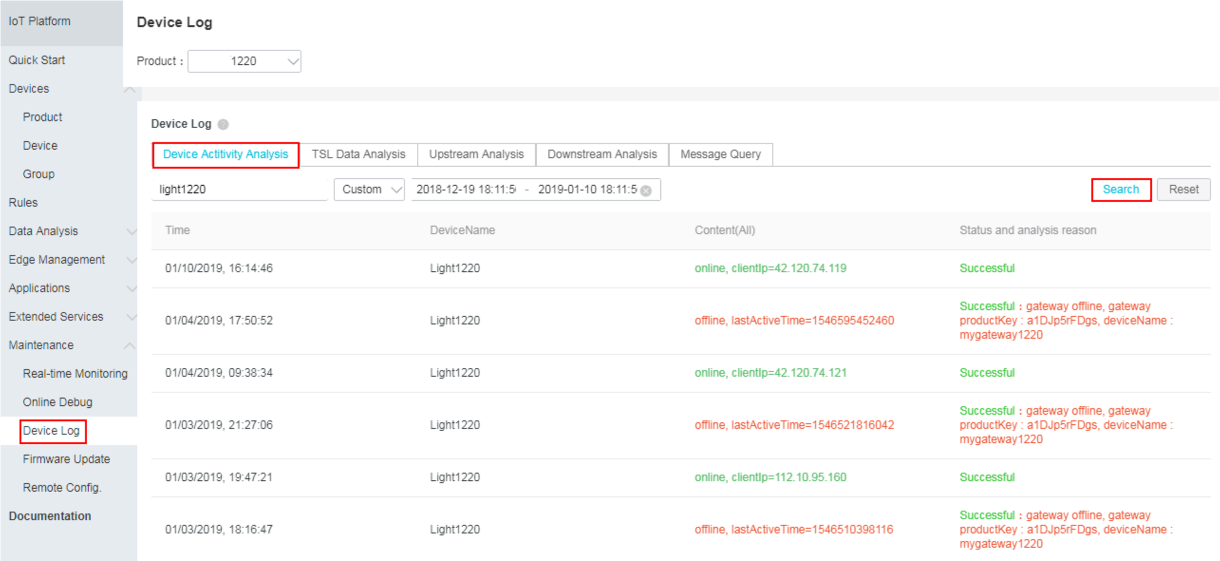
- In the following sections, curly braces `{}` in log content represent variables. In actual log content, the real variable is displayed.
- Log content is in English.

- When error logs are displayed, all errors (except for `system errors`) are caused by improper operations or violations of product restrictions. Such errors need to be rectified carefully.

Device activity analysis logs

Device activity analysis logs include logs of devices connecting to IoT Platform (online) and logs of devices disconnecting from IoT Platform (offline).

Device activity analysis logs can be queried by device names and time ranges as shown in the following figure:



Product : 1220

light1220 | Custom | 2018-12-19 18:11:5 - 2019-01-10 18:11:5

Time	DeviceName	Content(All)	Status and analysis reason
01/10/2019, 16:14:46	Light1220	online, clientIp=42.120.74.119	Successful
01/04/2019, 17:50:52	Light1220	offline, lastActiveTime=1546595452460	Successful : gateway offline, gateway productKey : a1DJp5rFDgs, deviceName : mygateway1220
01/04/2019, 09:38:34	Light1220	online, clientIp=42.120.74.121	Successful
01/03/2019, 21:27:06	Light1220	offline, lastActiveTime=1546521816042	Successful : gateway offline, gateway productKey : a1DJp5rFDgs, deviceName : mygateway1220
01/03/2019, 19:47:21	Light1220	online, clientIp=112.10.95.160	Successful
01/03/2019, 18:16:47	Light1220	offline, lastActiveTime=1546510398116	Successful : gateway offline, gateway productKey : a1DJp5rFDgs, deviceName : mygateway1220

Device connection failures

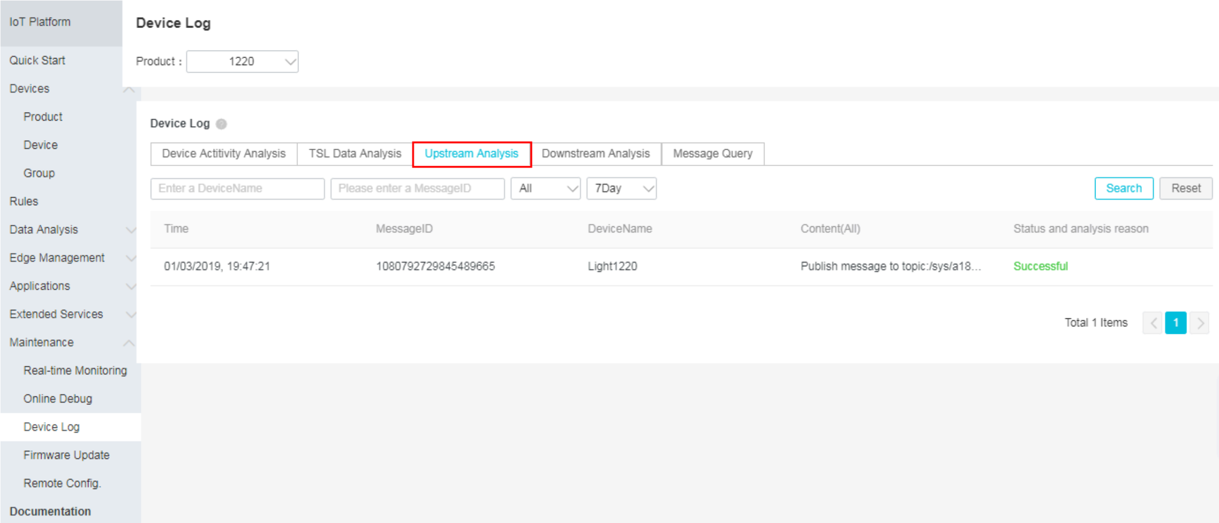
Message	Description
Kicked by the same device	Another device installed with the same device certificate as this device has connected to IoT Platform, and has brought this device offline.
Connection reset by peer	The TCP connection has been reset by the peer.
Connection occurs exception	A connection exception has occurred, and the IoT Platform server has closed the connection.
Device disconnect	The device sent a disconnection request.
Keepalive timeout	No package was received in a Keep Alive interval, and the IoT Platform server has closed the connection.

Message	Description
gateway offline	The gateway device of the sub-device is offline.

Upstream data analysis logs

Upstream data analysis logs indicate logs of the following processes: devices sending messages to topics, messages being forwarded to the rules engine, and the rules engine forwarding the messages to a target topic or other Alibaba Cloud services.

You can query the upstream data analysis logs by device names, message IDs, status, or time ranges, as shown in the following figure:



Product : 1220

All · 7Day

Time	MessageID	DeviceName	Content(All)	Status and analysis reason
01/03/2019, 19:47:21	1080792729845489665	Light1220	Publish message to topic:/sys/a18...	Successful

Error log description

Note:
Error logs include the log content, error messages, and error message descriptions.

Content	Error message	Description
Device publish message to topic:{},QoS={},protocolMessageId:{}	Rate limit:{maxQps}, current qps:{}	The device publishes messages in a frequency that exceeds the upper limit.
	No authorization	Not authorized.
	System error	A system error occurred.

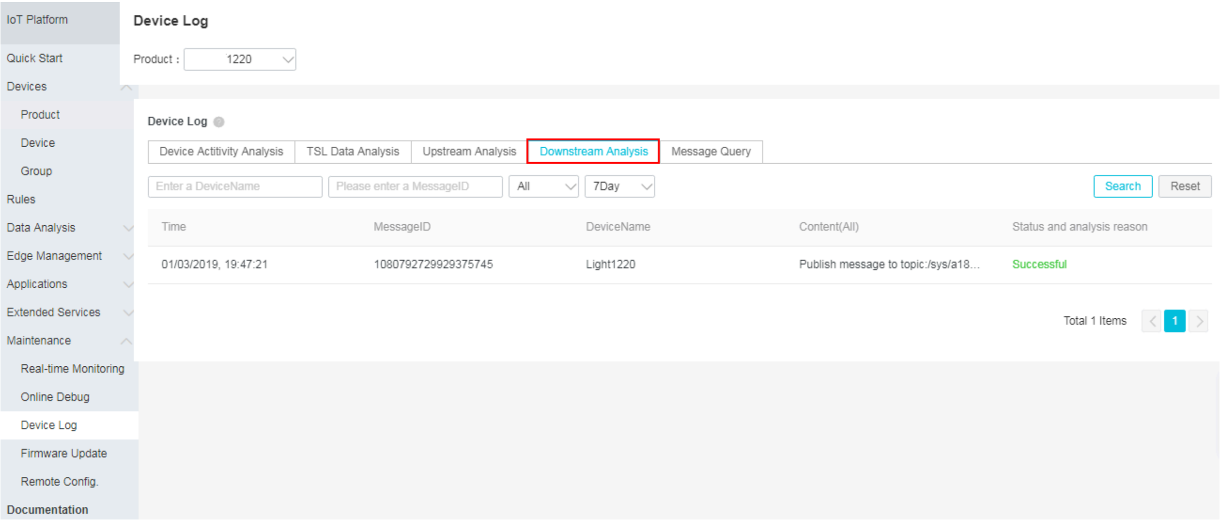
Content	Error message	Description
	Bad Request	A parameter error has occurred. A parameter or parameters such as topic, payload, token, or option for CoAP communication, are incorrect or are missing.
send message to RuleEngine, topic:{} protocolMessageId:{}	{eg, too many requests}	Other failure reasons, for example, IoT Platform sends too many requests to the rules engine.
	System error	A system error occurred.
Transmit data to DataHub, project:{},topic:{},from IoT topic:{}	DataHub Schema:{} is invalid!	Data type mismatch.
	DataHub IllegalArgumentException:{}	Invalid DataHub parameters.
	Write record to DataHub occurs error! errors:[code:{},message:{}]	An error occurred when data was written to DataHub.
	Datahub ServiceException:{}	DataHub service exception.
	System error	A system error occurred.
Transmit data to MNS, queue:{},theme:{},from IoT topic:{}	MNS IllegalArgumentException:{}	Message Service parameter exception.
	MNS ServiceException:{}	Message Service exception.
	MNS ClientException:{}	Message Service client exception.
	System error	A system error occurred.
Transmit data to MQ,topic:{},from IoT topic:{}	MQ IllegalArgumentException:{}	Message Queue parameter exception.
	MQ ClientException:{}	Message Queue client exception.
	System error	A system error occurred.

Content	Error message	Description
Transmit data to TableStore,instance:{}, tableName:{},from IoT topic:{}	TableStore IllegalArgumentException:{}	Table Store parameter exception.
	TableStore ServiceException:{}	Table Store service exception.
	TableStore ClientException:{}	Table Store client exception.
	System error	A system error occurred.
Transmit data to RDS, instance:{},databaseName:{},tableName:{},from IoT topic:{}	RDS IllegalArgumentException:{}	ApsaraDB for RDS parameter exception
	RDS CannotGetConnectionException:{}	Failed to connect to ApsaraDB for RDS.
	RDS SQLException:{}	SQL statement for ApsaraDB for RDS is invalid.
	System error	A system error occurred.
Republish topic, from topic:{} to target topic:{}	System error	A system error occurred.
RuleEngine receive message from IoT topic:{}	Rate limit:{maxQps}, current qps:{}	The frequency exceeds the upper limit.
	System error	A system error occurred.
Check payload, payload:{}	Payload is not json	The payload is not in JSON format.

Downstream data analysis logs

Downstream data analysis logs are logs about messages sent from IoT Platform to devices.

You can filter logs by device names, message IDs, status, and time ranges, as shown in the following figure.



Device Log

Product : 1220

Device Log

Enter a DeviceName | Please enter a MessageID | All | 7Day

Time	MessageID	DeviceName	Content(All)	Status and analysis reason
01/03/2019, 19:47:21	1080792729929375745	Light1220	Publish message to topic:/sys/a18...	Successful

Error log description

Note:

The logs include the log contents, error messages, and error message descriptions.

