Alibaba Cloud IoT Platform

User Guide

Issue: 20181113

Legal disclaimer

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

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

secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).

6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 **Danger:** Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 **Note:** 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]`*
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *`{stand | slave}`*

Contents

1 Accounts and logon

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

1.1 Log on to the console using the primary account

The primary account has full operation permissions on all resources under this account, and supports modifying account information.

Log on to the IoT Platform console using the primary account

You have full operation permissions on IoT Platform when logging on to the console using the primary account.

1. Visit the *Alibaba Cloud official website*.
2. Click **Console**.
3. Log on to the console using your account and password.

   Note:
   To retrieve an account or password, click **Forgot Username** or **Forgot Password** on the logon page to start the retrieval process.
4. Click **Products** in the console to display all products and services that are provided by Alibaba Cloud.
5. Search for IoT Platform, and click **IoT Platform** in the result to enter the **IoT Platform console**.

   Note:
   If you have not activated the IoT Platform service, the **IoT Platform console** prompts you to activate this service on the homepage. Click **Activate Now** to activate it quickly.

After entering the **IoT Platform console**, you can manage products, devices, and rules.

Create access control using the primary account

The primary account has full permissions, so the leakage of the primary account may cause serious security risks. Therefore, do not disclose your account and password when you authorize others to access your Alibaba Cloud resources. Instead, you should use Resource Access Management (RAM) to create sub-accounts and assign the required access permissions to these sub-accounts. All users except the primary account user or administrator access the resources using sub-accounts. For more information about accessing IoT Platform using RAM users, see*Use RAM users* and *Custom permissions*.

1.2 Resource Access Management (RAM)

This chapter describes IoT Platform access control.

1.2.1 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 *Cite LeftExamplesCite Right*.

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*.

1.2.2 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:CreateProduct"
```

- Define multiple API operations.

```
"Action": [
"iot:UpdateProduct",
"iot:QueryProduct"
]
```

- Define all read-only API operations.

```
{
"Version": "1",
"Statement": [
{
"Action": [
"rds:DescribeDBInstances",
"rds:DescribeDatabases",
"rds:DescribeAccounts",
"rds:DescribeDBInstanceNetInfo"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": "ram:ListRoles",
"Effect": "Allow",
"Resource": "*"
},
{
"Action": [
"mns:ListTopic"
```

```
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"dhs:ListProject",
"dhs:ListTopic",
"dhs:GetTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"ots:ListInstance",
"ots:ListTable",
"ots:DescribeTable"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"log:ListShards",
"log:ListLogStores",
"log:ListProject"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Effect": "Allow",
"Action": [
"iot:Query*",
"iot:List*",
"iot:Get*",
"iot:BatchGet*"
],
"Resource": "*"
}
]
}
```

- Define all read-write API operations.

```
{
"Version": "1",
"Statement": [
{
"Action": [
"rds:DescribeDBInstances",
"rds:DescribeDatabases",
"rds:DescribeAccounts",
"rds:DescribeDBInstanceNetInfo"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": "ram:ListRoles",
"Effect": "Allow",
```

```
"Resource": "*"
},
{
"Action": [
"mns:ListTopic",
"mns:CreateQueue"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"dhs:ListProject",
"dhs:ListTopic",
"dhs:GetTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"ots:ListInstance",
"ots:ListTable",
"ots:DescribeTable"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"log:ListShards",
"log:ListLogStores",
"log:ListProject"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*"
}
]
}
```

Define conditions

RAM authorization policies currently support multiple authentication conditions, such as the access IP address restrictions, the Hypertext Transfer Protocol Secure (HTTPS)-based access enabler, the multi-factor authentication (MFA)-based access enabler, and access time restrictions . All API operations on IoT Platform support these authentication conditions.

Access control based on source IP addresses

This access control restricts source IP addresses that can access IoT Platform, and supports filtering by Classless Inter-Domain Routing (CIDR) blocks. Typical scenarios are described as follows:

- Apply access control rules to a single IP address or CIDR blocks. For example, the following code indicates that only access requests from IP address 10.101.168.111 or 10.101.169.111/24 are allowed.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*",
"Condition": {
"IpAddress": {
"acs:SourceIp": [
"10.101.168.111",
"10.101.169.111/24"
]
}
}
}
],
"Version": "1"
}
```

- Apply access control rules to multiple IP addresses. For example, the following code indicates that only access requests from IP addresses 10.101.168.111 and 10.101.169.111 are allowed.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*",
"Condition": {
"IPaddress ":{
"acs:SourceIp": [
"10.101.168.111",
"10.101.169.111"
]
}
}
}
],
"Version": "1"
}
```

HTTPS-based access control

This access control allows you to enable or disable HTTPS-based access.

For example, the following code indicates that only HTTPS-based access is allowed.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
```

```
"Resource": "*",
"Condition": {
"Bool": {
"acs:SecureTransport": "true"
}
}
}
],
"Version": "1"
}
```

MFA-based access control

This access control allows you to enable or disable MFA-based access.

For example, the following code indicates that only MFA-based access is allowed.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*",
"Condition": {
"Bool": {
"acs:MFAPresent ": "true"
}
}
}
],
"Version": "1"
}
```

Access time restrictions

This access control allows you to limit the access time of requests. Access requests earlier than the specified time are allowed or rejected.

For example, the following code indicates that only access requests earlier than 00:00:00 Beijing Time (UTC+8) on January 1, 2019 are allowed.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*",
"Condition": {
"DateLessThan": {
"acs:CurrentTime": "2019-01-01T00:00:00+08:00"
}
}
}
],
"Version": "1"
```

```
}
```

Typical scenarios

Based on these definitions of actions, resources, and conditions, authorization policies are described in the following typical scenarios.

The following is an example of authorization policy that allows access.

Scenario: Assigns IoT Platform access permissions to the IP address 10.101.168.111/24, and only allows HTTPS-based access before 00:00:00 Beijing Time (UTC+8) on January 1, 2019.

```
{
"Statement": [
{
"Effect": "Allow",
"Action": "iot:*",
"Resource": "*",
"Condition": {
"IPaddress ":{
"acs:SourceIp": [
"10.101.168.111/24"
]
},
"DateLessThan": {
"acs:CurrentTime": "2019-01-01T00:00:00+08:00"
},
"Bool": {
"acs:SecureTransport": "true"
}
}
}
],
"Version": "1"
}
```

The following is an example of authorization policy to specify denied access.

Scenario: Rejects read requests from IP address 10.101.169.111.

```
{
"Statement": [
{
"Effect": "Deny",
"Action": [
"iot:Query*",
"iot:List*",
"iot:Get*",
"iot:BatchGet*"
],
"Resource": "*",
"Condition": {
"IpAddress": {
"acs:SourceIp": [
"10.101.169.111"
]
}
}
```

```
}
],
"Version": "1"
}
```

After creating the authorization policy, apply this policy to the RAM users on the **User Management** page in the RAM console. Authorized RAM users can perform the operations defined in this policy. For more information about creating RAM users and granting permissions, see *Use RAM users*.

1.2.3 API permissions

Each operation in the following table represents the value of Action that you specify when creating authentication policies for RAM users.

Operation	RAM action	Resource	Description
CreateProduct	iot:CreateProduct	*	Create a product.
UpdateProduct	iot:UpdateProduct	*	Modify the information of a product.
QueryProduct	iot:QueryProduct	*	Query the detailed informatio n of a product.
QueryProductList	iot:QueryProductList	*	Query the list of all products.
DeleteProduct	iot:DeleteProduct	*	Delete a product.
RegisterDevice	iot:RegisterDevice	*	Register a device.
QueryDevice	iot:QueryDevice	*	Query the list of all 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	*	Get the status of multiple devices at a time.
BatchRegisterDeviceW ithApplyId	iot:BatchRegisterDeviceW ithApplyId	*	Register multiple devices simultaneously using a given application ID.
BatchRegisterDevice	iot:BatchRegisterDevice	*	Register devices at a time (not specify device names).

Operation	RAM action	Resource	Description
QueryBatchRegisterDeviceStatus	iot:QueryBatchRegisterDeviceStatus	*	Query the processing status and result of device registration of multiple devices.
BatchCheck DeviceNames	iot:BatchCheck DeviceNames	*	Specify device names in batch.
QueryDeviceStatistics	iot:QueryDeviceStatistics	*	Query device statistics.
QueryDeviceEventData	iot:QueryDeviceEventData	*	Query historical events of a device.
QueryDeviceServiceData	iot:QueryDeviceServiceData	*	Query historical servicing records of a device.
SetDeviceProperty	iot:SetDeviceProperty	*	Configure properties of a device.
InvokeThingService	iot:InvokeThingService	*	Invoke a service of a device.
QueryDevicePropertyStatus	iot:QueryDevicePropertyStatus	*	Query the property information of a device.
QueryDeviceDetail	iot:QueryDeviceDetail	*	Query details about 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.
NotifyAddThingTopo	iot:NotifyAddThingTopo	*	Notify a gateway device to add topological relationships of the connected sub-devices.
QueryDevicePropertyData	iot:QueryDevicePropertyData	*	Query historical property records of a device.
GetGatewayBySubDevice	iot:GetGateway BySubDevice	*	Query the gateway device information using the sub-device information.
SaveDeviceProp	iot:SaveDeviceProp	*	Create a tag for a device.

Operation	RAM action	Resource	Description
QueryDeviceProp	iot:QueryDeviceProp	*	Query the tag lists of a device.
DeleteDeviceProp	iot:DeleteDeviceProp	*	Delete a tag of a device.
QueryAppDeviceList	iot:QueryAppDeviceList	*	Query the list of devices that are bound with an specified app.
StartRule	iot:StartRule	*	Enable a rule.
StopRule	iot:StopRule	*	Stop a rule.
ListRule	iot:ListRule	*	Query the information of all rules.
GetRule	iot:GetRule	*	Query the details of a rule.
CreateRule	iot:CreateRule	*	Create a rule.
UpdateRule	iot:UpdateRule	*	Modify a rule.
DeleteRule	iot:DeleteRule	*	Delete a rule.
CreateRuleAction	iot:CreateRuleAction	*	Create a data forwarding method for a rule.
UpdateRuleAction	iot:UpdateRuleAction	*	Modify a data forwarding method.
DeleteRuleAction	iot:DeleteRuleAction	*	Delete a data forwarding method.
GetRuleAction	iot:GetRuleAction	*	Query the detailed informatio n of data forwarding method
ListRuleActions	iot:ListRuleActions	*	Query the list of data forwarding methods in a rule.
Pub	iot:Pub	*	Publish messages.
PubBroadcast	iot:PubBroadcast	*	Publish messages to the devices that have subscribed to a broadcast topic.
RRpc	iot:RRpc	*	Send a request to a device and retrieve a response from the device.
CreateProductTopic	iot:CreateProductTopic	*	Create a topic category for a product.

Operation	RAM action	Resource	Description
DeleteProductTopic	iot:DeleteProductTopic	*	Delete a topic category.
QueryProductTopic	iot:QueryProductTopic	*	Query information about all topic categories of a product.
UpdateProductTopic	iot:UpdateProductTopic	*	Modify a topic category.
CreateTopicRouteTable	iot:CreateTopicRouteTable	*	Create message routing relationships between topics.
DeleteTopicRouteTable	iot:DeleteTopicRouteTable	*	Delete message routing relationships.
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	*	Modify the shadow information of a device.

1.2.4 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 permission s to the RAM user. Therefore, you need to assign permissions for accessing IoT Platform to this RAM user.

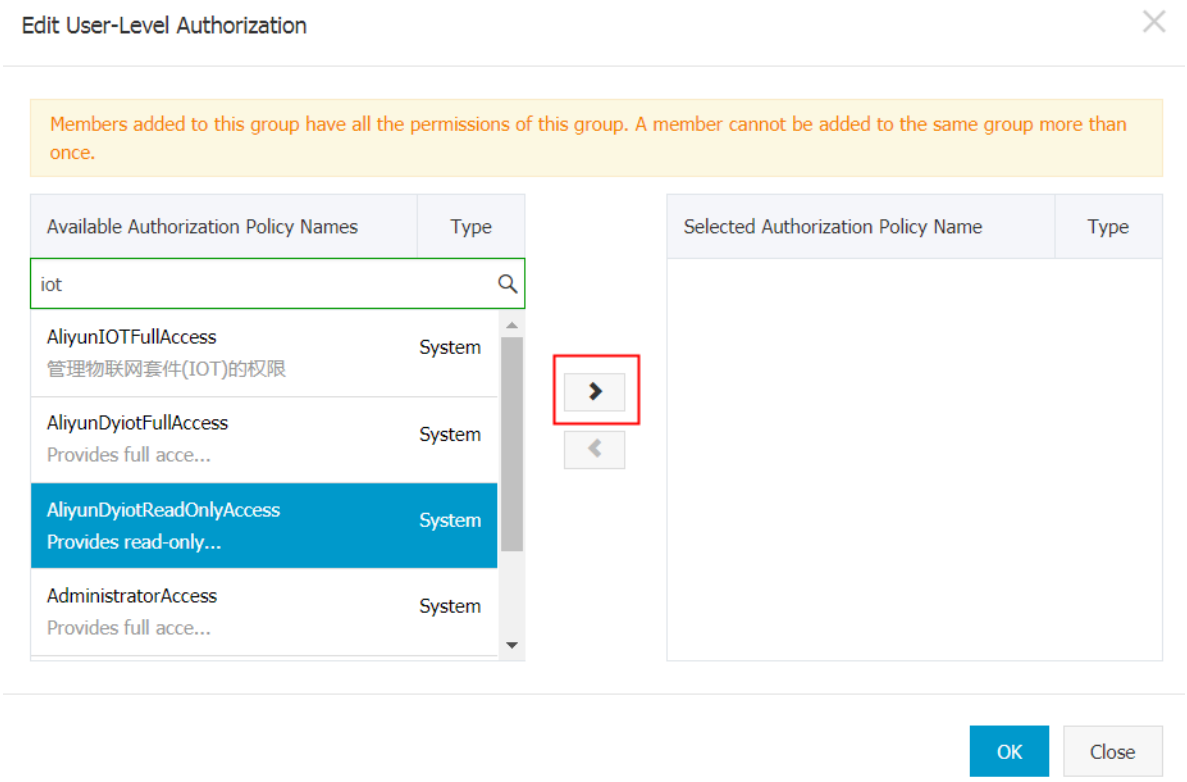
Authorize the RAM user to access IoT Platform

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*.