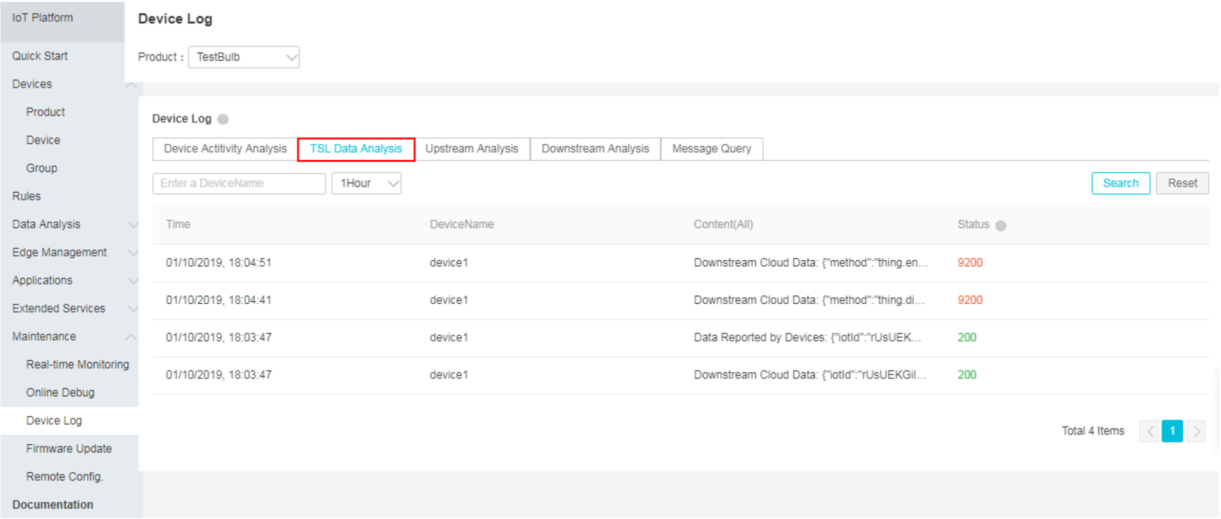
Content	Error message	Description
Publish message to topic:{},protocolMessageId:{}	No authorization	Not authorized.
Publish message to device, QoS={}	IoT Hub cannot publish messages	If the IoT Platform server does not receive PUBACK from the device , it continues to send messages. When the number of messages reaches 50, the throttling policy is triggered. Consequently, IoT Platform cannot send new messages to the device.
	Device cannot receive messages	The device client failed to receive messages. This error may be caused by slow network transmission speeds, or because the device client cannot handle any more messages .
	Rate limit:{maxQps}, current qps:{}	The frequency exceeds the upper limit.

Content	Error message	Description
Publish RRPC message to device	IoT hub cannot publish messages	The device did not respond to the server, so the server continued to send messages until it reached the frequency limit. Consequently, the server cannot send new messages .
	Response timeout	The device has not responded to the server within the specified timeout period.
	System error	A system error occurred.
Rrpc finished	{e.g rrpcCode}	Error messages such as UNKNOW, TIMEOUT, OFFLINE and HALFCONN are displayed.
Publish offline message to device	Device cannot receive messages	The device cannot receive messages from IoT Platform. The reason may be that the network condition is not stable, or the device client cannot handle any more messages .

TSL data analysis logs

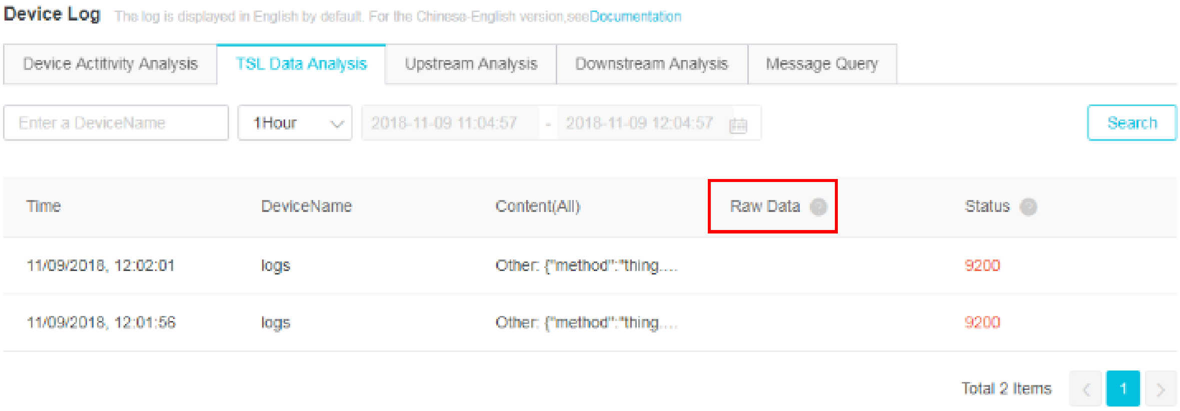
TSL data analysis logs include logs of devices reporting properties and events, property settings, service callings, and the replies to property and service calls.

You can filter logs by device names and time ranges. If the device data type is Alink JSON, the page is displayed as shown in the following figure.



Time	DeviceName	Content(All)	Status
01/10/2019, 18:04:51	device1	Downstream Cloud Data: {"method":"thing.en...	9200
01/10/2019, 18:04:41	device1	Downstream Cloud Data: {"method":"thing.di...	9200
01/10/2019, 18:03:47	device1	Data Reported by Devices: {"iotId":"rUsUEK...	200
01/10/2019, 18:03:47	device1	Downstream Cloud Data: {"iotId":"rUsUEKGil...	200

If the device data type is Do not parse/Custom (passthrough), in addition to the log content, the hexadecimal raw data are also displayed. as shown in the following figure:



Time	DeviceName	Content(All)	Raw Data	Status
11/09/2018, 12:02:01	logs	Other: {"method":"thing....		9200
11/09/2018, 12:01:56	logs	Other: {"method":"thing....		9200

Table 3-1: Log description

Parameter	Description
id	The message ID, which is the unique identifier of the message.
params	The request parameters.
Code	The result code.
method	The request method.
type	The type of message, which can be upstream or downstream.
scriptData	When the data type is Do not parse/Custom, the original data and parsed data are displayed.
downOriginalData	When the data type is Do not parse/Custom, the original downstream data to be parsed is displayed.

Parameter	Description
downTransformedData	When the data type is Do not parse/Custom, the parsed downstream data is displayed.
upOriginalData	When the data type is Do not parse/Custom, the original upstream data to be parsed is displayed.
upTransformedData	When the data type is Do not parse/Custom, the parsed upstream data is displayed.

Error logs of service callings and property settings

When you call a service on the IoT Platform, the service parameters will be verified according to the definitions of the service in the TSL of the product.

Error code	Description	Cause	Troubleshooting method
9201	The device is offline.	When the device is offline, this error is reported.	Check the device status in the IoT Platform console.
9200	The device is not activated yet.	The device has not been activated. When a new device connects to and reports data to IoT Platform, it is activated in IoT Platform.	Check the status of the device in the IoT Platform console.
6208	The device has been disabled.	The device has been disabled. You cannot call services of, or set properties for, a disabled device.	Check the status of the device 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 is verifying the parameters.	The specified identifier of service is not found in the TSL.	See the TSL of the product to which the device belongs in the IoT Platform console, and verify the identifier of the service.

Error code	Description	Cause	Troubleshooting method
6206	Failed to query the definition of the service.	The service is not found.	See the TSL of the product to which the device belongs, and check the definition of the service . Make sure that the definition of the service is the same as that in the TSL .
6200	Data parsing script is not found.	If the data type of the device is Do not parse/Custom, when you call a service, the data will be parsed by the script that you have defined. If you have not defined a parsing script for the product, this error code is displayed.	Go to the product details page in the IoT Platform console to verify whether the parsing script has been submitted. If the parsing script is ready, resubmit it and then try the call again.
6201	The parsing result is empty.	The parsing script runs normally, 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 and troubleshoot the cause.

Error code	Description	Cause	Troubleshooting method
6207	The data format is incorrect.	This error may occur when devices report data to IoT Platform or you call services using the synchronous method. When you call services in the synchronous method, this error may be caused because: · The format of the data returned by the device is incorrect. · The format of the parsed result for Do not parse/Custom data is incorrect. · The format of the input parameters is incorrect.	For data format in calling services, see *API documentations* and the TSL of the product. For the data format of Alink JSON, see *Alink protocol*.
System exception codes			
5159	Failed to obtain the property information from the TSL.	A system exception occurred.	Open a ticket in the console and submit information about the error in the ticket for further consultation.
5160	Failed to obtain the event information from the TSL.		
5161	Failed to obtain the event information from the TSL.		

Error code	Description	Cause	Troubleshooting method
6661	Failed to query the tenant information.		
6205	An error occurred when calling the service.		

Error logs for reporting properties and events

When a device is reporting a property or an event, the parameters of the property or event that you input will be verified based on the TSL of the device.

Error code	Description	Cause	Troubleshooting method
6106	The number of properties reported exceeds the upper limit.	A device can only report up to 200 properties at a time.	View the logs of property reports and check the number of properties of the device on the IoT Platform. Or, view the local logs for the property number of the device.
6300	The method parameter is not found when the system is verifying the parameters.	The method parameter , which is required by the Alink protocol, is not found in the Alink (standard) format data reported by the device or in the parsed data of the passthrough data reported by the device.	View the logs of property reports for the reported data on the IoT Platform. Or, view the local logs for the reported data.

Error code	Description	Cause	Troubleshooting method
6320	The property information is not found when the system is verifying the property parameters.	The specified property is not found in the TSL of the device.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs to determine whether the specified property has already been defined. If the property has not been defined in the TSL, define it.
6450	The method parameter in Alink data is not found .	The parameter of method is not found in the Alink data reported by a device or in the parsed result of Do not parse/Custom data.	View the logs of device property reporting and check whether the parameter of method has been reported. Or you can check the local device logs for the information.
6207	The data format is incorrect.	This error occurs when you call a service in the synchronous method or devices report data to IoT Platform. When devices report data to IoT Platform, this error may occur because the Alink data reported by devices is not in JSON format, or the parsed result of Do not parse/Custom is not in JSON format.	For data format, see *Alink protocol documentations*
System exceptions			
6452	Traffic limiting	Traffic throttling has been triggered because too many requests have been submitted.	Open a ticket in the console for troubleshooting.

Error code	Description	Cause	Troubleshooting method
6760	The storage quota of the tenant is exceeded.	A system exception occurred.	Open a ticket in the console and submit information about the error in the ticket for further consultation.

The reply messages of service callings and property settings

Error code	Description	Cause	Troubleshooting method
Common error codes			
460	Invalid parameters.	The request parameters are invalid.	Open a ticket in the console for troubleshooting.
500	A system exception occurred.	An unknown exception occurred in the system.	Open a ticket in the console for troubleshooting.
400	An error occurred when calling the service.	An unknown exception occurred when calling the service.	Open a ticket in the console for troubleshooting.
429	Too many requests in the specified time period.	Traffic throttling has been triggered because too many requests have been submitted.	Open a ticket in the console for troubleshooting.
System exception codes			
6452	Traffic limiting	Traffic throttling has been triggered because too many requests have been submitted. Note: If the data type of the device is Do not parse/Custom, you may receive this error code. The input parameters will be verified again based on the TSL of the device.	Open a ticket in the console for troubleshooting.

Common error codes about TSL

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

Error code	Description	Cause	Troubleshooting method
6321	The identifier of the property is not found in the TSL.	A system exception occurred.	Open a ticket in the console and submit information about the error in the ticket for further consultation.
6317	The TSL of the device product is incorrect.	A system exception occurred.	Open a ticket in the console and submit information about the error in the ticket for further consultation.
6302	Required parameters are not found.	When verifying the input parameters of the service, the system does not find one or more required parameters in the request.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs for the required parameters. Check the parameters in the TSL and make sure that you have input all the required parameters.
6306	The input parameter does not comply with the integer data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The input parameter value is not in the range defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type in the TSL.

Error code	Description	Cause	Troubleshooting method
6307	The input parameter does not comply with the 32-bit floating point data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The input parameter value is not in the range defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type defined in the TSL, and the value is in the value range defined in the TSL.
6322	The input parameter does not comply with the 64-bit floating point data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The input parameter value is not in the range defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type defined in the TSL and the value is in the value range defined in the TSL.
6308	The input parameter does not comply with the boolean data specificat ion defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The input parameter value is not in the range defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type in the TSL.

Error code	Description	Cause	Troubleshooting method
6309	The input parameter does not comply with the enum data specification defined in the TSL.	The data type of the input parameter is different from the data type defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type defined in the TSL.
6310	The input parameter does not comply with the text data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The length of the input data exceeds the length limit defined in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type defined in the TSL and the data length does not exceed the limit.
6311	The input parameter does not comply with the date data specificat ion defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The input data is not a UTC timestamp.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type defined in the TSL.

Error code	Description	Cause	Troubleshooting method
6312	The input parameter does not comply with the struct data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The data type of the input parameter is different from the data type defined in the TSL. · The number of the struct data type parameters that you have input is different from the number of struct parameters defined in TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type in the TSL.
6304	The input parameter is not found in the defined struct parameters in the TSL.	When the parameters are verified according to the TSL, one or more input struct parameters are not found in the defined struct parameters in the TSL.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and make sure that the data type that you have input is the same as the data type in the TSL.
6324	The input parameter does not comply with the array data specification defined in the TSL.	When the parameters are verified according to the TSL, the following errors may be found: · The element data type that you input is different from the element type defined in the TSL. · The number of array type parameters that you have input exceeds the limit of array type parameters defined in the TSL.	· On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, and check the array type parameters. · View the upstream logs of the device, and check the number of array type elements in the data reported by the device.

Error code	Description	Cause	Troubleshooting method
6328	The input parameter is not an array type data.	When the parameters are verified according to the TSL, an input value of array type parameter is not of array type data.	On the product details page in the IoT Platform console, view the TSL of the product to which the device belongs, check the array type parameters in the TSL, and then check whether or not the parameter that you have input is of array type data.
6325	The element type of array type data is not supported by IoT Platform.	This error is reported when the parameters that you input according to the TSL are being verified. Currently, only the following element types of array type data are supported: int32, float, double, text, and struct.	Make sure that the element type that you have input is supported by IoT Platform.
System exception codes			
6318	A system exception occurred when parsing the TSL.	A system exception occurred.	Open a ticket in the console and submit information about the error in the ticket for further consultation.
6329	Failed to parse the data of the array type data specificat ion in the TSL when verifying the parameters.		
6323	The parameter type of the TSL is incorrect.		

Error code	Description	Cause	Troubleshooting method
6316	An error occurred when parsing the parameters in the TSL.		
6314	The data type is not supported.		
6301	An error occurred when verifying the input parameter type according to the TSL.		
Data parsing errors			
26010	Traffic throttling has been triggered because too many requests have been submitted.	Too many requests in the specified time period.	Open a ticket in the console for troubleshooting.
26001	The content of the parsing script is empty.	The parsing script content is not found.	On the product details page in the IoT Platform console, check your data parsing script. Make sure that the script has been saved and submitted. A draft script cannot be used to parse data.

Error code	Description	Cause	Troubleshooting method
26002	An exception occurred when running the script.	The script runs properly, however, the script content is incorrect, for example, there are syntax mistakes in the script.	In the IoT Platform console, enter the same parameters and run the script to debug. The console only has a basic script running environment. Therefore, it cannot precisely verify the content of the script. We recommend that you inspect your script carefully before you submit it.
26006	The required method is not found in the script.	The script runs properly, however, the script content is incorrect. protocolToRawData and rawDataToProtocol are required in a script. If they are not found, this error will be reported.	On the product details page in the IoT Platform console, check that protocolToRawData and rawDataToProtocol have been defined.

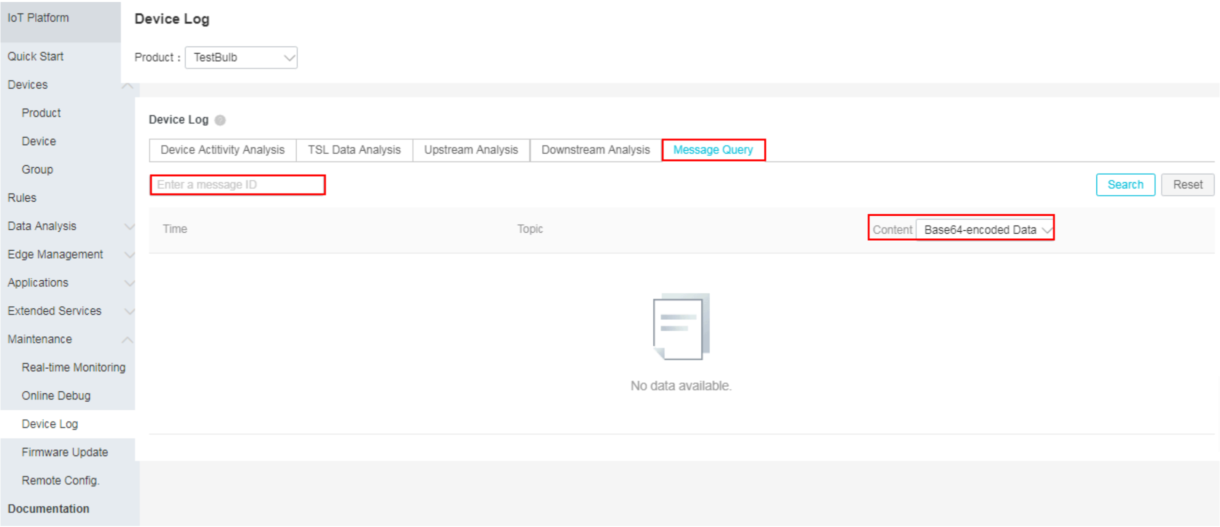
Error code	Description	Cause	Troubleshooting method
26007	The returned data type is incorrect after data parsing.	The script runs properly, but the returned result data type is incorrect. Check the definitions of protocolToRawData and rawDataToProtocol. The result data of protocolToRawData must be byte[] array, and the result data of rawDataToProtocol must be jsonObj (JSON object). If the defined result data types are not these two types, this error will be returned. After a device reports data, the execution result will be returned to the device. The returned result data also will be parsed. If you have not defined protocolToRawData in the script, the returned data may be incorrect.	Inspect the script in the IoT Platform console. Enter the input parameters, run the script, and verify whether the result data type is correct.

Query message content

Click Message Query and then query the payload contents that are sent by devices.

Search for payload contents by message IDs. Currently, only messages with QoS 1 can be queried.

You can select to display the original data or the Base64-encoded data.



3.3 Firmware update

IoT Platform provides the firmware update function. To update firmware, you need to configure your device client 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 client 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

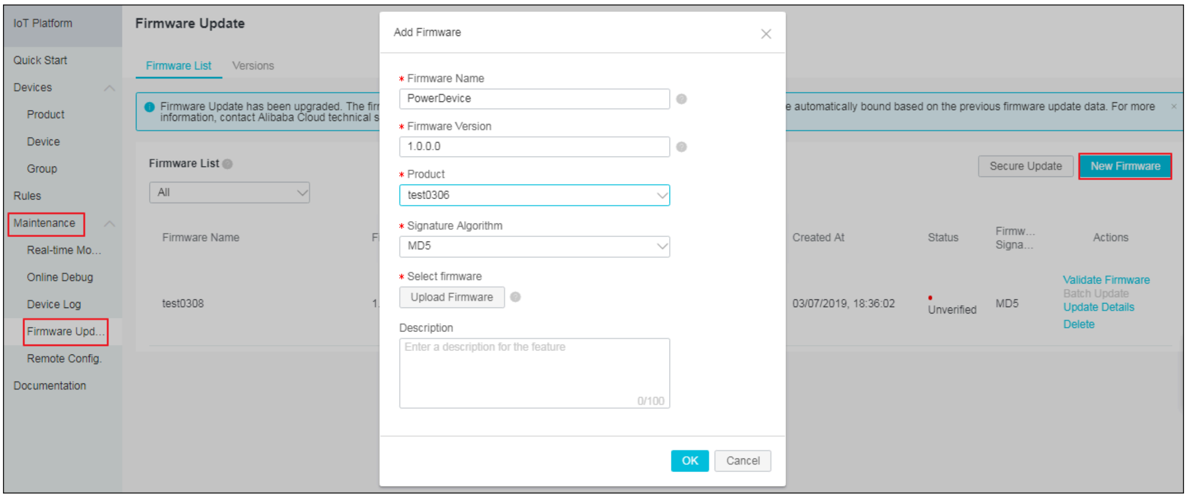
Note:

To provide better services, IoT Platform now allows you to manage firmware versions by product. As such, when you use the new version of the firmware update function for the first time, you need to associate your previously uploaded firmware files with your products manually. You can only associate a firmware file to one product. After you associate your 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 100 firmware files.

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



Firmware Update has been upgraded. The fir... information, contact Alibaba Cloud technical s... ...e automatically bound based on the previous firmware update data. For more

test0308 | 1. | 03/07/2019, 18:36:02 | Unverified | MD5

Add Firmware

* Firmware Name: PowerDevice

* Firmware Version: 1.0.0.0

* Product: test0306

* Signature Algorithm: MD5

* Select firmware

Description: Enter a description for the feature 0/100

Table 3-2: 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 10 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*.

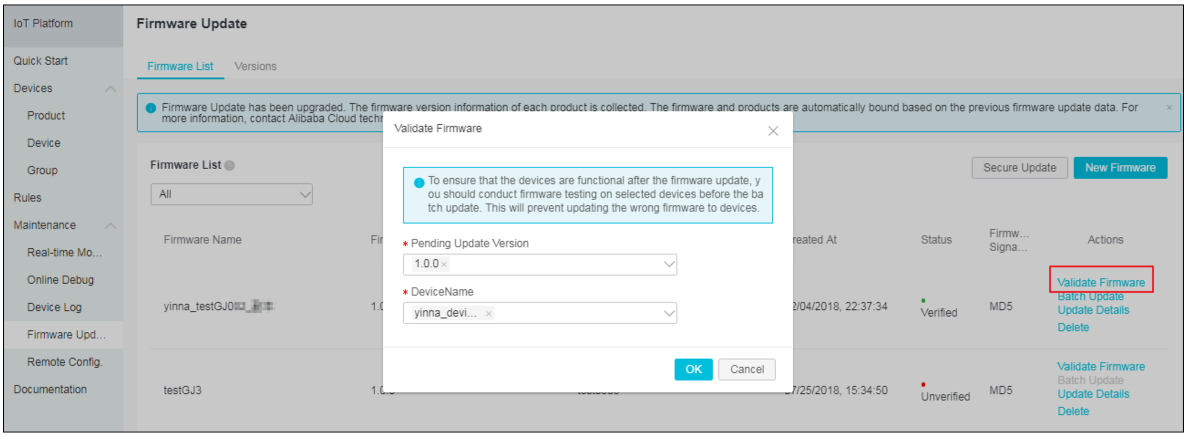
a) On the Firmware Update page, click Secure Update.

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 verify whether the uploaded firmware file is available.

> Note:
>
> After the firmware file is uploaded to IoT Platform, you need to test whether the firmware file is available on one or more devices. Only when you confirm that the test devices have been successfully updated can the firmware file be used for batch update. You can launch validations for a firmware to occur multiple times.



Firmware Update has been upgraded. The firmware version information of each product is collected. The firmware and products are automatically bound based on the previous firmware update data. For more information, contact Alibaba Cloud techr

Validate Firmware

To ensure that the devices are functional after the firmware update, you should conduct firmware testing on selected devices before the batch update. This will prevent updating the wrong firmware to devices.

* Pending Update Version: 1.0.0

* DeviceName: yinna_devi...

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.

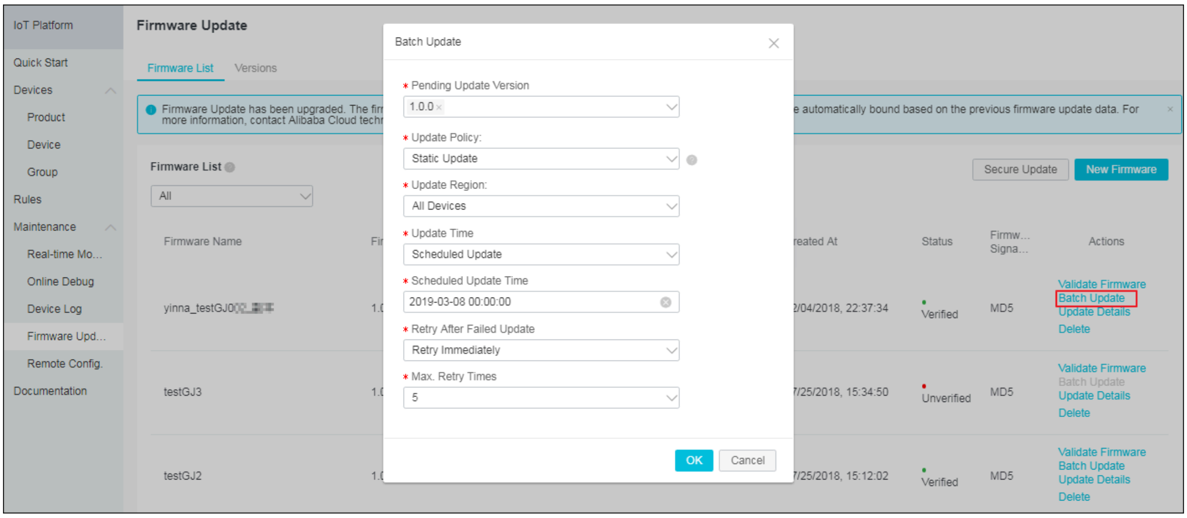
> Note:

- 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 or that the firmware file is available. 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.



Firmware Update has been upgraded. The fir... more information, contact Alibaba Cloud techn... e automatically bound based on the previous firmware update data. For

Batch Update

- * Pending Update Version: 1.0.0
- * Update Policy: Static Update
- * Update Region: All Devices
- * Update Time: Scheduled Update
- * Scheduled Update Time: 2019-03-08 00:00:00
- * Retry After Failed Update: Retry Immediately
- * Max. Retry Times: 5

Firmware Name	Fi...	...reated At	Status	Firmw... Signa...	Actions
yinna_testGJ0[redacted]	1.0	...2/04/2018, 22:37:34	Verified	MD5	Validate Firmware Batch Update Update Details Delete
testGJ3	1.0	...7/25/2018, 15:34:50	Unverified	MD5	Validate Firmware Batch Update Update Details Delete
testGJ2	1.0	...7/25/2018, 15:12:02	Verified	MD5	Validate Firmware Batch Update Update Details Delete

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.
Update Region	- All Devices: All devices that belong to the product will be updated. - Directional Upgrade: If you select Directional Upgrade, 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.