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**.

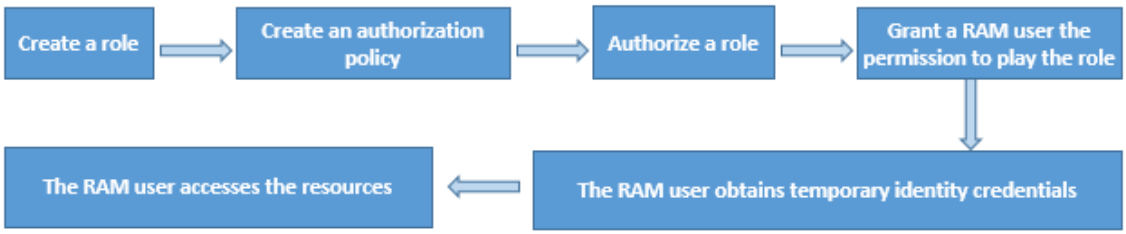
Then, the RAM user can perform authorized operations in the console.

1.2.5 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 informatio n of a RAM user has been leaked, if you fail to timely delete this user or revoke related permission s, 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 1-1: Process for granting temporary permissions to RAM users.



Step 1: Create a role

A role is a virtual entity that represents a virtual user with a group of permissions.

1. Log on to the *RAM console*.
2. Select **Roles** > **Create Role** to create a role.
3. Select **User Role**.
4. Use the default account information, and click **Next**.
5. Specify the role name and description, and click **Create**.
6. Click **Close** or **Authorize**.

If you have created the authorization policy that is to be granted to this role, click **Authorize** to authorize this user.

If you have not created the authorization policy, click **Close**. You can create an authorization policy for this role by clicking **Policies**.

Step 2: Create an authorization policy

An authorization policy defines the resource permissions that are to be granted to roles.

1. In the *RAM console*, click **Policies** > **Create Authorization Policy** .
2. Select the blank template.
3. Specify the authorization policy name and policy content, and click **Create Authorization Policy**.

   For more information about writing the policy content, click **Authorization Policy Format**.

   Authorization policy example: Read-only permission of IoT resources.

```
{
"Version": "1",
"Statement": [
{
"Action": [
"rds:DescribeDBInstances",
"rds:DescribeDatabases",
"rds:DescribeAccounts",
"rds:DescribeDBInstanceNetInfo"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": "ram:ListRoles",
"Effect": "Allow",
"Resource": "*"
},
{
"Action":[
"mns:ListTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"dhs:ListProject",
"dhs:ListTopic",
"dhs:GetTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
```

```
"Action": [
"ots:ListInstance",
"ots:ListTable",
"ots:DescribeTable"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action":[
"log:ListShards",
"log:ListLogStores",
"log:ListProject"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Effect": "Allow",
"Action": [
"iot:Query*",
"iot:List*",
"iot:Get*",
"iot:BatchGet*"
],
"Resource": "*"
}
]
}
```

Authorization policy example: Read-write permission of IoT resources.

```
{
"Version": "1",
"Statement": [
{
"Action": [
"rds:DescribeDBInstances",
"rds:DescribeDatabases",
"rds:DescribeAccounts",
"rds:DescribeDBInstanceNetInfo"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": "ram:ListRoles",
"Effect": "Allow",
"Resource": "*"
},
{
"Action":[
"mns:ListTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"dhs:ListProject",
```

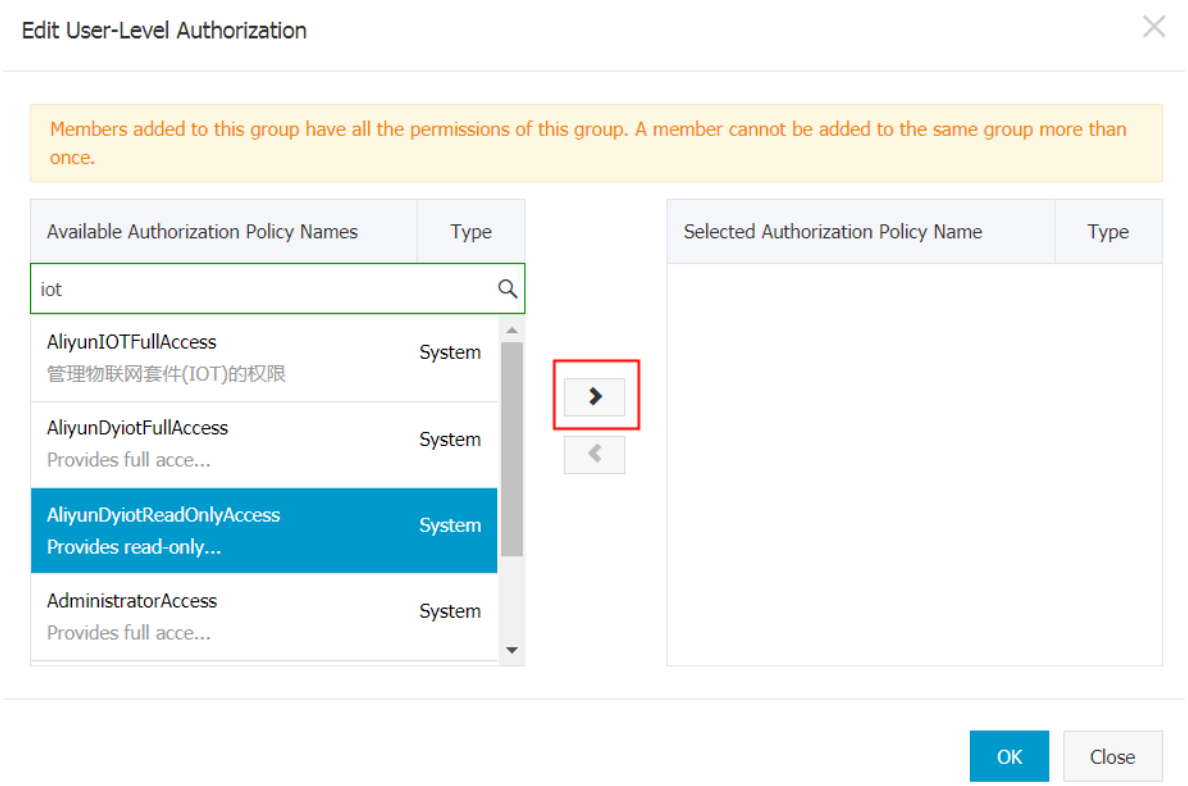
```
"dhs:ListTopic",
"dhs:GetTopic"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action": [
"ots:ListInstance",
"ots:ListTable",
"ots:DescribeTable"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Action":[
"log:ListShards",
"log:ListLogStores",
"log:ListProject"
],
"Resource": "*",
"Effect": "Allow"
},
{
"Effect": "Allow",
"Action": "iot:*",
"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**.



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 authorizat ion 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:QueryProduct",
"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::1234567890123456:role/iotstsrole
&RoleSessionName=iotreadonlyrole
&DurationSeconds=3600
&Policy=<url_encoded_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::1234567890123456:role/
iotstsrole RoleSessionName=iotreadonlyrole Policy='{"Version":"1","
Statement":[{"Effect":"Allow","Action":"iot:*","Resource":"*"}]}'
DurationSeconds=3600 --id=id --secret=secret
```

After the request has been received, the temporary identity credentials that are required to play the role are returned. The credentials include an AccessKeyId, AccessKeySecret, and SecurityToken.

Step 6: The RAM user accesses the resources

After obtaining the temporary identity credentials, the RAM user can pass in the credentials in the SDK requests to play the specified role.

Java SDK example: The RAM user passes in the AccessKeyId, AccessKeySecret, and SecurityToken parameters that are contained in the temporary identity credentials in the request and creates the IAcsClient object.

```
IClientProfile profile = DefaultProfile.getProfile("cn-hangzhou",
AccessKeyId,AccessSecret);
RpcAcsRequest request.putQueryParameter("SecurityToken", Token);
IAcsClient client = new DefaultAcsClient(profile);
AcsResponse response = client.getAcsResponse(request);
```

2 Create products and devices

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

2.1 Create a product (Basic Edition)

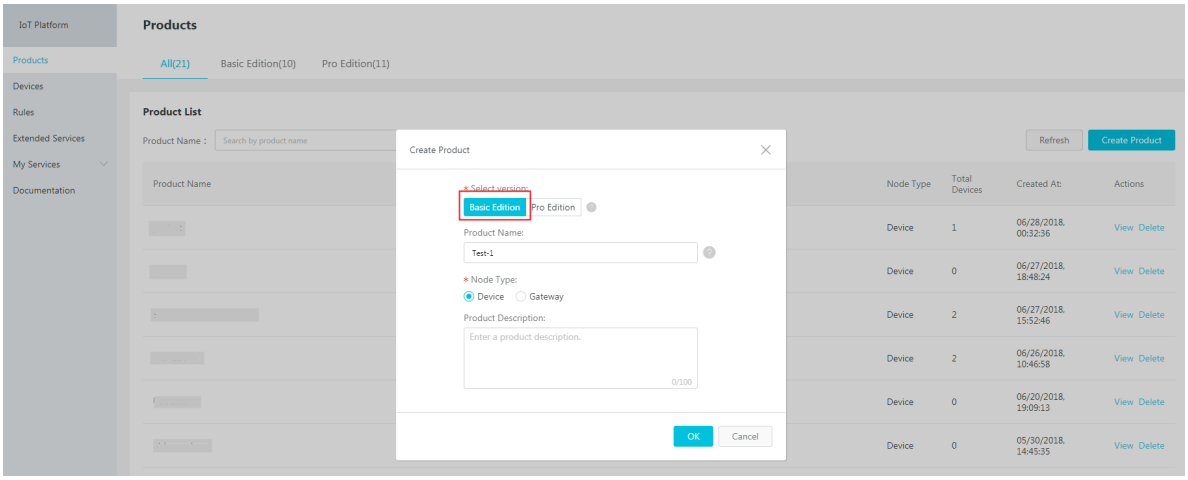
The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is a specific device of the product model.

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 **Products**, and then click **Create Product**.
3. In the dialog box that appears, select **Basic Edition**, enter a valid product name, choose a node type, and then click **OK**.



Descriptions:

- Product Name: Enter a name for your product. The product name will act as a unique identifier. For example, you can enter the product model as the product name.
- Node Type: Select either a device or gateway.
  - Device: This type of device can be connected directly to the IoT Hub, or attached as a sub-device to a gateway.

- Gateway: A device that connects directly to the IoT Hub and can attach sub-devices. A gateway can manage sub-devices, maintain the topological relationship with sub-devices , and synchronize the topological relationship to the IoT Hub.

Result

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

2.2 Create a product (Pro Edition)

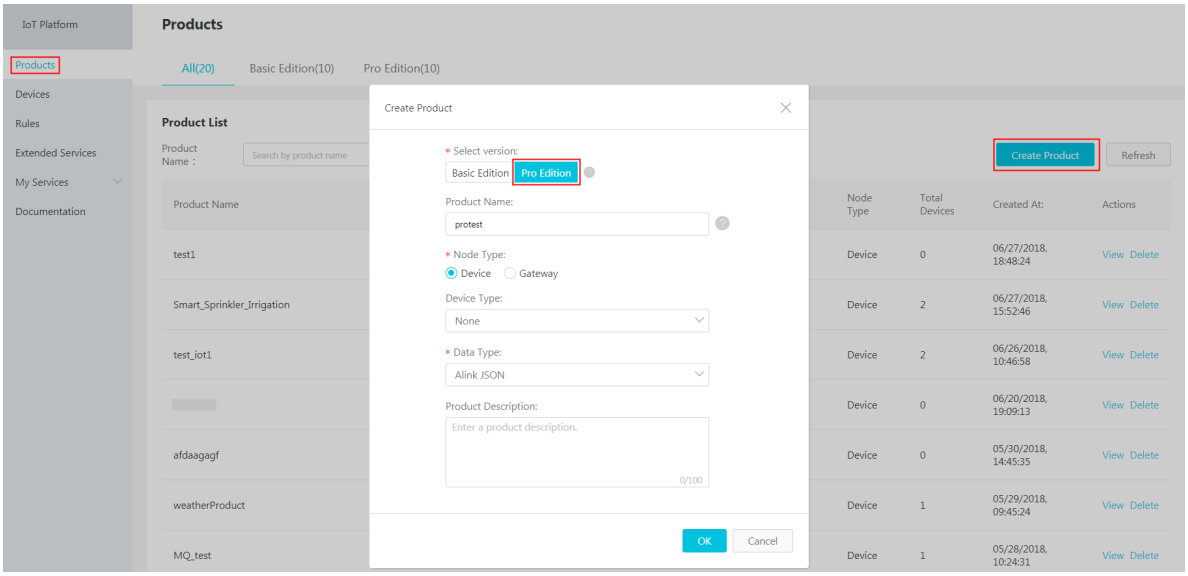
The first step in using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

Context

IoT Platform supports two editions of products: Basic Edition and Pro Edition. This article introduces how to create 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 **Products** and then click **Create Product**.
3. Select **Pro Edition** as the version, enter all the required information, and then click **OK**.