3.4 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

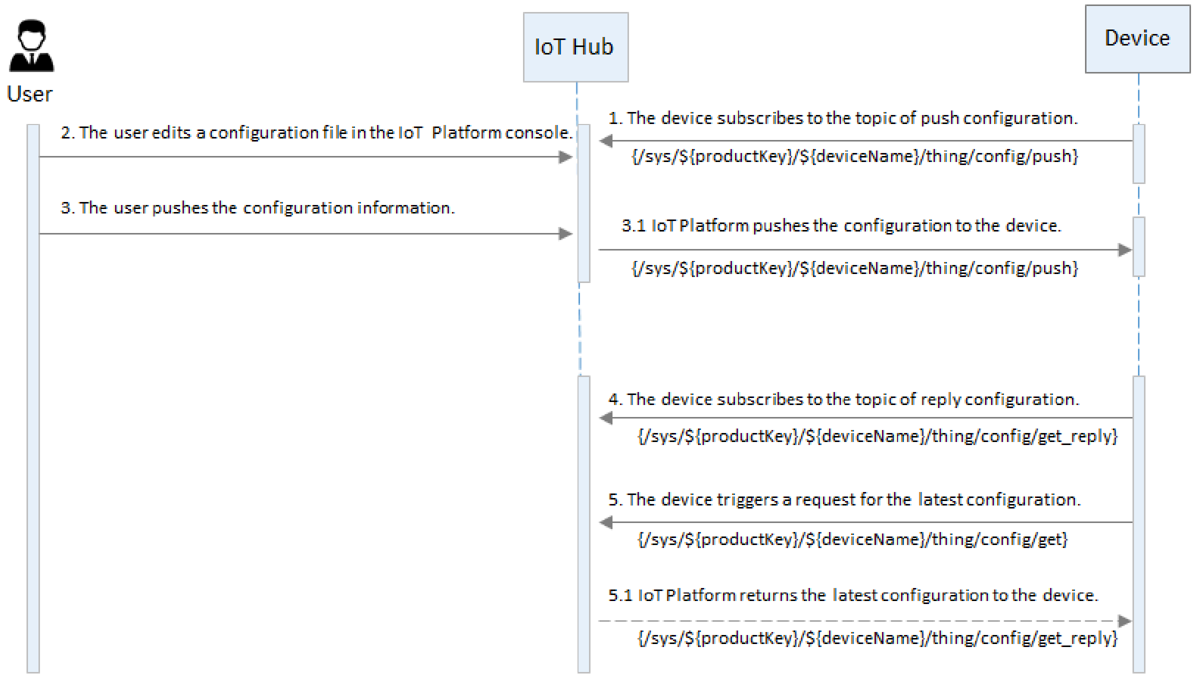
Developers often need to update device configurations, such as the system parameters, network parameters, and security policies of devices. Generally, device configurations are updated using the firmware update function. However, firmware update requires more time for firmware version maintenance, and devices must stop their services in order to install the update. To streamline the device configurat ion 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:


User
IoT Hub
Device
1. The device subscribes to the topic of push configuration.
{/sys/${productKey}/${deviceName}/thing/config/push}
2. The user edits a configuration file in the IoT Platform console.
3. The user pushes the configuration information.
3.1 IoT Platform pushes the configuration to the device.
{/sys/${productKey}/${deviceName}/thing/config/push}
4. The device subscribes to the topic of reply configuration.
{/sys/${productKey}/${deviceName}/thing/config/get_reply}
5. The device triggers a request for the latest configuration.
{/sys/${productKey}/${deviceName}/thing/config/get}
5.1 IoT Platform returns the latest configuration to the device.
{/sys/${productKey}/${deviceName}/thing/config/get_reply}

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.

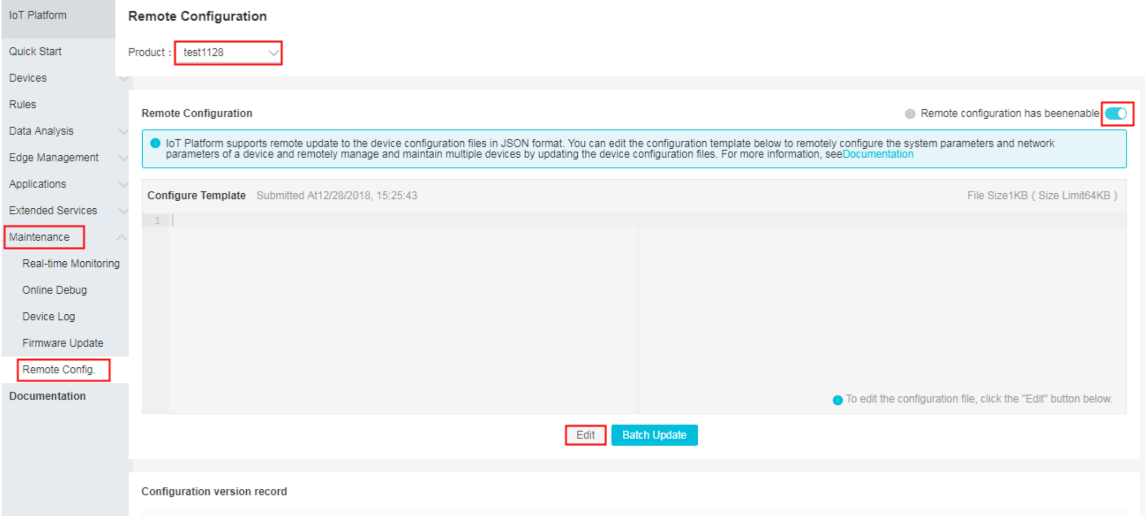
Scenario 1: Push configuration information to devices from IoT Platform.

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.



Remote Configuration

Product : test1128

Remote Configuration ● Remote configuration has beenenable

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, seeDocumentation

Configure Template Submitted At12/28/2018, 15:25:43 File Size1KB (Size Limit64KB)

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

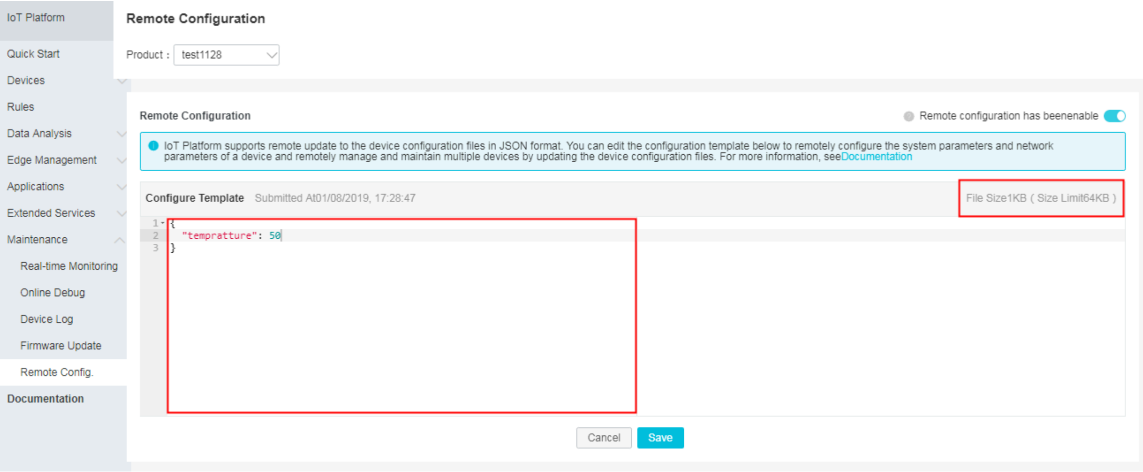
Configuration version record

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.



Remote Configuration

Product : test1128

Remote Configuration

Remote configuration has beenenable

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, seeDocumentation

Configure Template Submitted At01/08/2019, 17:28:47

File Size1KB (Size Limit64KB)

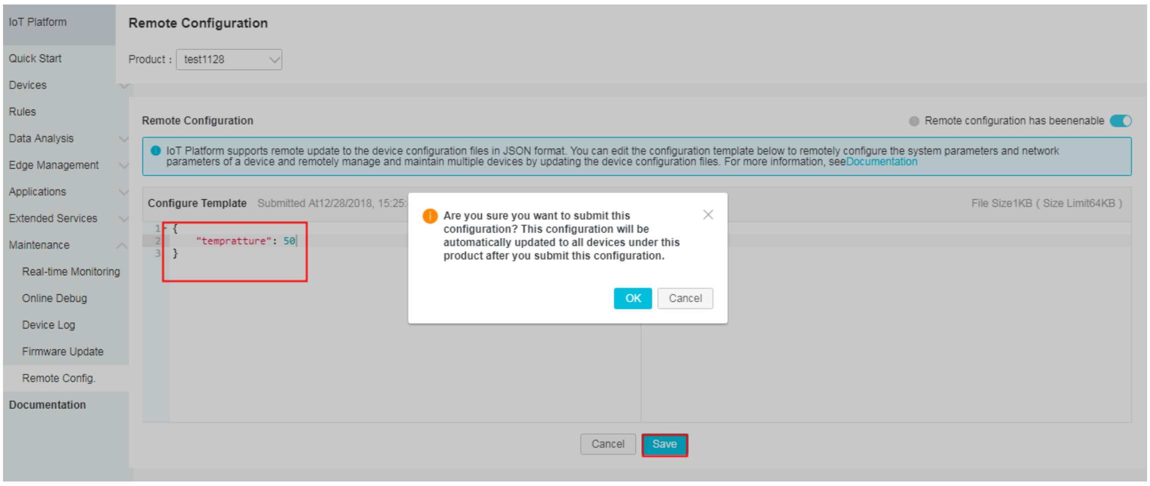
```
{
  "tempratture": 50
}
```

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.



Remote Configuration

Product : test1128

Remote Configuration

Remote configuration has beenenable

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, seeDocumentation

Configure Template Submitted At12/28/2018, 15:25:

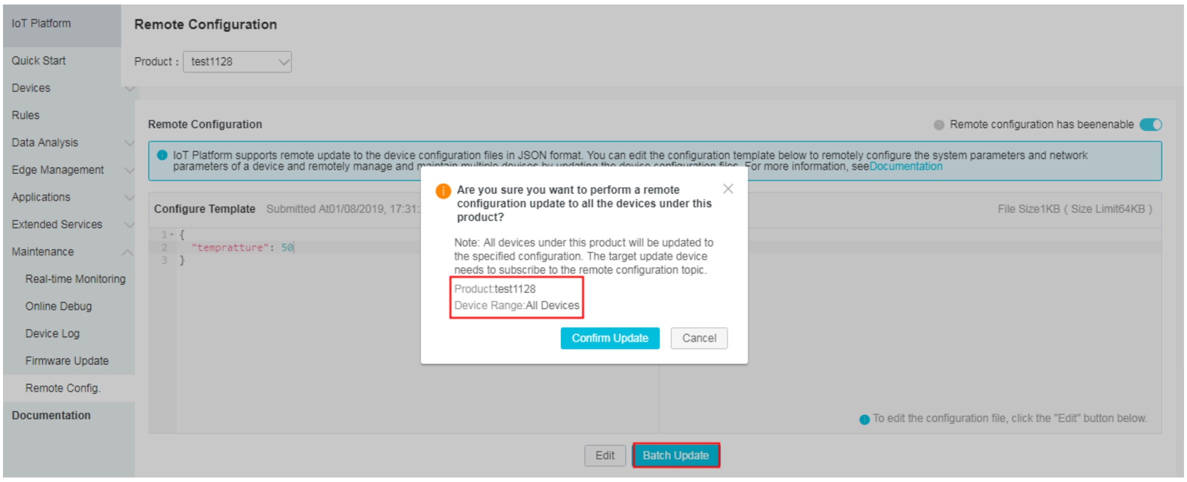
File Size1KB (Size Limit64KB)

```
{
    "tempratture": 50
}
```

Are you sure you want to submit this configuration? This configuration will be automatically updated to all devices under this product after you submit this configuration.

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.



Remote Configuration

Product : test1128

Remote Configuration

Remote configuration has beenenable

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 For more information, seeDocumentation

Configure Template Submitted At01/08/2019, 17:31:

File Size1KB (Size Limit64KB)

```
{
  "tempratture": 50
}
```

Are you sure you want to perform a remote configuration update to all the devices under this product?

Note: All devices under this product will be updated to the specified configuration. The target update device needs to subscribe to the remote configuration topic.

Product:test1128
Device Range:All Devices

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

Note:

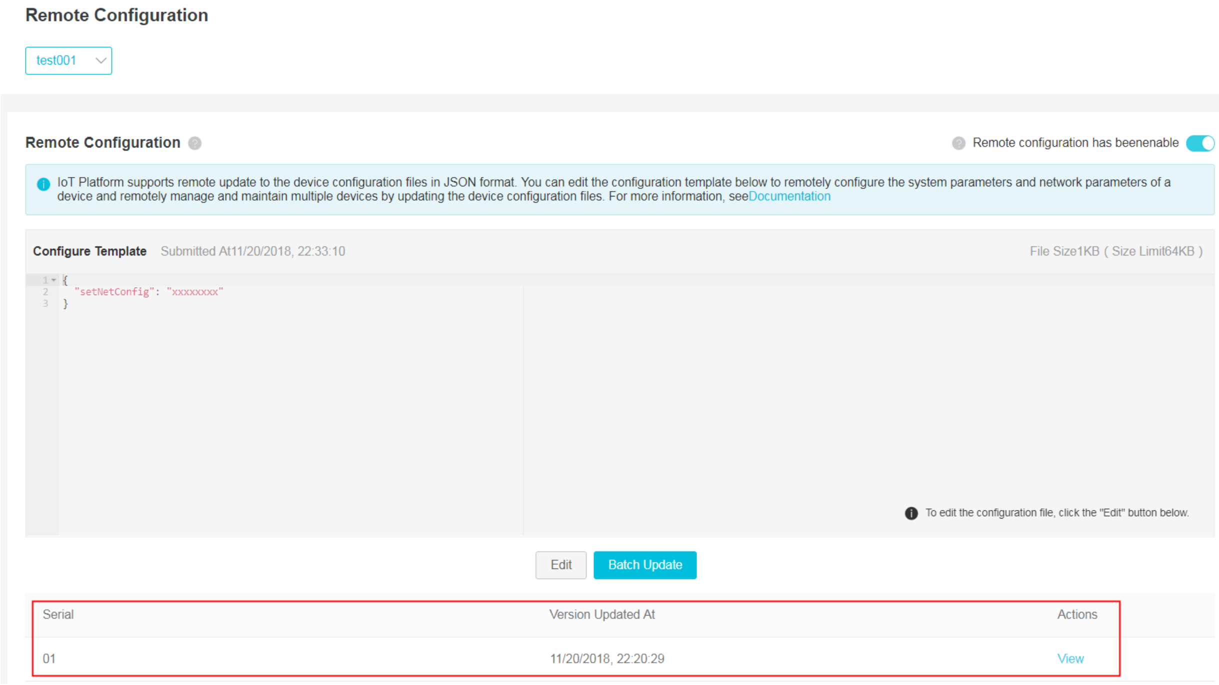
- Operation frequency limit: You can only perform a batch update once per hour.

- 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 beenenable

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, seeDocumentation

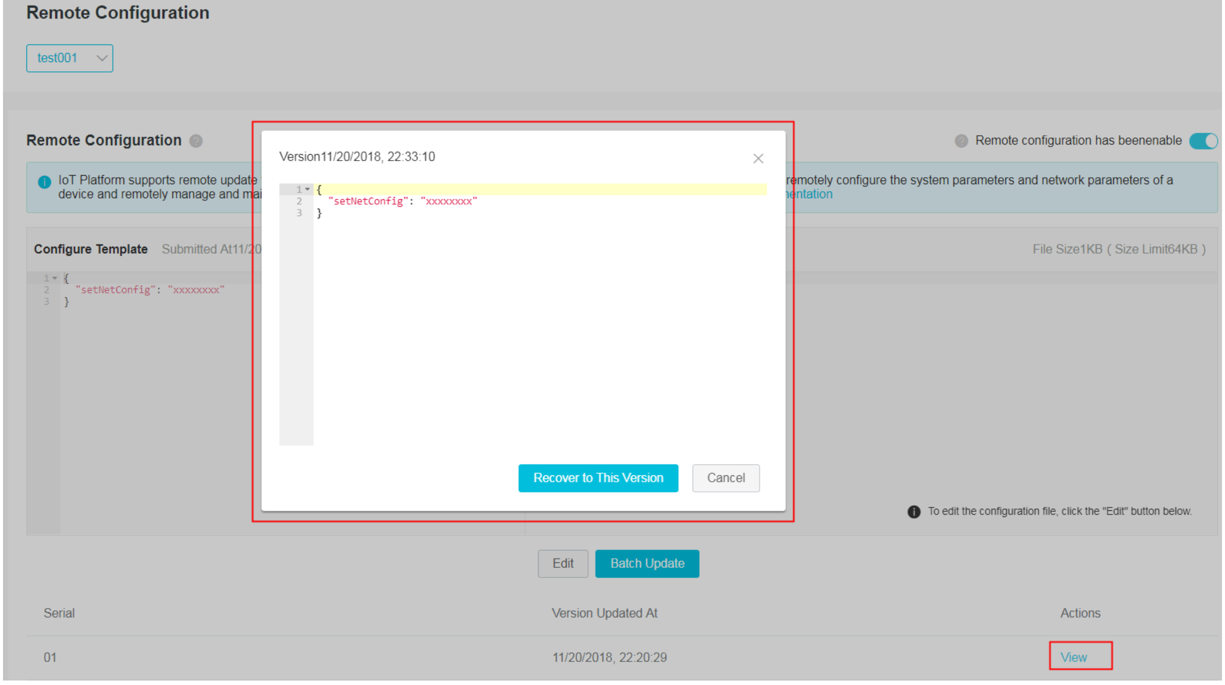
Configure Template Submitted At11/20/2018, 22:33:10 File Size1KB (Size Limit64KB)

```
{
  "setNetConfig": "xxxxxxxx"
}
```

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

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.



Remote Configuration

test001

Remote Configuration

Remote configuration has beenenable

IoT Platform supports remote update
device and remotely manage and mai

remotely configure the system parameters and network parameters of a
entation

Configure Template Submitted At11/20

File Size1KB (Size Limit64KB)

```
{
  "setNetConfig": "xxxxxxxx"
}
```

Version11/20/2018, 22:33:10

```
{
  "setNetConfig": "xxxxxxxx"
}
```

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

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

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_callb ack` function to process the configuration file that is sent through the remote configuration function.

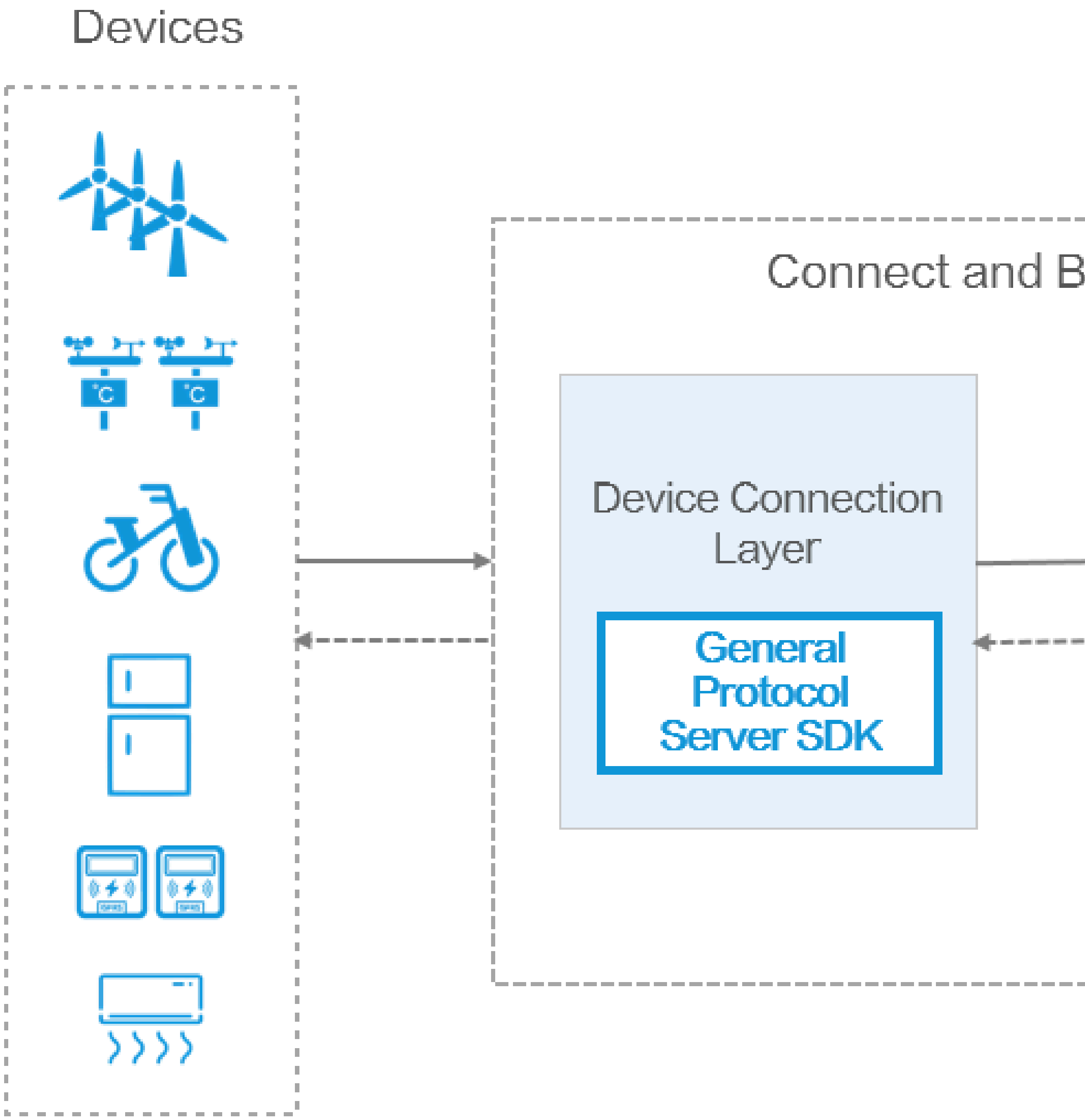
4 General protocols

4.1 Overview

The Alibaba Cloud IoT Platform already supports MQTT, CoAP, HTTP and other common protocols, yet fire protection agreement GB/T 26875.3-2011, Modbus and JT808 is not supported, and in some specialized cases, devices may not be able to connect to IoT Platform. At this point, you need to use general protocol SDK to quickly build a bridge between your devices and platform to Alibaba Cloud IoT Platform, allowing two-way data communication.

General protocol SDK

The general protocol SDK is a protocol self-adaptive framework, using for high-efficiency bi-directional communication between your devices or platform to IoT Platform. The SDK architecture is shown below:



General protocol provides two SDKs: Core SDK and Server SDK.

- General protocol core SDK

  Core SDK abstracts abilities like session and configuration management. It acts like a net bridge between devices and IoT Platform and communicates with the

Platform in representation of devices. This greatly simplifies the development of IoT Platform. Its main features include:

- provides non-persistent session management capabilities.
- provides configuration management capabilities based on configuration files.
- provides connection management capabilities.
- provides upstream communication capability.
- provides downstream communication capabilities.
- supports device authentication.

If your devices are already connected to the internet and you want to build a bridge between IoT Platform and your devices or existing platform, choose core SDK.

- General protocol server SDK

  Server SDK provides device connection service on the basis of core SDK function. Its main features include:

  - supports any protocol that is based on TCP/UDP.
  - supports TLS/SSL encryption for transmission.
  - supports horizontal expansion of the capacity of device connection.
  - provides Netty-based communication service.
  - provides automated and customizable device connection and management capability.

  If you want to build the connection service from scratch, choose server SDK which provides socket for communication.

Development and deployment

Create products and devices in IoT console

Create products and devices in console. See *Create a product (Pro Edition)* for more information. Acquire the ProductKey, DeviceName and DeviceSecret of the net bridge device you've just created.

> Note:
>
> Net bridge is a virtual concept, and you can use the information of any device as device information of the net bridge.