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 Alibaba Cloud account. For example, you can enter the product model as the product name.
Node Type	• Device: A device that cannot attach sub-devices. This type of device can be connected directly to the IoT Hub, or attached as sub-device to a gateway. • Gateway: A device that connects directly to the IoT Hub. Sub-devices can be attached to a gateway. A gateway can manage sub-devices, maintain the topological relationship with sub-devices , and synchronize the topological relationship to the cloud. For more information about gateway devices and sub-devices, see *Gateways and sub-devices*.
Device Type	Select a standard template with predefined features. If you select **None**, no standard template is created and you must manually define features for the product. **Note:** Alibaba Cloud IoT Platform provides various templates with predefined features. For example, the **Electric Meter** template contains the following predefined features: power usage, voltage, electric current, total power consumption and other standard features. You can also customize the template by editing the features or adding additional user-defined features.
Data Type	The format of the uploaded or downloaded device data. You can select either **Alink JSON** or **Do not parse/Custom**. • Alink JSON: A data exchange protocol between devices and the IoT Hub. It is provided in JSON format. • Do not parse/Custom: If you want to customize the serial data format, select **Do not parse/Custom**. You must then convert the user-defined formatted data to Alink JSON script by using data parsing function so that your devices can communicate with the IoT Hub.
Connect to Gateway **Note:**	Will this product connect to a gateway product as its sub-device? • **Yes**: This product can be connect to a gateway.

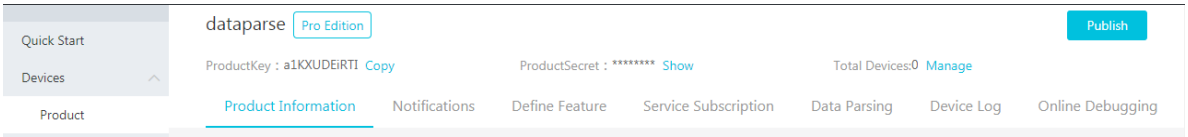
Parameter	Description
Required when the node type is **Device**.	Select a protocol used for sub-device and gateway communication. • Modbus: Indicates the connection protocol for sub-device and gateway communication is Modbus. • OPC UA: Indicates the connection protocol for sub-device and gateway communication is OPC UA. • Custom: Indicates that you want to use another protocol as the connection protocol for sub-device and gateway communication . • **No**: This product cannot be connect to a gateway.
Product Description	Describe the product information.

4. Click **OK**.

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

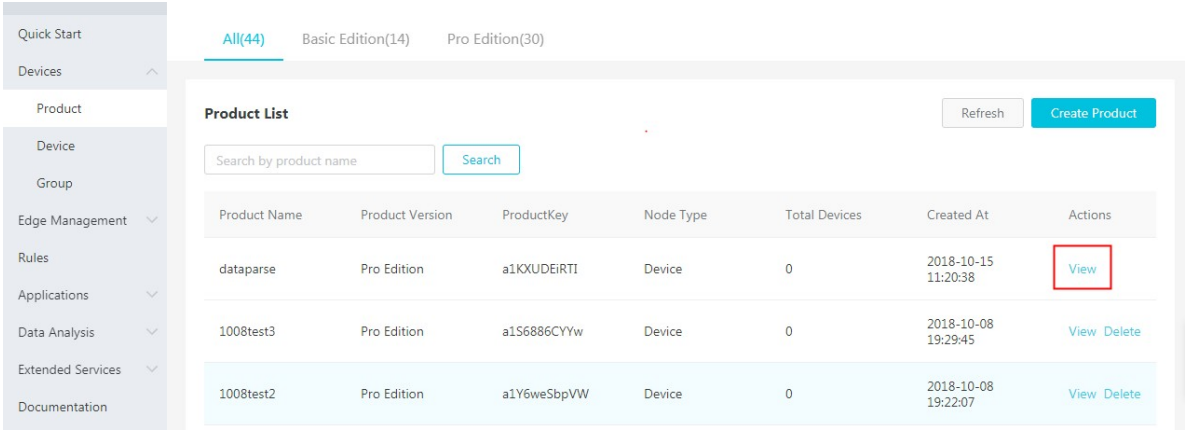
What's next

1. To configure a product's features (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. To learn more about configurations you can apply during device development, see *Developer Guide (Devices)*.
3. To publish a product, go to the product details page and click **Publish**.



Note that before you publish a product make sure that you have configured all the correct information for the product, have completed debugging of it, 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.



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

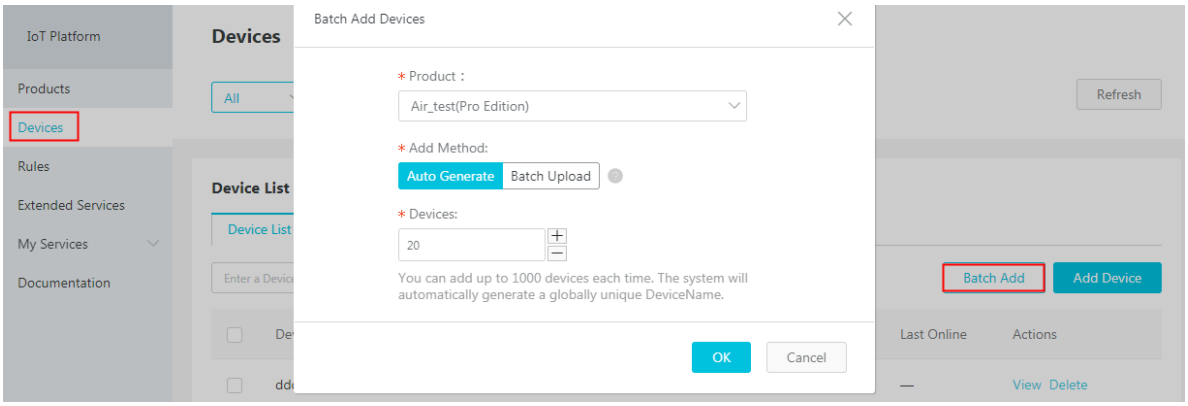
2.3 Create devices

2.3.1 Create multiple devices at a time

A product is a collection of devices. After you create products, you create specific devices of the product models. You can create one device or multiple devices at a time. This article introduces 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** > **Batch Add**.
3. Select a product that you have created. All the devices to be created will be assigned with the features of the selected product.
4. Select one of the following methods for adding device names.
   - Auto Generate: You do not need to specify names for the devices to be created. You only specify the number of devices you want to create, and the system will generate a name for each device.
   - Batch Upload: You must specify names for the devices to be created. Click Download.csv Template to download the naming template. Enter names for the devices in the template file , and then click Upload File to upload the file.



5. Click **OK**.
6. Click **Download Device Certificate**.

Result

After the devices are successfully created, on the **Batch Management** page, you can:

- Click **View Details** to view the detailed information of the devices.
- Click **Download CSV** to download the certificates of the devices.

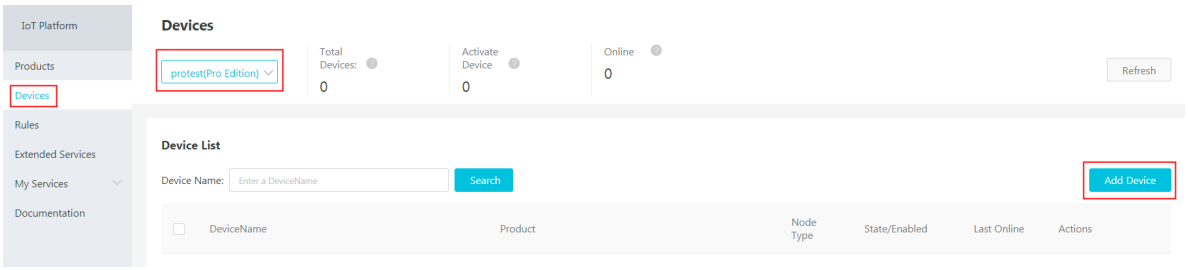
2.3.2 Create a device

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 a batch of devices at a time. This topic 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** > **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 as the device identifier.

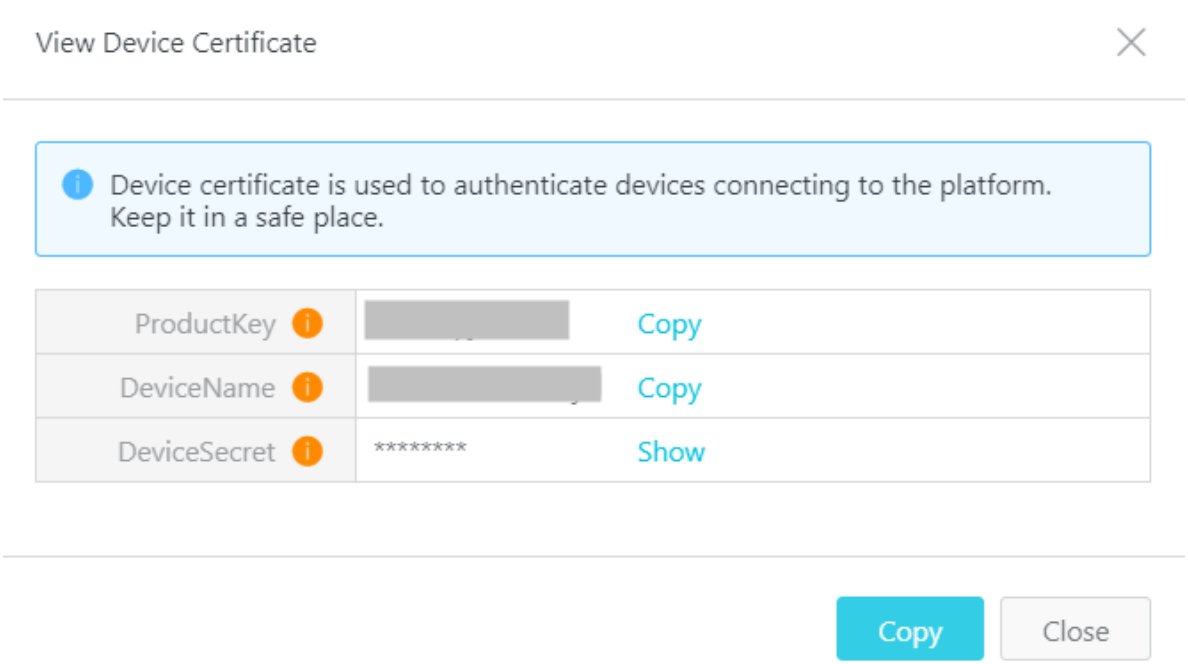
> **Note:**
> A DeviceName (device name) must be unique within a product. It is used as a device identifier for communications with the IoT Hub.



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 authentica tion encryption and must be used in pairs with the DeviceName.



2.4 TSL

2.4.1 What is Thing Specification Language (TSL)?

Thing Specification Language (TSL) is a data model that digitizes a physical entity and constructs the entity in the Cloud. On the IoT platform, a TSL model refers to a set of product features. After the features have been defined 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.

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

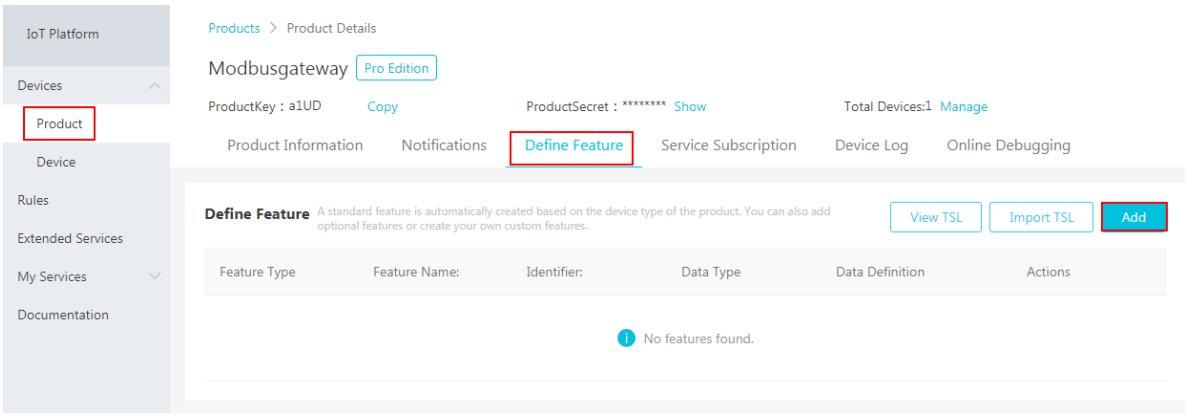
Feature Type	Description
Properties	Describes the running status of a device, such as the current temperature read by the environmental monitoring equipment. Properties support GET and SET request methods. Application systems can send requests to retrieve and set properties.
Services	Capabilities or methods of a device that are exposed and can be used by an external requester. You can set the input and output parameters. Compared with properties, services can use instructions to implement more complex business logic, such as a specific task.
Events	Events generated during operation. Events typically contain notifications that require action or attention, and they may contain multiple output parameters. For example, an event may be a notification about the completion of a task, a system failure, or a temperature alert. You can subscribe to or push events.