SDK dependency

General protocol SDKs are currently only available in Java, and supports JDK 1.8 and later versions. Maven dependencies:

```
<! --  Core   SDK  -->
< dependency >
    < groupId > com . aliyun . openservic  es </ groupId >
    < artifactId > iot - as - bridge - sdk - core </ artifactId >
    < version > 1 . 0 . 0 </ version >
</ dependency >

<! --  Server   SDK  -->
< dependency >
    < groupId > com . aliyun . openservic  es </ groupId >
    < artifactId > iot - as - bridge - sdk - server </ artifactId >
    < version > 1 . 0 . 0 </ version >
</ dependency >
```

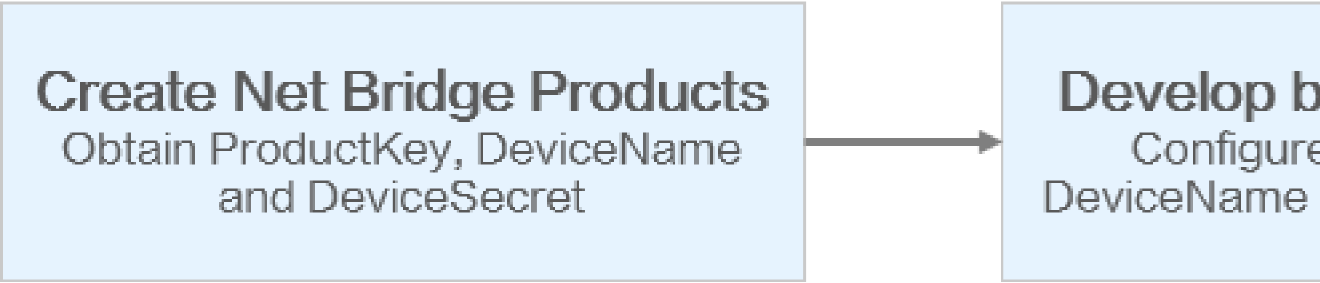
Develop SDK

#unique_79and Server SDK briefly introduces the development process. For detailed implementation, refer to javadoc.

Deployment

The completed bridge connection service can be deployed on Alibaba Cloud using services like ECS and SLB, or deployed in local environment to guarantee communication security.

The whole process (if using Alibaba Cloud ECS to deploy) is shown below:


Create Net Bridge Products
Obtain ProductKey, DeviceName and DeviceSecret
Develop b
Configure
DeviceName

4.2 Develop Core SDK

You can integrate the IoT Platform bridge service with existing connection services or platforms that use the general protocol core SDK to allow devices or servers to quickly access Alibaba Cloud IoT Platform.

Prerequisites

For information about the concepts, functions, and Maven dependencies of the general protocol core SDK, see *Overview*.

Configuration management

The general protocol core SDK uses file-based configuration management by default. For information about customized configurations, see *Custom components > Configuration management*. The general protocol core SDK supports:

- Java Properties, JSON, and *HOCON* formats.
- Structured configuration to simplify maintenance.
- The override of file configurations with Java system properties, such as java -Dmyapp.foo.bar=10.
- Configuration file separation and nested references.

Table 4-1: application.conf

Net bridge is a virtual concept. You can use the `productKey` , `deviceName` , and `deviceSecret` of any device as the information of the net bridge.

Parameter	Required	Description
`productKey`	Yes	The product ID of the net bridge product.
`deviceName`	No	The device name of the net bridge device. The default value is the ECS instance MAC address.
`deviceSecret`	No	The device secret of the net bridge device.

Parameter	Required	Description
`http2Endpoint`	Yes	HTTP/2 gateway service address. The address format is `${UID}.iot-as-http2.${RegionId}.aliyuncs.com:443`. where: · `${UID}` indicates your account ID. To view your account ID, log on to the Alibaba Cloud console, hover your mouse over your account image, and click Security Settings. You are then directed to the Account Management page that displays your account ID. · `${RegionId}` indicates the region ID where your service is located. For example, if the region is Shanghai, the HTTP/2 gateway service address is `123456789 . iot - as - http2 . cn - shanghai . aliyuncs . com : 443`. For information about RegionId expressions, see *Regions and zones*.
`authEndpoint`	Yes	Device authentication service address Device authentication service address: `https://iot-auth. ${RegionId}.aliyuncs.com/auth/bridge`. `${RegionId}` indicates the region ID where your service is located. For example, if the region is Shanghai, the device authentication service address is `https :// iot - auth . cn - shanghai . aliyuncs . com / auth / bridge`. For information about RegionId expressions, see *Regions and zones*.
`popClientProfile`	Yes	Call APIs to configure the client. For details, see the *API client configuration*.

Table 4-2: API client configuration

Parameter	Required	Description
accessKey	Yes	The access key of the API caller.
accessSecret	Yes	The secret key of the API caller.

Parameter	Required	Description
name	Yes	The region name of the API.
region	Yes	The region ID of the API.
product	Yes	The name of the product. Set it to `Iot` if not specified.
endpoint	Yes	The endpoint of the API. Endpoint structure: `iot.${RegionId}.aliyuncs.com`. `${RegionId}` indicates the region ID of your service. For example, If the region is Shanghai, the endpoint is `iot . cn - shanghai . aliyuncs . com`. For information about RegionId expressions, see *Regions and zones*.

devices.conf

Configure the ProductKey, DeviceName, and DeviceSecret of the device. For information about customizing configuration files, see *Custom components > Configuration management*.

```
 XXXX  //   Original   identifier  s   of   the   device
{
    " productKey ": " 123 ",
     deviceName : "",
     deviceSecr  et : ""
}
```

Interfaces

Initialization

`com . aliyun . iot . as . bridge . core . BridgeBoot  strap` initializes the communication between the device and Alibaba Cloud IoT Platform. After the BridgeBootstrap instance is created, the *Basic configurations* component of the gateway will be initialized. For information about customizing configurations, see *Custom components > Configuration management*.

Complete the initialization using one of the following interfaces:

- `bootstrap ()`: initialization without downstream messaging.

- `bootstrap(DownlinkChannelHandler handler)`: initialize using `DownlinkChannelHandler` specified by the developer.

Sample code:

```
 BridgeBootstrap bootstrap = new BridgeBootstrap();
// Do not implement downstream messaging
 bootstrap.bootstrap();
```

Connect devices to IoT Platform

Only devices that are online can establish a connection with or send connection requests to IoT Platform. There are two methods that can enable devices to get online : local session initialization and device authentication.

1. Session initialization

   The general protocol SDK provides non-persistent local session management. See *Custom components > Session management* for information on customization.

   Interfaces for creating new instances:

   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel)`
   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel, int heartBeatInterval)`
   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel, int heartBeatInterval, int heartBeatProbes)`

   `originalIdentity` indicates the unique device identifier and has the same function as SN in the original protocol. `channel` is the communication channel between devices and bridge service, and has the same function as a channel in Netty. `heartBeatInterval` and `heartBeatProbes` are used for heartbeat monitoring. The unit of heartBeatInterval is seconds. heartBeatProbes indicates the maximum number of undetected heartbeats that is allowed. If this number is exceeded, a heartbeat timeout event will be sent. To handle a timeout event, register `com.aliyun.iot.as.bridge.core.session.SessionListener`.

2. Authenticate devices

   After the initialization of local device session, use `com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler.doOnline(Session newSession, String originalIdentity, String... credentials)` to complete local device authentication and Alibaba Cloud IoT Platform online authentication. The device will then either be allowed to communicate or will be disconnected according to the authentication result. SDK provides online authentication for IoT Platform. By default, local authentication is disabled. If you need to set up local authentication, see *Customized components > Connection authentication*.

   Sample code:

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
Session session = session.newinstance(device, Channel);
boolean success = uplinkHandler.doOnline(session, originalIdentity);
if (success) {
    // Successfully got online, and will accept communication requests.
} else {
    // Failed to get online, and will reject communication requests and disconnect (if connected).
}
```

Device Offline

When a device disconnects or detects that it needs to disconnect, a device offline operation must be initiated. Use `com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler.doOffline(String originalIdentity)` to bring a device offline.

Sample code:

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
Uplinkhandler.dooffline(originalidentity);
```

Report Data

You can use `com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler` to report data to Alibaba Cloud IoT Platform. Data reporting involves three key steps: identify the device that is going to report data, locate the corresponding session for this device, and report data to IoT Platform. Use the following interfaces to report data.

Note:
Make sure that the data report has been managed and security issues have been handled.

- `CompletableFuture doPublishAsync(String originalIdentity, String topic, byte[] payload, int qos)`: send data asynchronously and return immediately. You can then obtain the sending result using future.
- `CompletableFuture doPublishAsync(String originalIdentity, ProtocolMessage protocolMsg)`: send data asynchronously and return immediately. You can then obtain the sending result using future.
- `boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg, int timeout)`: send data asynchronously and wait for the response.
- `boolean doPublish(String originalIdentity, String topic, byte[] payload, int qos, int timeout)`: send data asynchronously and wait for the response.

Sample code:

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
DeviceIdentity identity = ConfigFactory.getDeviceConfigManager().getDeviceIdentity(device);
if (identity == null) {
    // Devices are not mapped with those registered on IoT Platform, and messages are dropped.
    return;
}
Session session = SessionManagerFactory.getInstance().getSession(device);
if (session == null) {
    // The device is not online. You can either get the device online or drop messages. Make sure devices are online before reporting data to IoT Platform.
}
boolean success = uplinkHandler.doPublish(session, topic, payload, 0, 10);
if (success) {
    // Data is successfully reported to Alibaba Cloud IoT Platform.
} else {
    // Failed to report data to IoT Platform
}
```

Downstream Messaging

The general protocol SDK provides `com.aliyun.iot.as.bridge.core.handler.DownlinkChannelHandler` as the downstream data distribution processor. It supports unicast and broadcast (if the message sent from the cloud does not include specific device information).

Sample code:

```
public class SampleDownlinkHandler implements DownlinkChannelHandler {
    @Override
    public boolean pushToDevice(Session session, String topic, byte[] payload) {
        // Process messages pushed to the device
    }

    @Override
    public boolean broadcast(String topic, byte[] payload) {
        // Process broadcast
    }
}
```

Custom components

You can customize the device connection authentication, session management, and configuration management components. You must complete the initialization and substitution of those components before calling BridgeBootstrap intialization.

Connection authentication

To customize the device connection authentication, implement `com.aliyun.iot.as.bridge.core.auth.AuthProvider` and then, before initializing BridgeBootstrapcall, call `com.aliyun.iot.as.bridge.core.auth.AuthProviderFactory.init(AuthProvider customizedProvider)` to replace the original authentication component with the customized component.

Session management

To customize the session management, implement `com.aliyun.iot.as.bridge.core.session.SessionManager` and then, before initializing BridgeBootstrapcall, call `com.aliyun.iot.as.bridge.core.session.SessionManagerFactory.init(SessionManager<?>`

`customizedSessionManager)` to replace the original session management component with the customized component.

Configuration management

To customize the configuration management, implement `com.aliyun.iot.as.bridge.core.config.DeviceConfigManager` and `com.aliyun.iot.as.bridge.config.BridgeConfigManager`. Then, before initializing BridgeBootstrapcall, call `com.aliyun.iot.as.bridge.core.config.ConfigFactory.init(BridgeConfigManager bcm, DeviceConfigManager dcm)` to replace the original configuration management component with the customized component. If the parameters are left empty, the general protocol SDK default values will be used.

4.3 Server SDK

4.3.1 Interfaces for UDP

You can build an access service which uses UDP transmission protocol and bridge it to Alibaba Cloud IoT Platform using the interfaces of the general protocol SDK for UDP.

Bootstrap

com.aliyun.iot.as.bridge.server.BridgeServerBootstrap is the bootstrap class for booting socket server and bridge service. After a new BridgeServerBootstrap is created, components based on configuration files will be initialized.

Example:

```
 BridgeServerBootstrap bootstrap = new BridgeServerBootstrap(new UdpDecoderFactory() {
    @Override
  public MessageToMessageDecoder newInstance() {
// Return decoder
    }
}, new UdpEncoderFactory() {
    @Override
 public MessageToMessageEncoder<?> newInstance() {
// Return encoder
    }
}, new UdpBasedProtocolAdaptorHandlerFactory() {
    @Override
 public CustomizedUdpBasedProtocolHandler newInstance() {
// Return protocol adapter
    }
});
```

```
  try  {
      bootstrap . start ();
 }  catch  ( BootExcept  ion  |  ConfigExce  ption   e ) {
 //  Process   boot   exception
 }
```

Instantiation of UDP type BridgeServerBootstrap

- com.aliyun.iot.as.bridge.server.channel.factory.UdpDecoderFactory: Create a new decoder instance using the factory method to decode upload data. Thread is secure. Can be null.
- com.aliyun.iot.as.bridge.server.channel.factory.UdpEncoderFactory: Create a new encoder instance using the factory method to encode downstream data to adapt to UDP protocol. Thread is secure. Can be null.
- com.aliyun.iot.as.bridge.server.channel.factory.UdpBasedProtocolAdaptorHandlerFactory: Create a new protocol adapter instance using the factory method to adapt decoded data so they can be uploaded to the cloud. Thread is secure. Cannot be null.

Start socket server

After the creation of BridgeServerBootstrap, call com.aliyun.iot.as.bridge.server.BridgeServerBootstrap.start() to start the socket server.

Protocol decoding

The component for protocol decoding derives from io.netty.handler.codec.MessageToMessageDecoder<I>. Refer to *MessageToMessageDecoder Documentation* for details.

Example:

```
  public   class   SampleDeco  der   extends    MessageToM  essageDeco
  der < DatagramPa  cket > {
     @ Override
  protected   void   decode ( ChannelHan  dlerContex  t   ctx ,
  DatagramPa  cket   in ,  List < Object >  out )  throws   Exception
  {
 //  The   decoding   protocol
     }
 }
```

Protocol encoding

The component for protocol encoding derives from io.netty.handler.codec.MessageToMessageEncoder<I>. Refer to *MessageToMessageEncoder Documentation* for details.

Example:

```
public   class   SampleEnco der   extends   MessageToM essageEnco
der < T >{
   @ Override
    protected   void   encode ( ChannelHan dlerContex t   ctx ,  T
  msg ,  ByteBuf   out )  throws   Exception  {
//  Protocol   encoding
   }
}
```

Protocol adapter

To reduce cost and improve the efficiency of development, the general protocol server SDK provides protocol adapters with extensible and customizable basic capability class com.aliyun.iot.as.bridge.server.channel.CustomizedUdpBasedProtocolHandler. It encapsulates details to access Alibaba Cloud IoT Platform, so you can focus on other business. The protocol adapter derives from this class.

Device Online

Only online devices can establish a connection with or send connection requests to IoT Platform. There are two steps for devices to get online: local session initialization and device authentication.

1. Session Initialization

   Refer to *Core SDK develop > Device Online > Session Initialization* for details.

2. Device Authentication

   After local session initialization, call doOnline(Session newSession, String originalIdentity, String... credentials) or doOnline(String originalIdentity, String ... credentials) to complete local device authentication and Alibaba Cloud IoT Platform online authentication. The device can communicate with IoT Platform if authentication succeeds, and will be disconnect from IoT Platform if authentication fails.

   Example:

```
Session   session   =  Session . newInstanc  e ( device ,  channel
);
boolean   success   =  doOnline ( session ,  originalId  entity );
if  ( success ) {
   //  Successful  ly   got   online ,  and   will   accept
communicat  ion   requests .
}  else  {
   //  Failed   to   get   online ,  and   will   reject
communicat  ion   requests   and   disconnect  ( if   connected ).
}
```

Device Offline

When the device is disconnected or detects that it needs to be disconnected, the device offline action should be initiated. Using server SDK, devices will automatically get offline when they are disconnected, so you can focus on other tasks. Call doOffline (Session session) to bring devices offline.

Report Data

The protocol adapter needs to use override channelRead(ChannelHandlerContext ctx , Object msg). It is the entrance for all devices to report data. Object msg is the data output from the decoder.

There are three steps for data reporting: identify the device that is going to report data, find the corresponding session for this device, and then report data to IoT Platform. Use the following interfaces to report data:

- CompletableFuture doPublishAsync(String originalIdentity, String topic, byte[] payload, int qos): send data asynchronously and return immediately. Developers obtain the sending result using future.
- CompletableFuture doPublishAsync(String originalIdentity, ProtocolMessage protocolMsg): send data asynchronously and return immediately. Developers obtain the sending result using future.
- boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg, int timeout): send data asynchronously and wait for the response.
- boolean doPublish(String originalIdentity, String topic, byte[] payload, int qos, int timeout): send data asynchronously and wait for the response.

Example:

```
 DeviceIden  tity   identity  =  ConfigFact  ory . getDeviceC
 onfigManag  er (). getDevivic  eIdentity ( device );
 if  ( identity  ==  null ) {
    //  Devices   are   not   mapped   with   those   registered   on
   IoT   Platform .  Messages   are   dropped .
     return ;
}
 Session   session  =  SessionMan  agerFactor  y . getInstanc  e ().
 getSession ( device );
 if  ( session  ==  null ) {
    //  The   device   is   not   online .  Please   get   online   or
   drop   messages .   Make   sure   devices   are   online   before
 reporting   data   to   IoT   Platform .
}
```

```
 boolean   success  =  doPublish ( session ,  topic ,  payload ,  0 ,
 10 );
 if ( success ) {
    //  Data   is   successful  ly   reported   to   Alibaba   Cloud
 IoT   Platform .
}  else  {
    //  Failed   to   report   data   to   IoT   Platform
}
```

Downstream Messaging

Not supported yet.

4.3.2 Interfaces for TCP

You can build an access service which uses TCP transmission protocol and bridge it to Alibaba Cloud IoT Platform using the interfaces of the general protocol SDK for TCP.

Bootstrap

com.aliyun.iot.as.bridge.server.BridgeServerBootstrap is the bootstrap class for booting socket server and bridge service. After a new BridgeServerBootstrap is created, components based on configuration files will be initialized.

Example:

```
 BridgeServ  erBootstra  p   bootstrap  =  new   BridgeServ
 erBootstra  p ( new   TcpDecoder  Factory () {
    @ Override
     public   ByteToMess  ageDecoder   newInstanc  e () {
        //  Return   decoder
    }
},  new   TcpEncoder  Factory () {
    @ Override
     public   MessageToB  yteEncoder <? >  newInstanc  e () {
//  Return   encoder
    }
},  new   TcpBasedPr  otocolAdap  torHandler  Factory () {
@  Override
     public   Customized  TcpBasedPr  otocolHand  ler   newInstanc  e
 () {
//  Return   protocol   adapter
    }
},  new   DefaultDow  nlinkChann  elHandler ());
 try  {
     bootstrap . start ();
}  catch  ( BootExcept  ion  |  ConfigExce  ption   e ) {
//  Process   boot   exception
}
```

Instantiation of TCP type BridgeServerBootstrap

- com.aliyun.iot.as.bridge.server.channel.factory.TcpDecoderFactory: Create a new decoder instance using factory method to decode upload data. Thread is secure. Can be null.

- com.aliyun.iot.as.bridge.server.channel.factory.TcpEncoderFactory: Create a new encoder instance using factory method to encode downstream data to adapt to TCP protocol. Thread is secure. Can be null.
- com.aliyun.iot.as.bridge.server.channel.factory.TcpBasedProtocolAdaptorHandlerFactory: Create a new protocol adapter instance using factory method to adapt decoded data so they can be uploaded to the cloud. Thread is secure. Cannot be null.
- com.aliyun.iot.as.bridge.core.handler.DownlinkChannelHandler: Distributor for downstream data. Supports unicast and broadcast. Unicast forwards data directly to the device by default. Broadcast settings must be customized by developers. Can be null. Null indicates that downstream data is not allowed.

Start socket server

After the creation of BridgeServerBootstrap, call com.aliyun.iot.as.bridge.server.BridgeServerBootstrap.start() to start the socket server.

Protocol decoding

The component for protocol decoding derives from io.netty.handler.codec.ByteToMessageDecoder. Refer to *ByteToMessageDecoder Documentation* for details.

Example:

```
 public   class   SampleDeco  der   extends   ByteToMess  ageDecoder
 {
    @ Override
     protected   void   decode ( ChannelHan  dlerContex  t   ctx ,
 ByteBuf   in ,  List < Object >  out )  throws   Exception  {
//  The   decoding   protocol
    }
}
```

Protocol encoding

The component for protocol encoding derives from io.netty.handler.codec.MessageToByteEncoder<I>. Refer to *MessageToByteEncoder Documentation* for details.

Example:

```
 public   class   SampleEnco  der   extends   MessageToB  yteEncoder <
 String >{
    @ Override
     protected   void   encode ( ChannelHan  dlerContex  t   ctx ,
 String   msg ,  ByteBuf   out )  throws   Exception  {
```

```
//  Protocol   encoding
    }
}
```

Protocol adapter

To reduce cost and improve the efficiency of development, the general protocol server SDK provides protocol adapters with extensible and customizable basic capability class com.aliyun.iot.as.bridge.server.channel.CustomizedTcpBasedProtocolHandler. It encapsulates details to access Alibaba Cloud IoT Platform, so you can focus on protocol related business. The protocol adapter derives from this class.

Device Online

Only online devices can establish a connection with or send connection requests to IoT Platform. There are two steps for devices to get online: local session initialization and device authentication.

1. Session Initialization

   Refer to *Core SDK develop > Device Online > Session Initialization*

2. Device Authentication

   After local session initialization, call doOnline(ChannelHandlerContext ctx, Session newSession, String originalIdentity, String... credentials) to complete local device authentication and Alibaba Cloud IoT Platform online authentication. The device can communicate with IoT Platform if authentication succeeds, and will be disconnect from IoT Platform if authentication fails.

   Example:

   ```
    Session   session  =  Session . newInstanc  e ( device ,   channel
    );
    boolean   success  =  doOnline ( session ,  originalId  entity );
    if  ( success ) {
       //  Successful  ly   got   online ,  and   will   accept
    communicat  ion   requests .
   }  else  {
       //  Failed   to   get   online ,  will   reject   communicat
    ion   requests   and   disconnect  ( if   connected ).
   }
   ```

Device Offline

When the device is disconnected or detects that it needs to be disconnected, the device offline action should be initiated. Using server SDK, devices will automatically get offline when they are disconnected, so you can focus on other tasks. Call doOffline (Session session) to bring devices offline.

Report Data

The protocol adapter needs to use override channelRead(ChannelHandlerContext ctx , Object msg). It is the entrance for all devices to report data. Object msg is the data output from the decoder.

There are three steps for data reporting: identify the device that is going to report data, find the corresponding session for this device, and then report data to IoT Platform. Use the following interfaces to report data:

- CompletableFuture doPublishAsync(Session session, String topic, byte[] payload, int qos): send data asynchronously and return immediately. Developers obtain the sending result using future.
- CompletableFuture doPublishAsync(Session session, ProtocolMessage protocolMsg): send data asynchronously and return immediately. Developers obtain the sending result using future.
- boolean doPublish(Session session, ProtocolMessage protocolMsg, int timeout): send data asynchronously and wait for the response.
- boolean doPublish(Session session, String topic, byte[] payload, int qos, int timeout): send data asynchronously and wait for the response.

Example:

```
 DeviceIden  tity   identity  =  ConfigFact  ory . getDeviceC
 onfigManag  er (). getDevivic  eIdentity ( device );
 if  ( identity  ==  null ) {
    //  Devices   are   not   mapped   with   those   registered   on
   IoT   Platform .  Messages   are   dropped .
     return ;
}
 Session   session  =  SessionMan  agerFactor  y . getInstanc  e ().
 getSession ( device );
 if  ( session  ==  null ) {
    //  The   device   is   not   online .  Please   get   online   or
   drop   messages .  Make   sure   devices   are   online   before
 reporting   data   to   IoT   Platform .
}
 boolean   success  =  doPublish ( session ,  topic ,  payload ,  0 ,
 10 );
 if ( success ) {
    //  Data   is   successful  ly   reported   to   Alibaba   Cloud
 IoT   Platform .
}  else  {
    //  Failed   to   report   data   to   IoT   Platform
}
```

Downstream Messaging

Refer to *Core SDK development > Downstream Messaging* for details.

The SDK provides com.aliyun.iot.as.bridge.core.handler.DefaultDownlinkChannelHandler as the downstream data distributor. It supports unicast and broadcast. Unicast forwards data from the cloud directly to the device by default, and broadcast requires developers to customize specific implementations. Customization can be realized by deriving subclass.

Example:

```
 import   io . netty . channel . Channel ;
 import   Io .  netty .  Channel .  channelfut  ure ;
...

 public   class   SampleDown  linkChanne  lHandler   implements
 DownlinkCh  annelHandl  er  {
    @ Override
     public   boolean   pushToDevi  ce ( Session   session ,  String
 topic ,  byte []  payload ) {
        //  Obtain   communicat  ion   channel   from   device ' s
 correspond  ing   session .
         Channel   channel  = ( Channel )  session . getChannel ().
 get ();
         if  ( channel  ! =  null  &&  channel . isWritable ()) {
             String   body  =  new   String ( payload ,  StandardCh
 arsets . UTF_8 );
            //  Send   downstream   data   to   devices
             ChannelFut  ure   future  =  channel . pipeline ().
 writeAndFl  ush ( body );
             future . addListene  r ( ChannelFut  ureListene  r .
 FIRE_EXCEP  TION_ON_FA  ILURE );
             return   true ;
        }
         return   false ;
    }