2.4.2 Define features using TSL

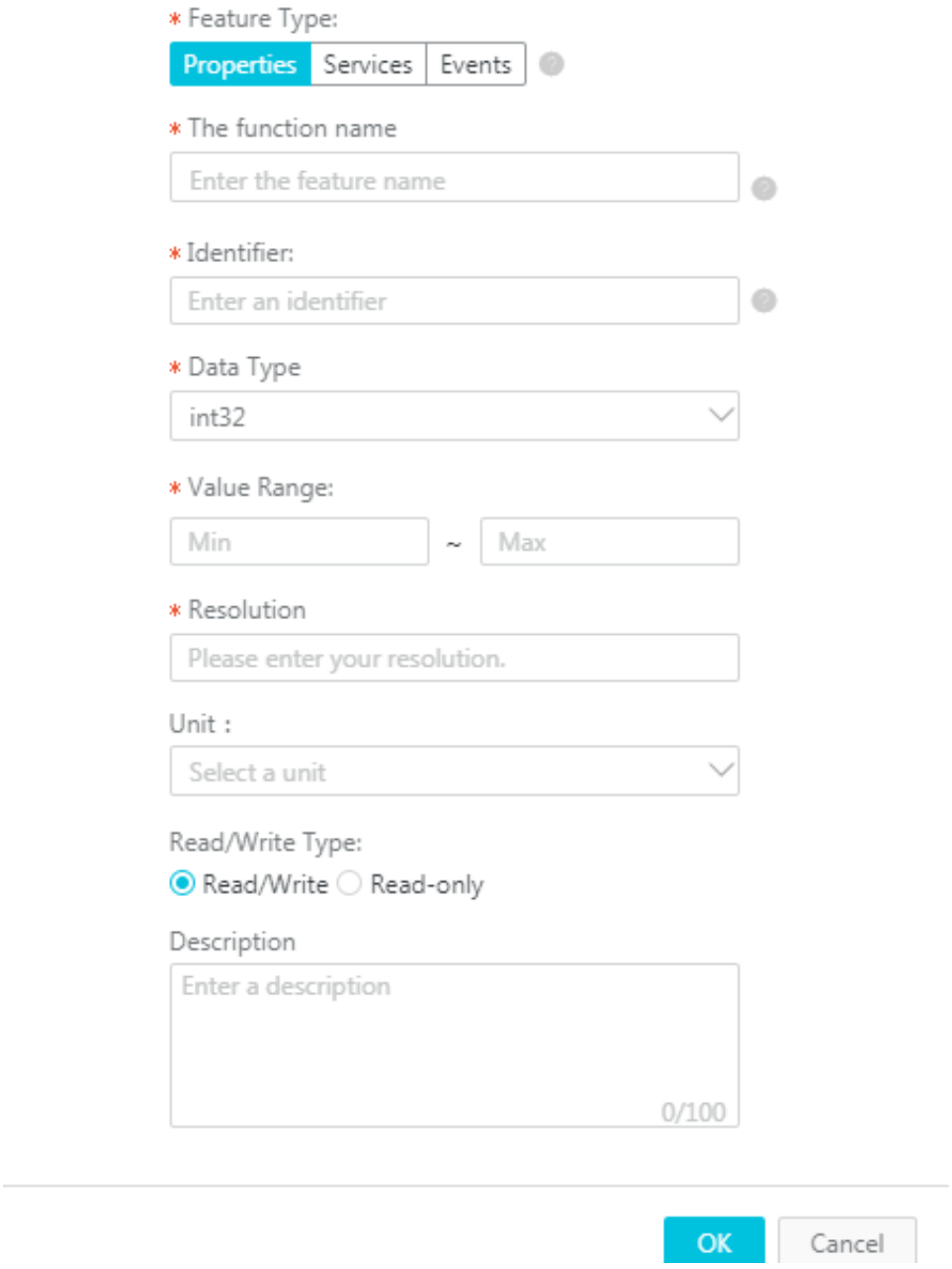
This article introduce how to define features in the IoT Platform console.

Procedure

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click **Products**, find the Pro Edition product for which you want to add features, and then click **View** next to it.
3. Click **Define Feature** > **Add**.



4. In the **Add Feature** dialog box, select **Properties** as the feature type.



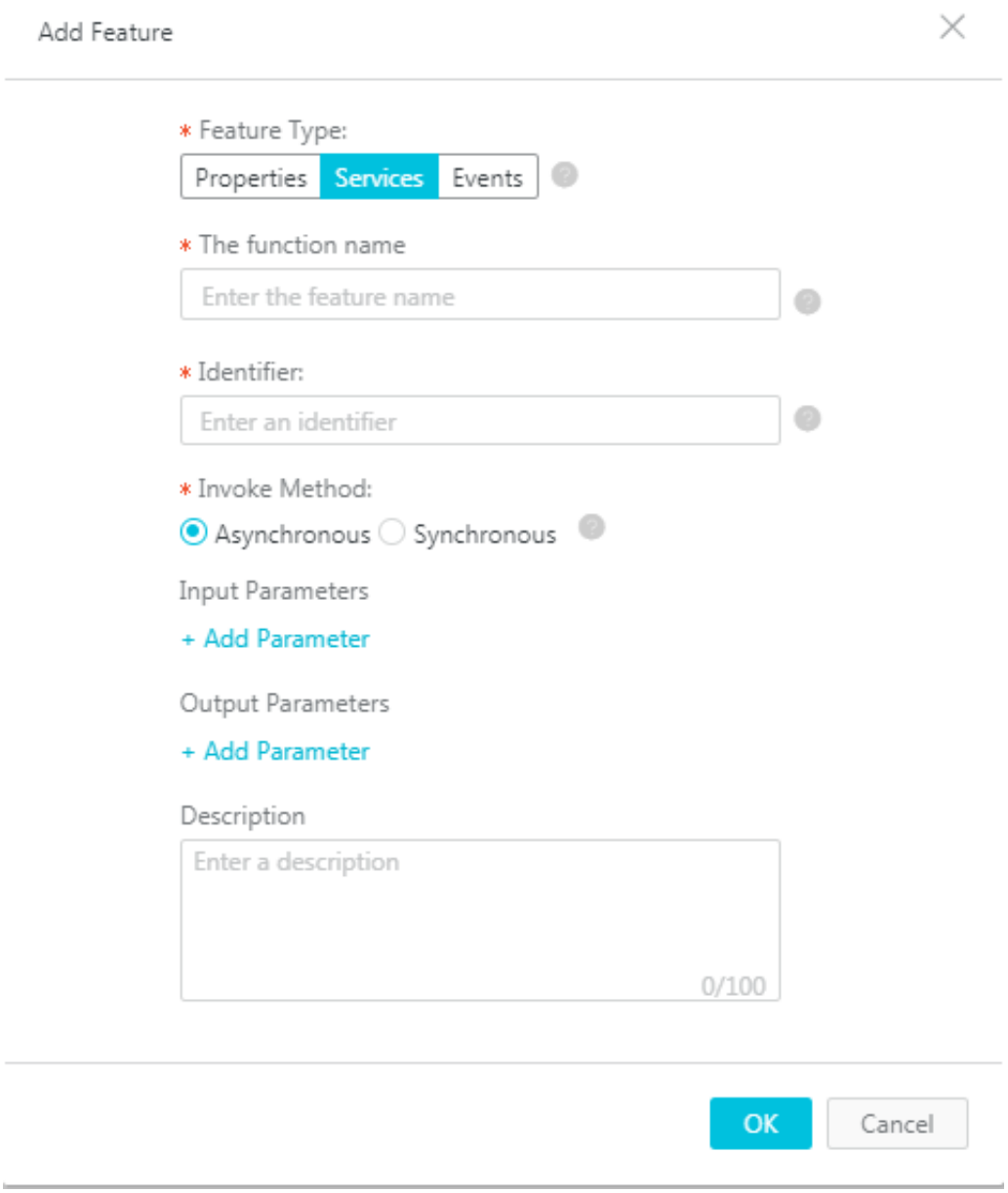
The parameters of properties are listed in the following table.

Parameter	Description
The function name	The name of the property, for example, Power Consumption. Each feature name under a product must be unique. If you have selected a feature template during product creation, the system displays standard properties from the standard feature library for you to choose. **Note:** If the gateway connection protocol is Modbus, standard properties are not supported. Instead, you must define properties manually.
Identifier	Identifies a property. It must be unique under a product. The properties parameter is **`identifier`** in Alink JSON, and is used as the key when a device is reporting the data of this property. Specifically, the cloud uses this parameter to verify the identifier and determine whether to receive the data. An identifier can contain letters, numbers, and underscores (_), for example, Power_Consumption1.
Data Type	• Int32: 32-bit integer. You must define the value range, resolution, and unit. • float: Float. You must define the value range, resolution, and unit. • double: Double. You must define the value range, resolution, 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 value. Values include 0 and 1. 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. • data: Timestamp. A UTC timestamp in string type, in milliseconds. • struct: A JSON target 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**, and **text**. 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:**

Parameter	Description
	If the gateway connection protocol is Modbus, you do not need to set this parameter.
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:** If the gateway connection protocol is Modbus, you do not need to set this parameter.
Description	Enter a description or remarks about the feature.
Extended Information **Note:** The gateway connection protocol is Modbus.	• Operation Type: You can select an operation type from Input Status (read-only), Coil Status (read and write), Holding Registers (read and write), and Input Registers (read-only). • Register Address: Enter a hexadecimal address beginning with 0x, for example, 0xFE. The range is 0x0-0xFFFF. • Original Data Type: Multiple data types are supported, including int16 , uint16, int32, uint32, int64, uint64, float, double, string, bool, and customized data (raw data). • Number of Registers: If the operation type is set to be Input Status (read-only) and Coil Status (read and write), the number range of registers is 1-2,000. If the operation type is Holding Registers (read and write) and Input Registers (read-only), the number range of registers is 1-125. • 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. • Switch Register Bits Sequence: Swap the bits of the original 32-bit data . • 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.
Extended Information **Note:**	Node Name: Each node name must be unique under the property

Parameter	Description
The gateway connection protocol is OPC UA.	

5. In the **Add Feature** dialog box, select the feature type as **Services**.



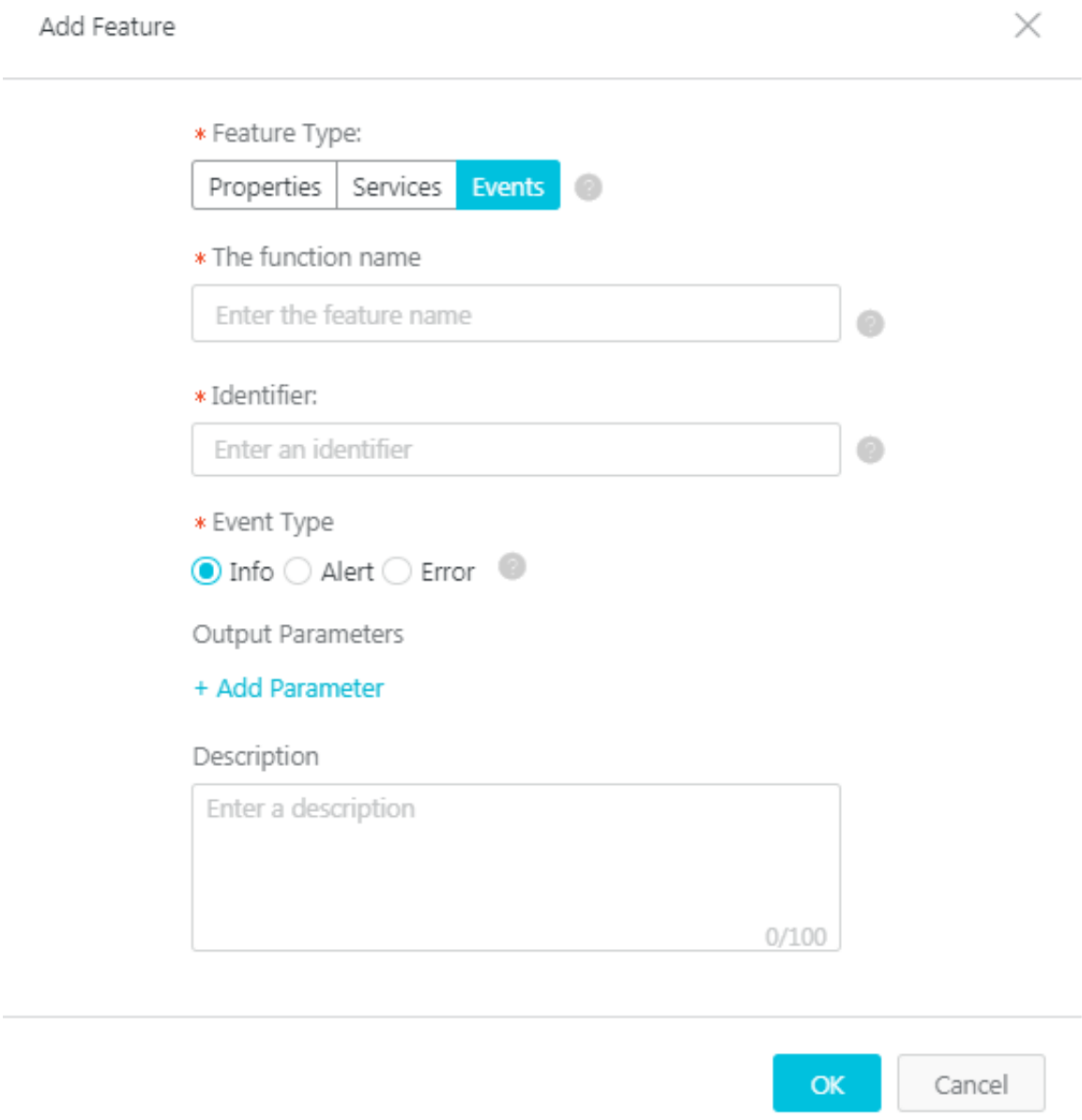
The parameters of a service are as follows.

Parameter	Description
The function name	Service name. If you have selected a feature template during product creation, the system displays standard services from the standard feature library for you to choose.