    @ Override
     public   boolean   broadcast ( String   topic ,  byte []  payload
 ) {
         throw   new   RuntimeExc  eption (" not   implemente  d ");
    }
}
```

4.3.3 Server SDK

You can use the general protocol server SDK to quickly build a bridge service that connects your existing devices or services to Alibaba Cloud IoT Platform.

Prerequisites

Refer to *Overview* for concepts, functions and Maven dependencies of the general protocol server SDK.

Configuration Management

The general protocol server SDK uses file-based configuration management by default. Add the socketServer parameter in *application.conf*, and set the socket server related parameters listed in the following table. For customized configuration, refer to *Custom Components > Configuration Management* .

Parameter	Description	Required
address	The connection listening address. Supports network names like eth1, and IPv4 addresses with 10.30 prefix.	No
backlog	The number of backlogs for TCP connection.	No
ports:	Connection listening port. The default port is 9123. You can specify multiple ports.	No
listenType	The type of socket server. Can be `udp` or `tcp` . The default value is `tcp` . Case insensitive.	No
broadcastEnabled	Whether UDP broadcasts are supported. Used when `listenType` is `udp` . The default value is true.	No
unsecured	Whether unencrypted TCP connection is supported. Used when listenType is tcp.	No
keyPassword	The certificate store password. Used when listenType is tcp.	No Note: Effective when keyPassword, keyStoreFile, and keyStoreType are all configured. Otherwise, keyPassword does not need to be configured.
keyStoreFile	The file address of the certificate store. Used when listenType is tcp.	No

Parameter	Description	Required
keyStoreType	The type of certificate store. Used when listenType is tcp.	No

Interfaces

The following two articles assume that you have a basic understanding of Netty-based development. Refer to *Netty Documentation* for more details on Netty-based development.

- *Interfaces for TCP*
- *Interfaces for UDP*

Custom Components

Besides file-based configuration, you can also set your own customized configurat ions.

If you want to customize configurations, implement com.aliyun.iot.as.bridge.server .config.BridgeServerConfigManager first and call com.aliyun.iot.as.bridge.server. config.ServerConfigFactory.init(BridgeServerConfigManager bcm) to replace default configuration management components with customized ones, and then initialize these components. Then, connect the net bridge products to the Internet.

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 need 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: The remote synchronous process call.
- 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, 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:

```
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 . 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 );
```

```
RRpcRespon  se   response  =  client . getAcsResp  onse ( 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:

```
mqttClient  Id :  clientId +"| securemode = 3 , signmethod =
hmacsha1 , timestamp = 132323232 |"
```

   After ext=1 is added to the clientId, the clientId that the device sends is as follows:

```
mqttClient  Id :  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 shadows

A device shadow is a JSON file that is used to store the reported status and the desired status of the device.

- Each device only has one device shadow. The device gets and sets the device shadow based on Message Queuing Telemetry Transport (MQTT). Therefore, the device shadow status and the device status can synchronize.
- The application uses the SDK of IoT Platform to get and set the device shadow . Then, the application can obtain the latest device status from and deliver the desired status to the target device by using the device shadow.

Scenario 1

A device frequently disconnects from and reconnects to IoT Platform. This is caused by unstable network conditions. The application cannot obtain the device status when requesting the status from an offline device, and fails to send another device status request when the device is reconnected.

The device shadow can synchronize with the device to update and store the latest device status. Therefore, the application can obtain the current device status from the device shadow of an offline or online device.

Scenario 2

A device has to respond to each status request when multiple applications request the status of this device in stable network conditions. Even if the responses are the same, the device may be overloaded when processing these requests.

On IoT Platform, the device synchronizes the status to the device shadow only. Applications can request the latest device status from the device shadow, instead of the target device. Therefore, applications are decoupled from the device.

Scenario 3

- A device frequently disconnects from and reconnects to IoT Platform. This is caused by unstable network conditions. A device that is in offline status cannot receive application commands.
  - Quality of Service 1 or 2 (QoS 1 or 2) may solve this issue. However, we do not recommend that you use this method. This method increases the workload of the service.
  - On IoT Platform, the device shadow stores the control commands that contain the timestamps when the application sends these commands. The device obtains these commands and checks their timestamps to determine whether to execute the commands when the device has reconnected to IoT Platform.
- A device in offline status cannot receive the commands from the application. When the connection has recovered, the device executes valid commands by checking the timestamps of the device shadow commands.

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.
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 10000 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 / 10000 / 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 `method` is required and must be set to `update` .

Parameter	Description
`state`	The status information that the device sends to the device shadow. The `reported` 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 / 10000 / 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 ": "${ errorcode }",
" errormessa  ge ": "${ errormessa  ge }"
}
},
" 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 / 10000 / 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 / 10000 / 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 / 10000 / 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 client sends a message to the topic `/ shadow / update / 10000 / 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 / 10000 / 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 / 10000 / 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 / 10000 / 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 Accounts and logon

This topic describes IoT Platform accounts and how to log on to the IoT Platform console.

7.1 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, seeUse RAM users and Custom permissions.

7.2 Resource Access Management (RAM)

This chapter describes IoT Platform access control.

7.2.1 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*.

7.2.2 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 : SecureTran  sport ": " 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 ": {
" DateLessTh  an ": {
" acs : CurrentTim  e ": " 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 "
]
},
" DateLessTh  an ": {
" acs : CurrentTim  e ": " 2019 - 01 - 01T00 : 00 : 00 + 08 : 00 "
},
" Bool ": {
" acs : SecureTran  sport ": " 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*.

7.2.3 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.

7.2.4 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.

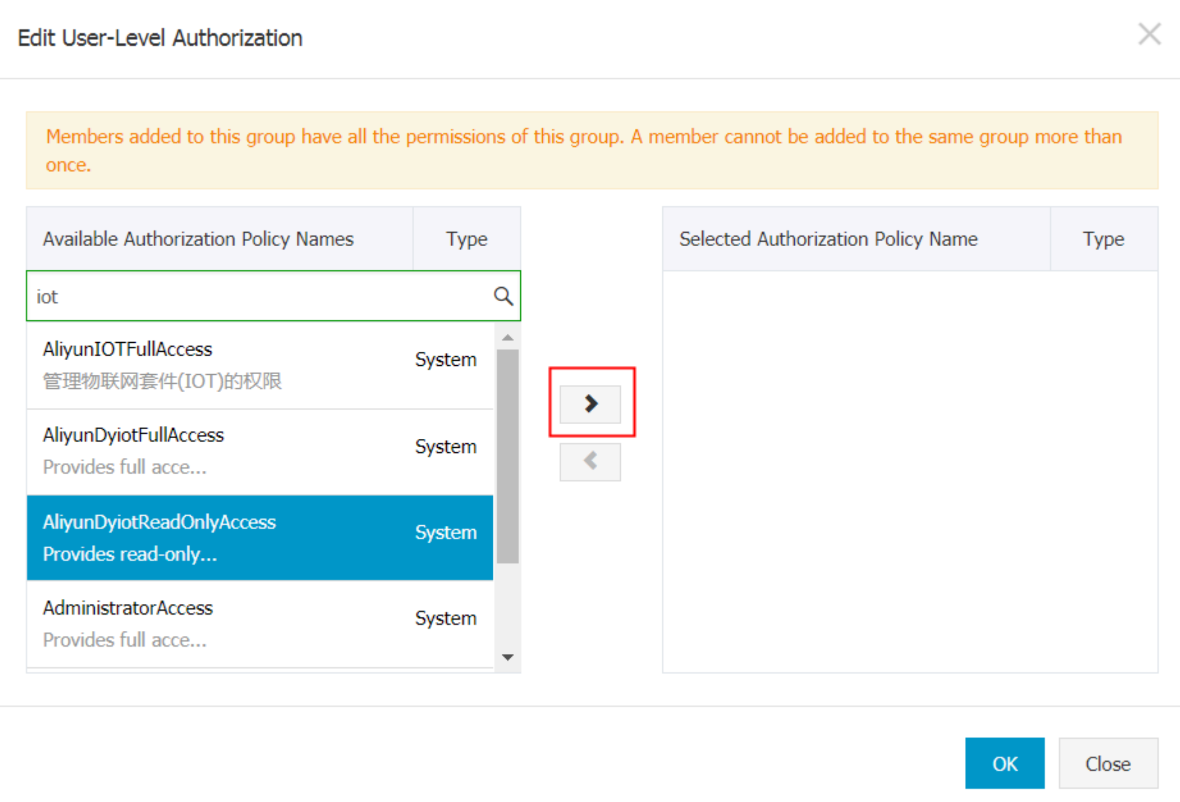
Authorize the RAM user to access IoT Platform

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
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

Selected Authorization Policy Name	Type

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.

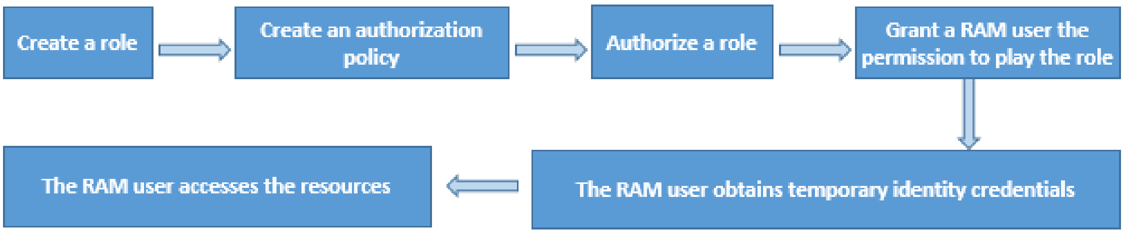
Then, the RAM user can perform authorized operations in the console.

7.2.5 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 7-1: Process for granting temporary permissions to RAM users.


Create a role
Create an authorization policy
Authorize a role
Grant a RAM user the permission to play the role
The RAM user obtains temporary identity credentials
The RAM user accesses the resources

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 : 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 : Query *",
" iot : List *",
" iot : Get *",
" iot : BatchGet *"
],
" Resource ": "*"
}
]
}
```

Authorization policy example: Read-write 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 : 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 ": "*"
}
]
}
```

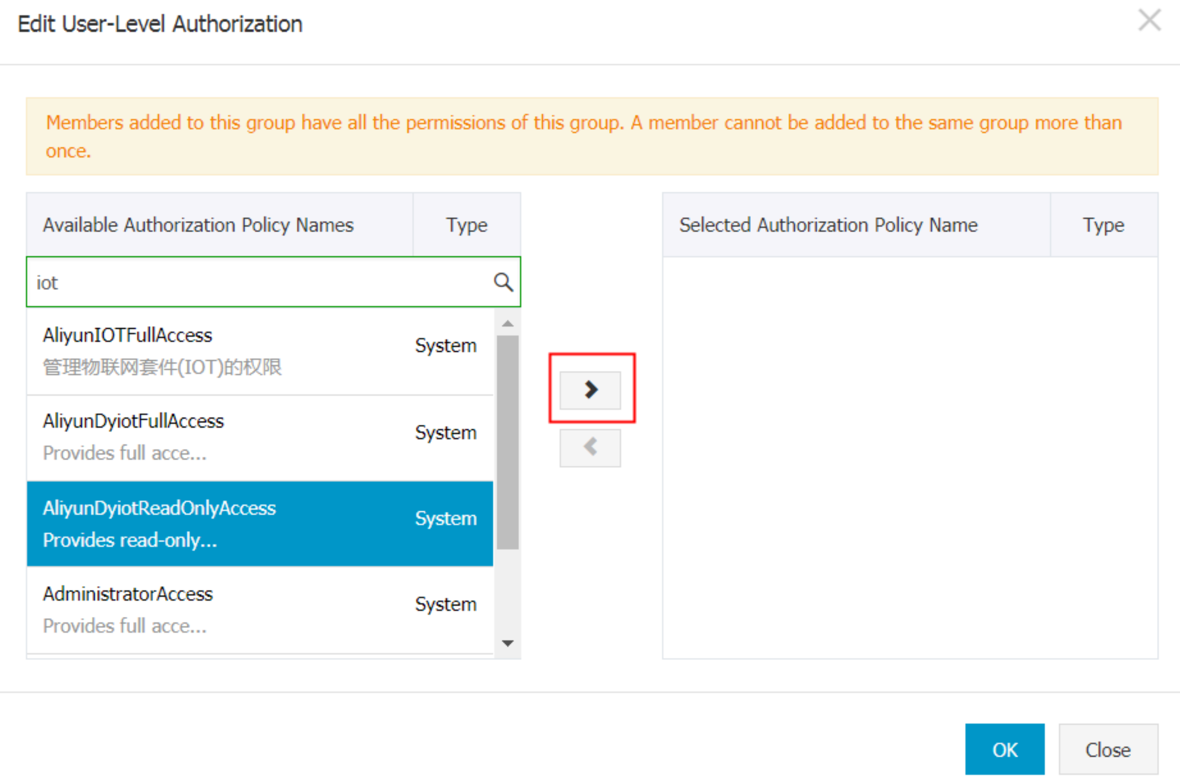
After an authorization policy has been created, you can grant the permissions defined in this policy to roles.

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
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

Selected Authorization Policy Name	Type

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  123456 : role / iotstsrole
& RoleSessio  nName = iotreadonl  yrole
& DurationSe  conds = 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  123456 : role / iotstsrole   RoleSessio  nName =
 iotreadonl  yrole   Policy ='{" Version ":" 1 "," Statement ":
[{" Effect ":" Allow "," Action ":" iot :*"," Resource ":"*"}]}'
 DurationSe  conds = 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.

```
IClientPro  file   profile  =  DefaultPro  file . getProfile (" cn -
hangzhou ",  AccessKeyI  d , AccessSecr  et );
RpcAcsRequ  est   request . putQueryPa  rameter (" SecurityTo  ken
",  Token );
IAcsClient   client  =  new   DefaultAcs  Client ( profile );
AcsRespons  e   response  =  client . getAcsResp  onse ( request );
```