Parameter	Description
	Note: If the gateway connection protocol is Modbus, you cannot define custom services for the product.
Identifier	Identifies a service. It must be unique under a product. The service parameter is **identifier** in Alink JSON, and is used as the key when a device is reporting the data of this feature. The identifier can contain letters, numbers, and underscores (_).
Invoke Method	• Asynchronous: For an asynchronous call, the cloud returns the result directly after the call is executed instead of waiting for responses from the device. • Synchronous: For a synchronous call, the cloud waits for the response from the device. If no responses are received, the call times out.
Input Parameters	(Optional) Set input parameters for the service. Click **Add Parameter**, and add an input parameter in the dialog box that appears. For more information about input parameters, see *step 4*. If the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. **Note:** • 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. • A service supports a maximum of 10 input parameters.
Output Parameters	(Optional) Set output parameters for the service. Click **Add Parameter**, and add an output parameter in the dialog box that appears. For more information about input parameters, see *step 4*. If the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. **Note:** • You can either use a property as an output parameter or define an output parameter. For example, you can specify the property **Soil Humidity** as an output parameter. Then, when Automatic Sprinkler is called, the cloud returns the data about soil humidity.

Parameter	Description
	• A service supports a maximum of 10 output parameters.
Description	Enter a description or remarks about the service.
Extended Information **Note:** The gateway connection protocol is OPC UA.	Node Name: Each node name must be unique under the service.

6. In the **Add Feature** dialog box, select the feature type as **Events**.



The parameters of an event are as follows.

Parameter	Description
The function name	Enter an event name. **Note:** If the gateway connection protocol is Modbus, you cannot define events.
Identifier	Identifies an event. It must be unique under a product. The event parameter is **`identifier`** in Alink JSON, and is used as the key when the device is reporting the data of this feature. For example, ErrorCode.
Event Type	• `Info`: Indicates general notifications reported by the device, such as the completion of a specific task.

Parameter	Description
	• `Alert`: Indicates alerts that are reported by the device 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 the device 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 the event. Click **Add Parameter**. 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, the device reports the error with the current voltage value for further fault diagnosis. If the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. **Note:** An event supports a maximum of 10 output parameters.
Description	Enter a description or remarks about the event.
Extended Information **Note:** The gateway connection protocol is OPC UA.	Node Name: Each node name must be unique under the event.

2.4.3 Import Thing Specification Language (TSL)

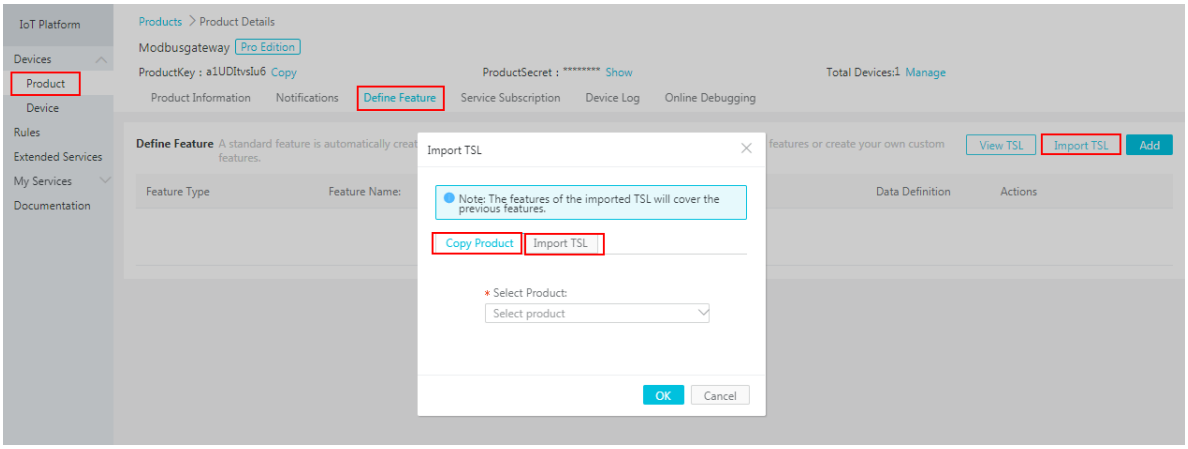
This article introduces how to import TSL for a product.

Procedure

1.
2. In the left-side navigation pane, click **Products** and then click **View** next to the product for which you want to import a TSL.
3. Click **Define Feature** > **Import TSL**.

Note:

- After you have imported a new TSL for the product, any previously defined features of the product will be overwritten. Exercise caution when using this function.
- You cannot import a TSL for a product whose gateway connection protocol is defined as Modbus.



- Copy the TSL of another product

  1. On the **Copy Product** page, select an existing product and click **OK** to import the TSL.
  2. You can then click the **Edit** button of a defined feature for this product to modify.

- If you wrote the TSL by yourself, you can just click **Import TSL** and then paste the file into the edit box.

2.4.4 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 milliseconds
), 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 description",
            "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 milliseconds), 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 description",
            "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 milliseconds), 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 milliseconds), bool (integer, 0 or 1), enum (integer), struct (
supports int, float, double, text, date, and bool), array (supports
int, double, float, and text)",
                        "specs":{
                            "min":"Minimum value, available only for
the int, float, and double data types",
                            "max":"Maximum value, available only for
the int, float, and double data types",
                            "unit":"Property unit",
                            "unitName":"Unit name",
                            "size":"Array size, up to 128 elements,
available only for the array data type",
                            "item":{
                                "type":"Type of an array element,
available only for the array data type"
                            }
                        }
                    }
                }
```

```
            ],
            "method":"Name of the method to invoke the service, which
is generated according to the identifier"
        }
    ]
}
```

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

```
{
"profile": {
"productKey": "Product ID",
  },
"properties": [
    {
"identifier": "Identifies a property. It must be unique under a
product",
"operateType": "(coilStatus/inputStatus/holdingRegister/inputRegister
)",
"registerAddress": "Register address",
"originalDataType": {
"type": "Data type: int16, uint16, int32, uint32, int64, uint64, float
, double, string, customized data(returns hex data according to big-
endian)",
"specs": {
"registerCount": "The number of registers, available only for string
and customized data",
"swap16": "swap the first 8 bits and the last 8 bits of the 16 bits of
 the register data(for example, byte1byte2 -> byte2byte10). Available
for all the other data types except string and customized data",
"reverseRegister": "Ex: Swap the bits of the original 32 bits data (
for example, byte1byte2byte3byte4 ->byte3byte4byte1byte2”. Available
for all the other data types except string and customized data"
        }
      },
"scaling": "Scaling factor",
"pollingTime": "Polling interval. The unit is ms",
"trigger": "The trigger of data report. Currently, two types of
triggering methods are supported: 1: report at the specified time; 2:
report when changes occurred"
    }
  ]
}
```

2.5 Data parsing

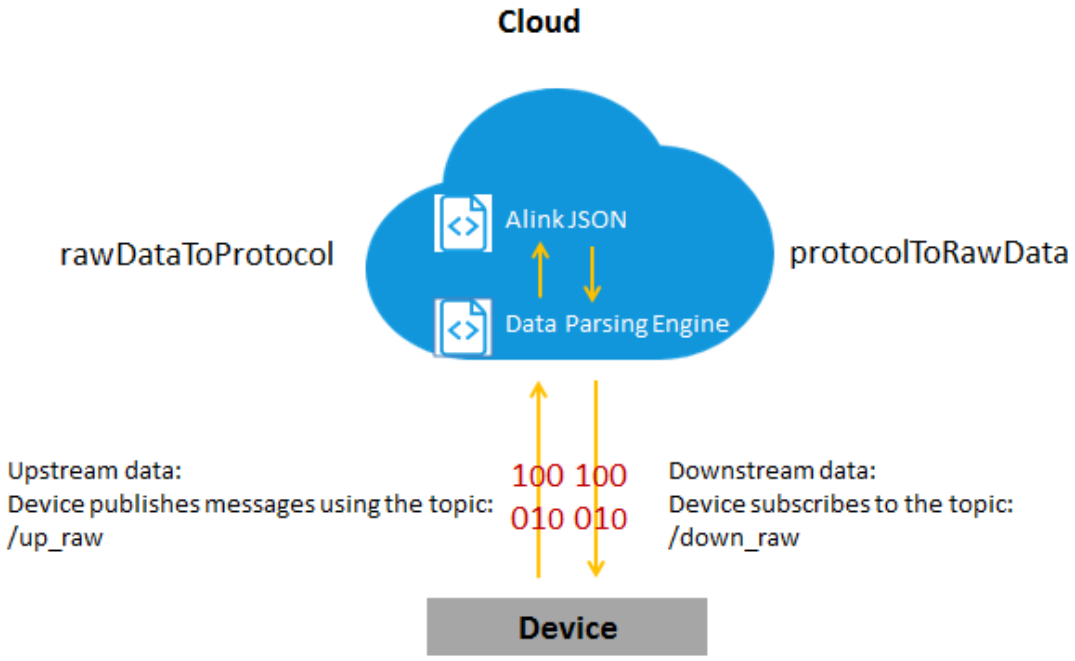
When you create a product on the IoT Platform console, if you select **Do not parse/Custom** as the data type, you can write a script in the IoT Platform console to parse the original data into Alink JSON format.

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:

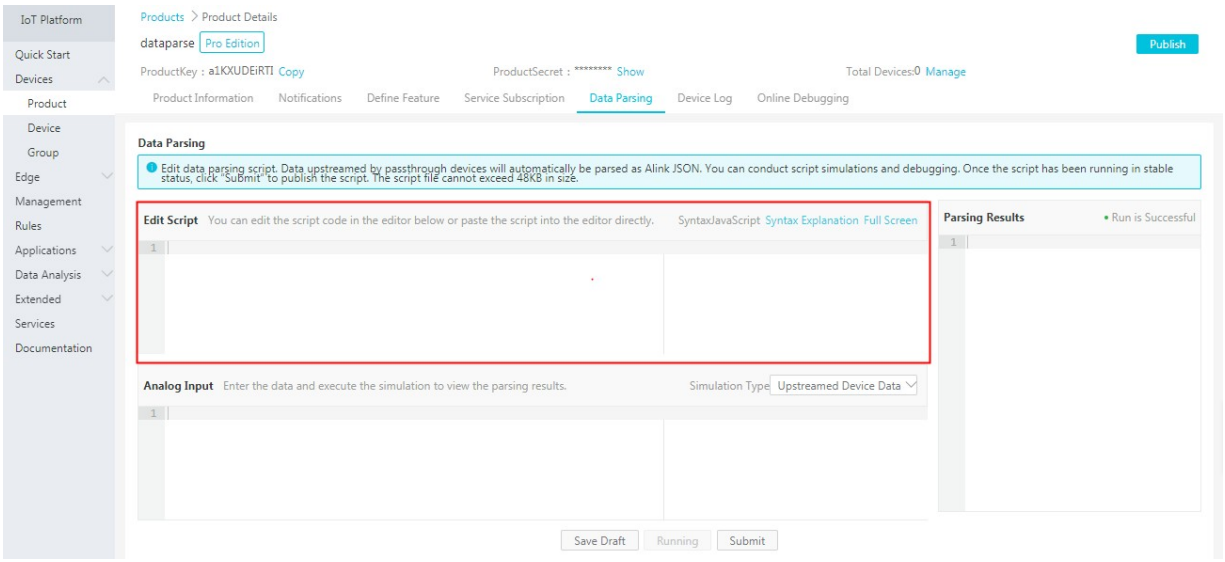


Using the data parsing script editor, you can:

- 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.

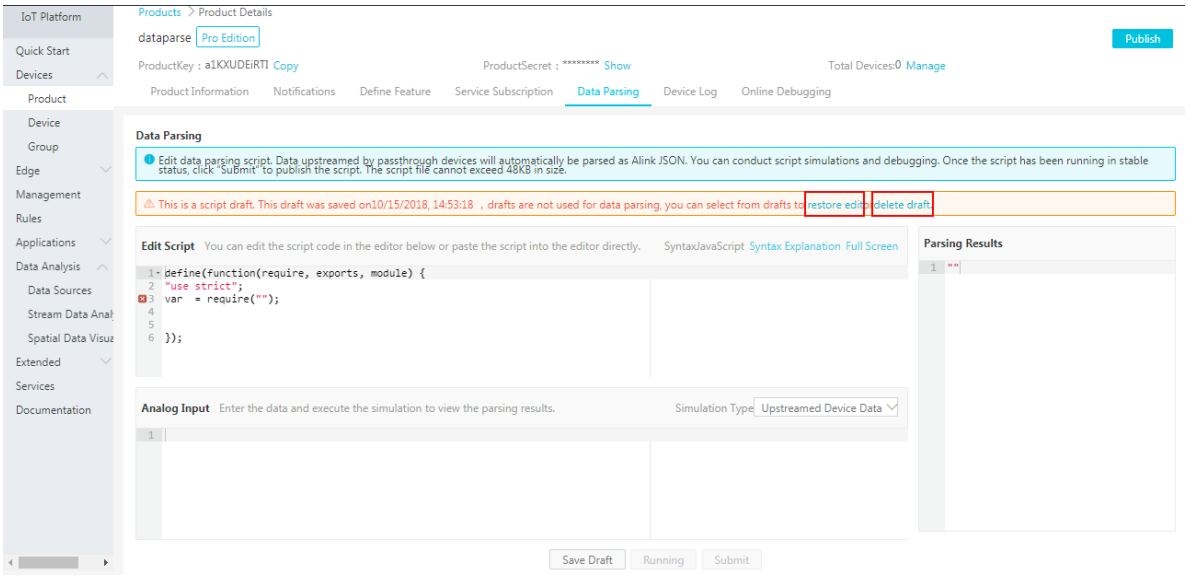
Edit a script online

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



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

- 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.

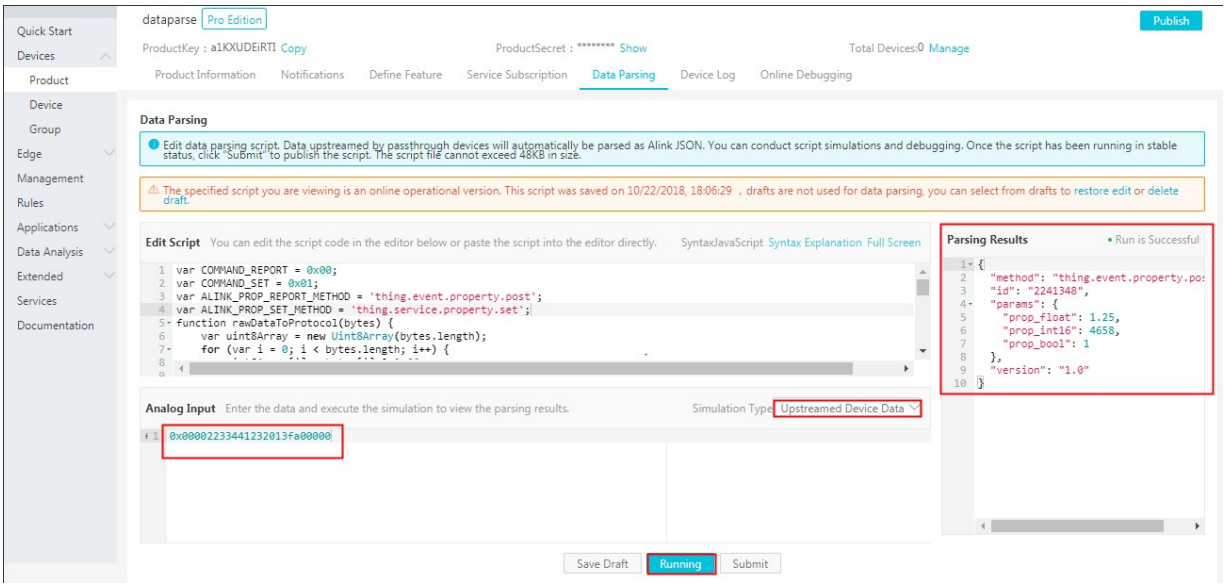


Verify the script using analog data

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.

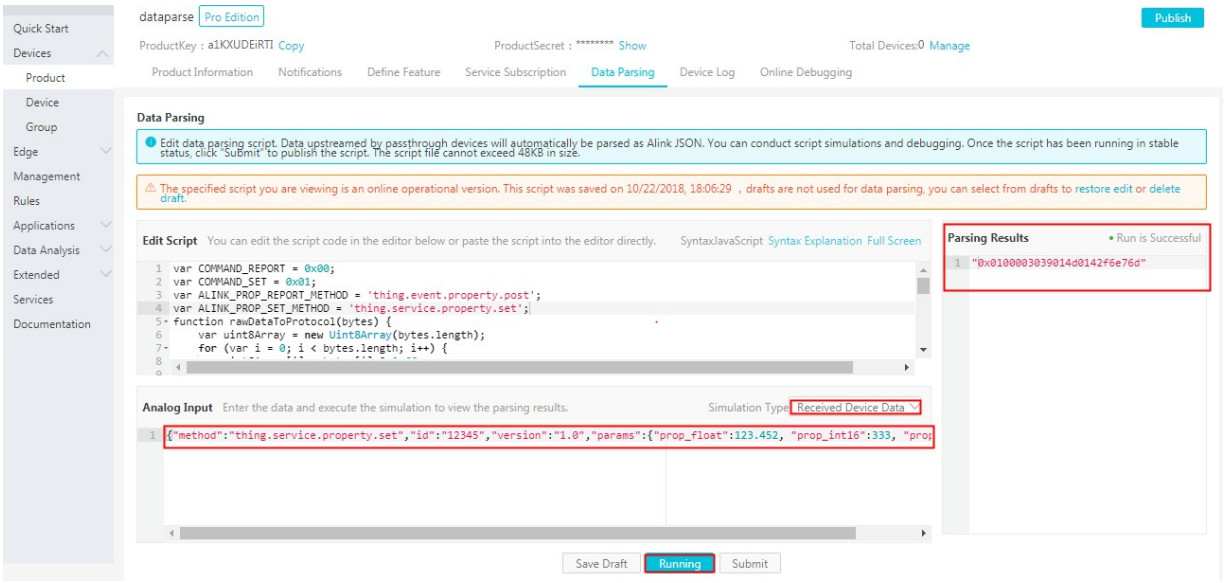
Parse upstream analog data

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.



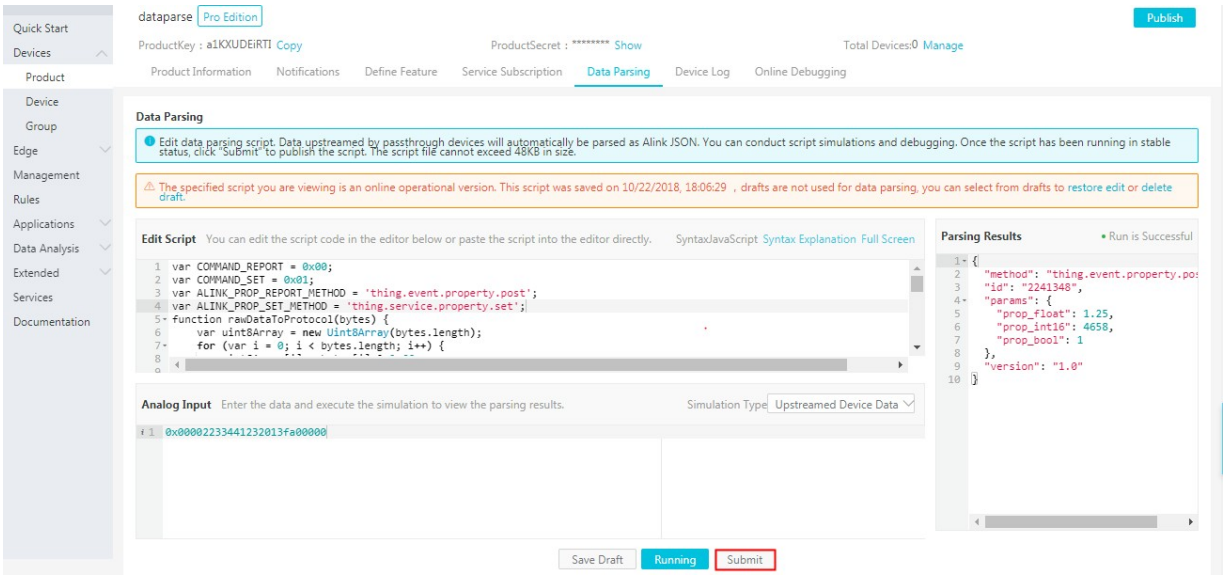
Parse downstream analog data

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.



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.



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:

```
0x0100003039014d0142f6e76d
```

- Convert binary data to Alink JSON format data:

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

```
}
```

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

```
0x00002233441232013fa00000
```

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_REPORT = 0x00;
var COMMAND_SET = 0x01;
```

```
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //A
standard ALink JSON formatted topic for devices to upload property
data to the cloud.
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'; //A
standard ALink JSON formatted topic for the cloud to send property
management commands to devices.
/*
Sample data:
Input parameters ->
    0x000022334412320l3fa00000
Output parameters  ->
    {"method":"thing.event.property.post","id":"2241348",
    "params":{"prop_float":1.25,"prop_int16":4658,"prop_bool":1},
    "version":"1.0"}
*/
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]; // command
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD; //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 ->
    0x0100003039014d0142f6e76d
*/
function protocolToRawData(json) {
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD) // 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
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET
)); // command field
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id
))); // ID in ALink JSON format
        payloadArray = payloadArray.concat(buffer_int16(prop_int16
)); // The value of property 'prop_int16'
        payloadArray = payloadArray.concat(buffer_uint8(prop_bool
)); // The value of property 'prop_bool'
        payloadArray = payloadArray.concat(buffer_float32(prop_float
)); // The value of property 'prop_float'
    }
    return payloadArray;
}
// The followings are the auxiliary functions
function buffer_uint8(value) {
    var uint8Array = new Uint8Array(1);
    var dv = new DataView(uint8Array.buffer, 0);
    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);
    return [].slice.call(uint8Array);
}
function buffer_float32(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:

     ```
     0x00002233441232013fa00000
     ```

     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:

```
0x0100003039014d0142f6e76d
```

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()
{
    //0x001232013fa00000
    var rawdata_report_prop = new Buffer([
        0x00, //Fixed command header, 0 indicates reporting property
        0x00, 0x22, 0x33, 0x44, // Identify the request sequence
corresponding to the ID fields.
        0x12, 0x32, //Two-byte value in int16,  corresponding to the
property of prop_int16
        0x01, //One-byte value in bool, corresponding to the property
of prop_bool
        0x3f, 0xa0, 0x00, 0x00 //Four-byte value in float, correspond
ing to the property of prop_float
    ]);
    rawDataToProtocol(rawdata_report_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}}');
    protocolToRawData(JSON.parse(setString));
}
```

```
Test();
```

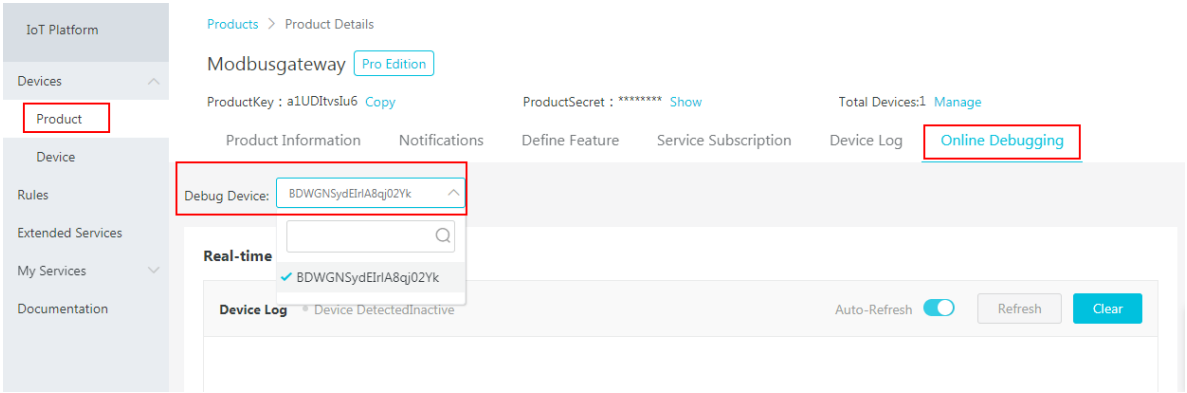
2.6 Virtual devices

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

The general development process of IoT is that, after a device client has been successfully developed, the devices report data to the cloud and developers use the data to develop the applications. Such a development process is often time consuming. In response, IoT Platform provides virtual devices that simulate physical devices connected with the cloud. The virtual devices report defined properties and handle events, and you can debug your applications according to the data reported by the virtual devices. After the physical devices become active, the virtual devices will automatically become inactive.

Procedure

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click **Products** and then click **View** next to the product that you want to debug.
3. On the **Product Details** page, click **Online Debugging**.
4. Select the target device to be debugged.

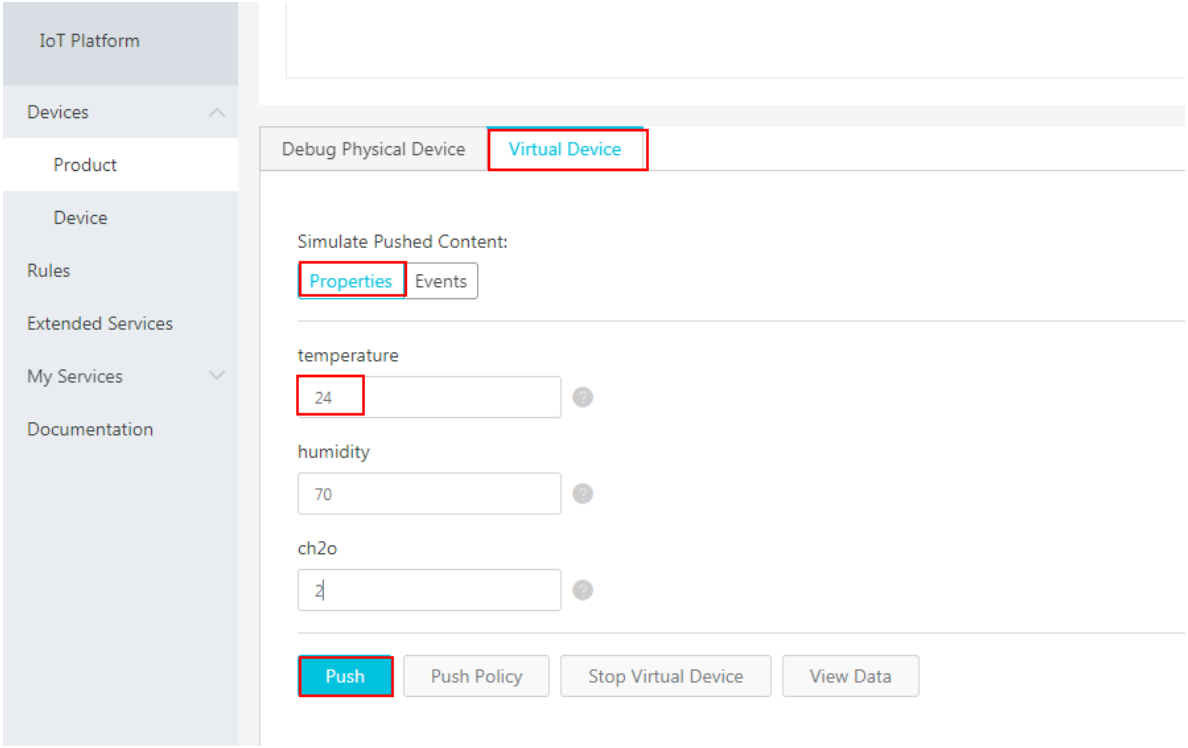


5. Click **Virtual Device** > **Start Virtual Device**.

> **Note:**
> If the physical device is active or disabled, you will be unable to start the virtual device.

6. Set the content for the simulated push.

   For example, you can set the **Properties**, indoor temperature to be 24 degrees Celsius.



7. Select a push method.

   - Push: Only push the data once.
   - 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 in seconds.

You can click **View Data** to view the running status of the device.

If you no longer require a virtual device, click **Stop Virtual Device** to stop it.

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.

2.7 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.

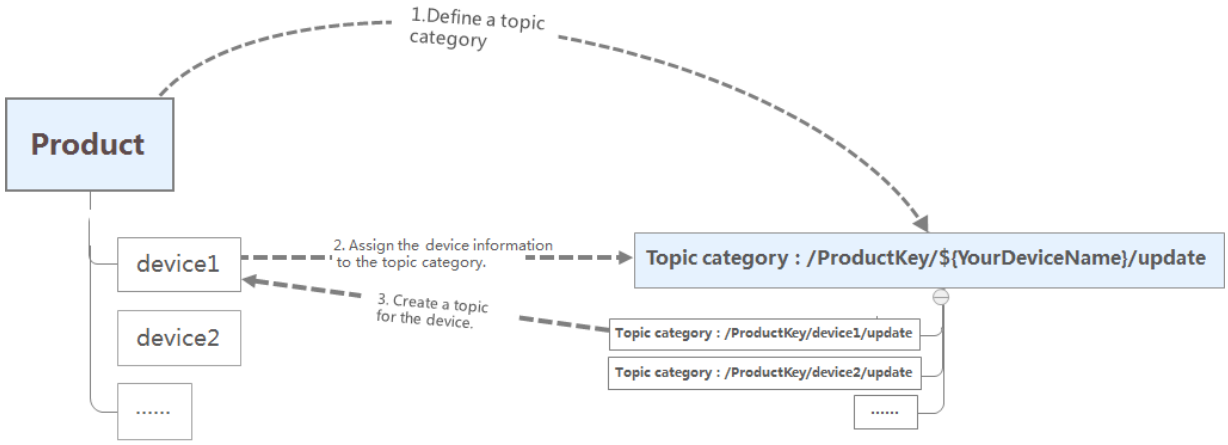
2.7.1 What is a topic?

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 2-1: The process of automatically creating a topic



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 **Communication** 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 2-1: 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

Wildcard character	Description
	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`.

2.7.2 System-defined topics

This document lists the system-defined topics.

Topic categories in IoT Platform Basic

IoT Platform Basic supports customized topic categories. IoT Platform automatically creates three customized topic categories by default when you create a product. You can also create additional customized topic categories.

- `/${productKey}/${deviceName}/update`: for devices to upload data. Permission operation: Publish.
- `/${productKey}/${deviceName}/update/error`: for devices to report errors. Permission operation: Publish.
- `/${productKey}/${deviceName}/get`: for the cloud to get device data. Permission operation: Subscribe.

Topic categories in IoT Platform Pro

IoT Platform Pro provides default system-defined topics. Topic categories are not customizeable. You can use the SDK, or use the system-defined topics associated with the Pub or Sub operations to enable two-way communication for device data between devices and the cloud.

The default topic category in IoT Pro has two categories: Alink and passthrough. The Alink topic category is used for device communication in the Alink JSON format. The passthrough topic category is used for device communication in a pass-through/custom format.
Alink topic categories:

- `/sys/${productKey}/${deviceName}/thing/event/property/post`: for devices to report properties. Permission operation: Publish.
- `/sys/${productKey}/${deviceName}/thing/service/property/set`: for devices to set properties. Permission operation: Subscribe.

- `/sys/${productKey}/${deviceName}/thing/event/{tsl.event.identifer}/post`: for devices to report events. Permission operation: Publish.
- `/sys/${productKey}/${deviceName}/thing/service/{tsl.service.identifer}`: for devices to call services. Permission operation: Subscribe.
- `/sys/${productKey}/${deviceName}/thing/deviceinfo/update`: for devices to report tags. Permission operation: Publish.

Passthrough topic categories:

- `/sys/{ProductKey}/{deviceName}/thing/model/up_raw`: uploaded passthrough data for devices to report device properties, events, and other device data. Permission operation: Publish.
- `/sys/{ProductKey}/{deviceName}/thing/model/down_raw`: downloaded passthrough data for devices to get properties, set properties, and call services. Permission operation: Subscribe.

Topic categories for device shadows

A device shadow provides the system-defined topic categories, which are mainly used by IoT Platform Basic products to update shadows.

- `/shadow/update/${productKey}/${deviceName}`: for updating the device shadows. Permission operation: Publish.
- `/shadow/get/${productKey}/${deviceName}`: for getting the device shadows. Permission operation: Subscribe.

Topic categories for remote configuration

Remote configuration provides the system-defined topic categories, which are used for sending configuration files to devices.

- /`sys/${productKey}/${deviceName}/thing/config/push`: for devices to receive the configuration files from the cloud. Permission operation: Subscribe.
- `/sys/${productKey}/${deviceName}/thing/config/get`: for devices to request a configuration update. Permission operation: Publish.
- `/sys/${productKey}/${deviceName}/thing/config/get_reply`: for devices to request a configuration update, and receive the configuration files from the cloud. Permission operation: Subscribe.

Topic categories for firmware upgrade

The firmware upgrade provides system-defined topic categories for devices to report firmware versions and receive upgrade notifications.

- `/ota/device/inform/${productKey}/${deviceName}`: for devices to report firmware versions to the cloud. Permission operation: Publish.
- `/ota/device/upgrade/${productKey}/${deviceName}`: for devices to receive firmware upgrade notifications from the cloud. Permission operation: Subscribe.
- `/ota/device/progress/${productKey}/${deviceName}`: for devices to report the progress of a firmware upgrade. Permission operation: Publish.
- `/ota/device/request/${productKey}/${deviceName}`: for devices to request a firmware upgrade. Permission operation: Publish.

Topic categories for device broadcasts

System-defined topic categories are available for you to broadcast to devices. You can define recipient devices.

`/broadcast/${productKey}/+`: for devices to receive broadcasts. Permission operation: Subscribe. The wildcard character (+) can be used to define the recipient devices of the broadcast.

Topic categories for Revert-RPC communication

The IoT Hub has integrated the synchronous communication mode based on the open-source MQTT protocol. The server sends commands to devices to receive responses in real time. These topic categories are as follows:

- `/sys/${YourProductKey}/${YourDeviceName}/rrpc/response/${messageId}`: for devices to respond to Revert-RPC requests. Permission operation: Publish.
- `/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/+`: for devices to subscribe to Revert-RPC requests. Permission operation: Subscribe.

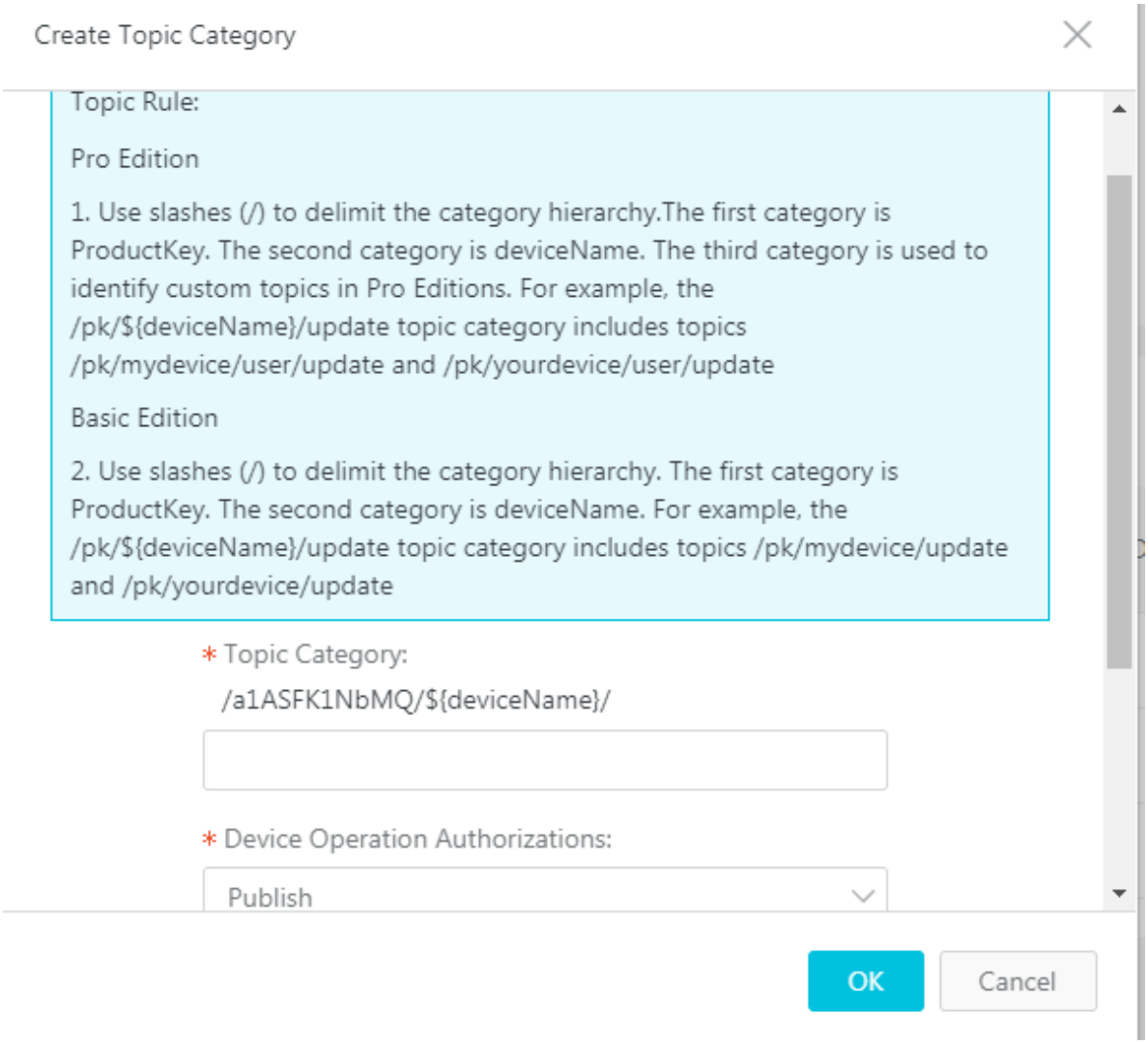
2.7.3 Create a topic category

This article introduces how to create a custom topic category for a product. Custom topic categories will be automatically assigned to devices under the product.

Procedure

1. Log on to the *IoT Platform console*.

2. In the left navigation pane, click **Products**.
3. On the **Product List** page, find the product you want to create a topic category for, and click **View** in the operation column.
4. On the **Product Details** page, click **Notifications** > **Create Topic Category**.
5. Define a topic category.



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

6. Click **OK**.

Create a topic category with a wildcard character

IoT Platform supports custom topic categories with wildcard characters. When you configure a topic subscription and you want to set topics with wildcard characters, you must first create topic categories with wildcard characters. The procedures of creating a topic category with a wildcard character is almost the same as that of creating a general topic category.

When you are creating a topic category with a wildcard character, pay attention to the following:

- You must firstly select `Subscribe` as the **`Device Operation Authorizations`**. Only when the device operation authorization is set as `Subscribe` can you enter wildcard characters in topic category name field.
- **`Topic Category`**: You can use wildcard characters `#` and `+` in the topic category name.

  Note:

  `#` can only be located at the end of the topic category name.

For topics with wildcard characters, you cannot click **Publish** to publish messages on the **Topic List** page of devices.

2.8 Tags

The IoT Platform tag is the user-defined identifier you set for a product or device. You can use tags to flexibly manage your products and devices.

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 provides tags to address this issue. You can set different tags for your products and devices. The use of tags allows the centralized management of your various products and devices.

Tags include product tags and device tags. The product tag is the template for the device tag. After a product tag is created, newly created devices under the product will be automatically assigned the product tag. You can also add device tags for each device individually. The structure of the tag is `key: value`.

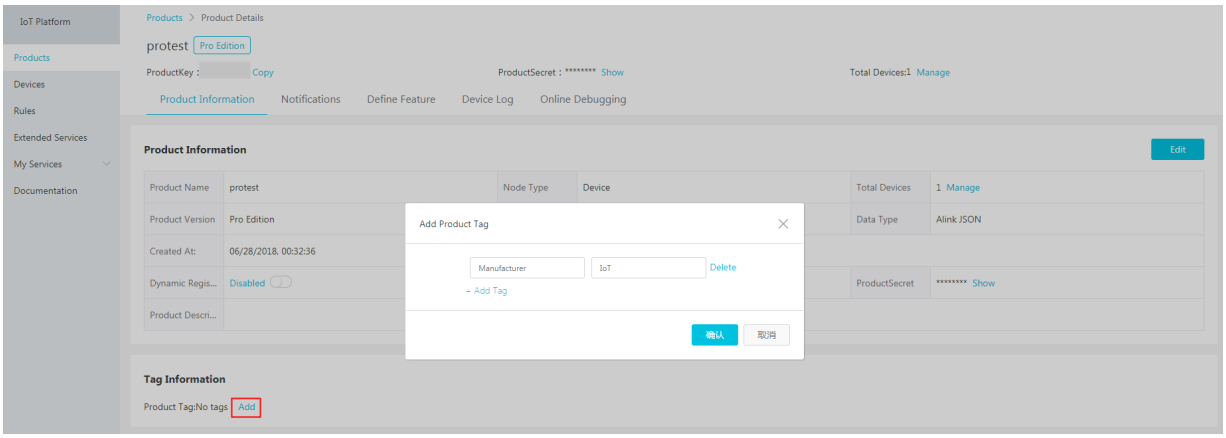
This document describes how to create a product tag and device tag.

Product tags

Product tags typically describe the information that is common to all devices under a product. For example, a tag can indicate a specific manufacturer, organization, physical size, and operating system. You have to create a product before you can add a product tag.

To add a product tag, follow these steps:

1. Log on to the *IoT Platform console*.
2. On the **Products** page, find the product to which you want to add a tag, click **View** in the Actions column, and then go to the **Product Details** page.
3. Click **Edit** in the **Product Tags** section.
4. In the **Edit Tags** dialog box, enter the **Key** and **Value** for the tag, and then click **OK**.



After a product tag is created, newly created devices under the product will be automatically assigned the product tag.

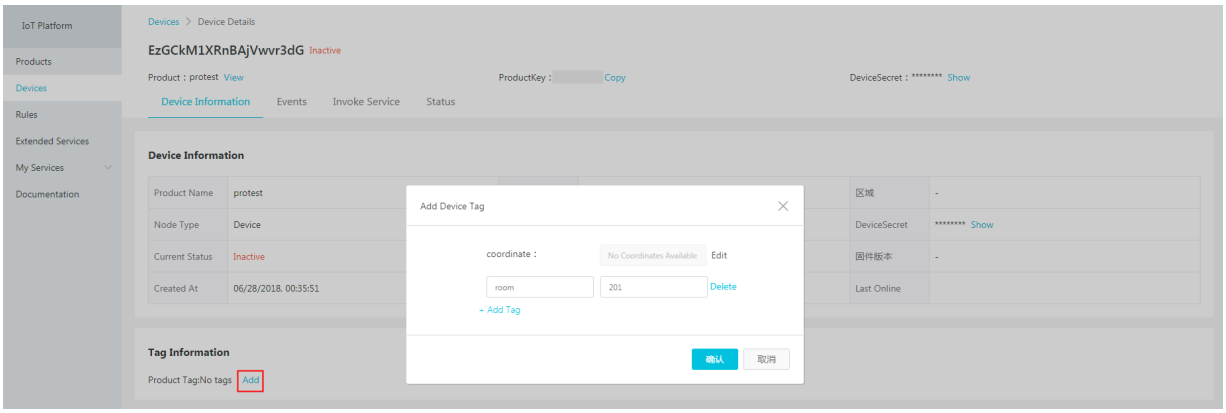
Device tags

After a device is created under a product, the device will be automatically assigned the product tag. However, the tag derived from a product only defines the information that is common to all devices under the product. Depending on the device features, you can facilitate device management by adding unique tags to the device. For example, for a smart meter in Room 201, you can add a `room:201` tag. You can manage device tags either in the console or using the API.

To add a device tag in the console, follow these steps:

1. Log on to the *IoT Platform console*.
2. Click **Devices**.

3. On the **Devices** page, find the device that you want to add a tag, click **View** under the Actions column, and then go to the **Device Details** page.
4. Click **Edit** in the **Device Tags** section.
5. In the **Edit Tags** dialog box, enter the **Key** and **Value** for the tag, and then click **OK**.



The device tag follows the device through the system. In addition, IoT Platform can send device tags to other services in Alibaba Cloud based on the Rule Engine.

2.9 Gateways and sub-devices

2.9.1 Gateways and sub-devices

IoT Platform allows devices to connect to it directly. Devices can also be mounted as sub-devices to a gateway that connects to IoT Platform.

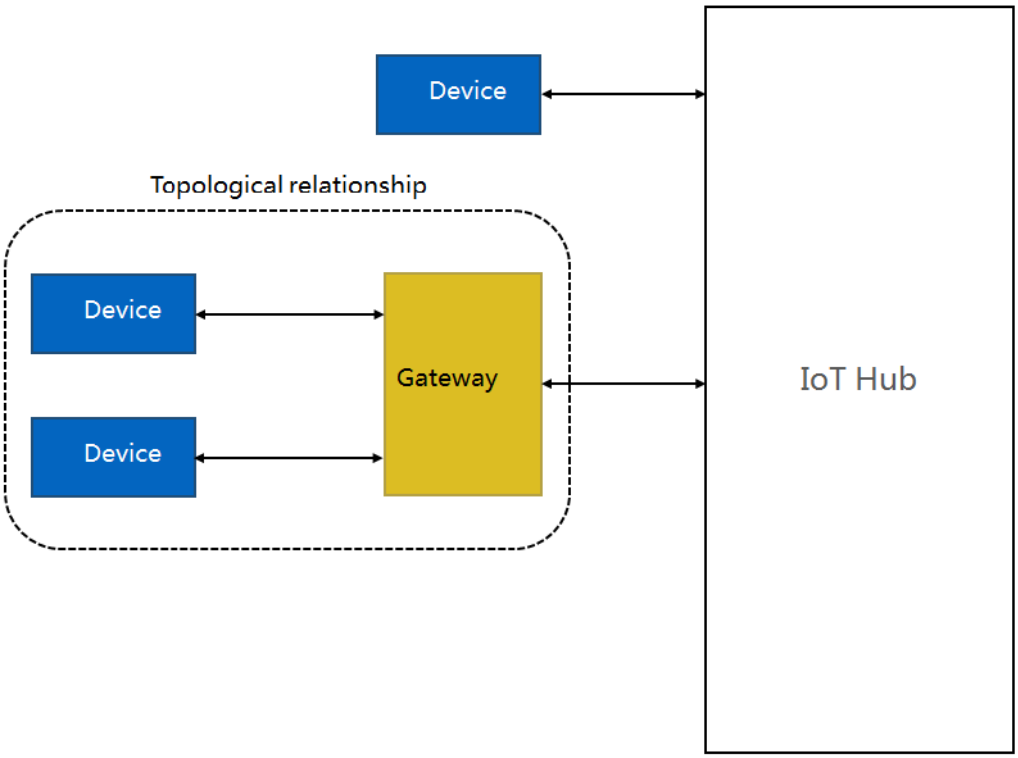
Gateways and devices

When creating products and devices, you need to select a node type. IoT Platform currently supports two node types, `device` and `gateway`.

- Device: refers to a device to which sub-devices cannot be mounted. Devices can connect directly to the IoT Hub. Alternatively, devices can connect as sub-devices mounted to gateways that are connected to the IoT Hub.
- Gateway: refers to a device to which sub-devices can be mounted. A gateway connects sub-devices to IoT Platform. Gateways can manage sub-devices, maintain their topological relationships with sub-devices, and synchronize these topological relationships to the cloud.

Topological relationship between a gateway and its sub-devices

Figure 2-2: Device topological relationship



After creating products and devices, you can:

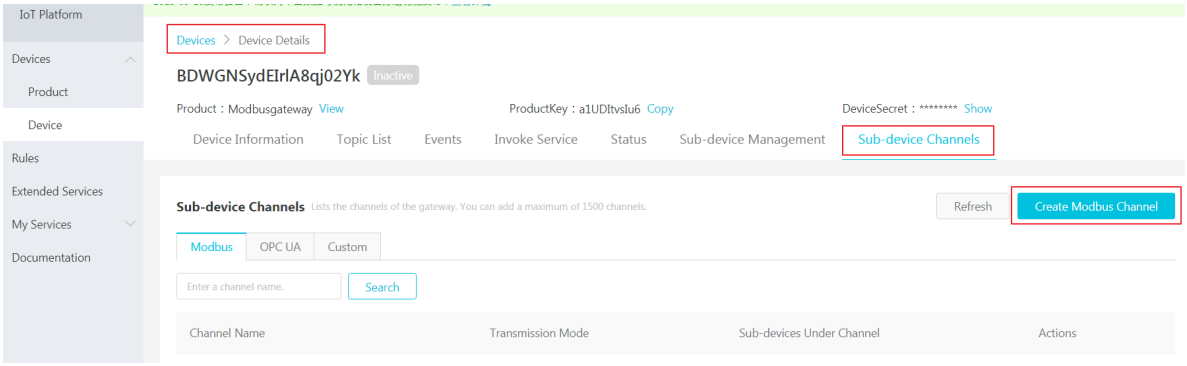
- Connect the gateway to IoT Platform and use the gateway to synchronize the topologial relationship with the cloud. The gateway is then responsible for device authentication, message uploading, instruction receiving and other communications with IoT Platform for all sub-devices. Please refer to *Developer Guide (Devices)* and *Connect sub-devices to IoT Platform* for details.
- Configure the sub-device communication channels in the console, manage the topological relationships and send the configuration details to the gateway. Please refer to *Sub-device channels* and *Sub-device management* for details.

2.9.2 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. On the **Devices** page, find the gateway device for which you want to create channels, and click **View** next to it.

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



- 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 under 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.
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 under the gateway device.
Connection Address	For example, opc.tcp://localhost:4840

Parameter	Description
User Name	An optional parameter.
Password	An optional parameter.
Function Call Timeout	The unit is in seconds.

- Custom

  1. Click **Custom** > **Create Customized Channel**.
  2. Enter a channel name in the dialog box.
  3. 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.

2.9.3 Sub-device management

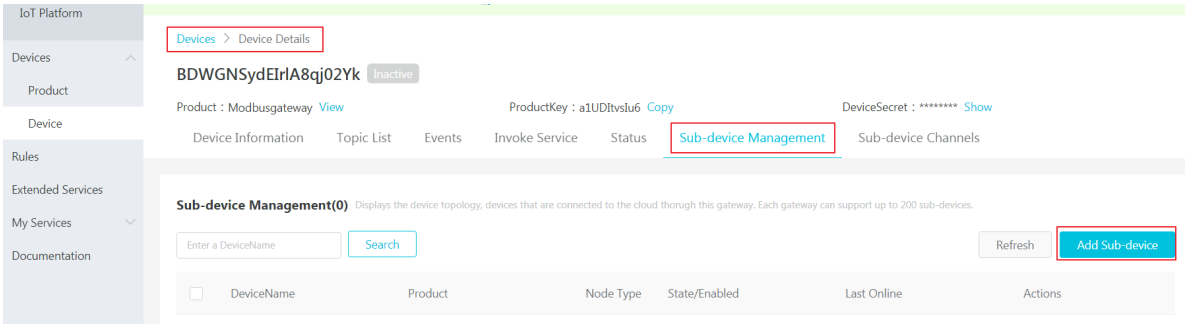
You can add a sub-device to a gateway device and assign the TSL and the extended information of the product (to which the sub-device belongs) to the sub-device.

Prerequisites

- If the gateway connection protocol of a sub-device is Modbus or OPC UA, before you connect the sub-device to the gateway, you must create a corresponding sub-device channel for the gateway. For more information about how to create sub-device channels, see the correspond ing documentation about sub-device channels.
- Products and devices created before September 3, 2018, Beijing time, can be added to gateways as sub-devices. However, you can only add their topological relationships,and not channel configurations.

Procedure

1. On the **Devices** page, find the gateway device for which you want to add sub-devices and click **View** next to it.
2. Click **Sub-device Management** > **Add Sub-device**.



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

Parameter	Description
Product	Select the name of the product that the sub-device belongs to.
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, set the following parameters.	
Associated Channel	Required. Select a channel that the sub-device uses from the created sub-device channels.
Slave Station Number	Enter an integer in the range of 1-247.
If the gateway connection protocol of the sub-device is OPC UA, set the following parameters.	
Associated Channel	Required. Select a channel that the sub-device uses from the created sub-device channels.
Node Path	Enter a node path. For example, Objects/Device1. In this example, Objects is a fixed root node, and Device1 is the node name of the device node path. Use `/` to separate node names.
If the gateway connection protocol of the sub-device is custom protocol, set the following parameters.	
Associated Channel	Optional. Select a channel that the sub-device uses from the created sub-device channels.
Custom Configuration	If you have selected an associated channel, you must customize the configuration. Only JSON format is supported.

4. 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

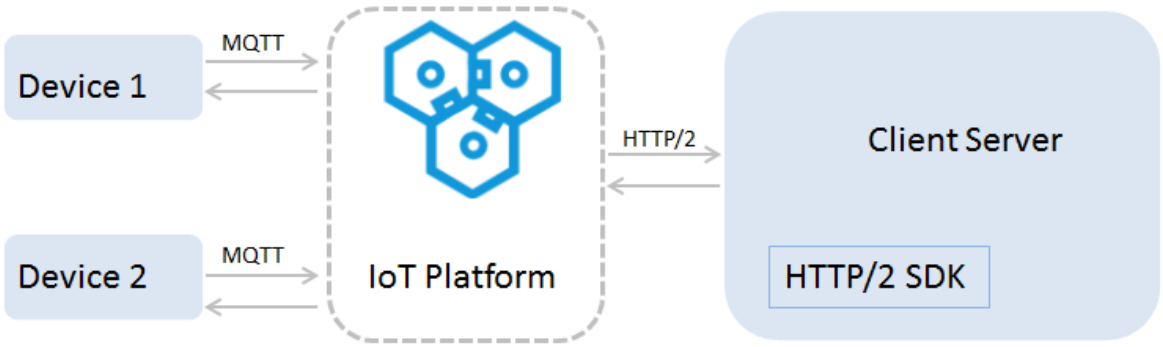
- You can refer to *Alink protocol* to develop your own devices and assign the configurations between the gateway device and the sub-device to the device client.

2.10 Service Subscription

2.10.1 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.

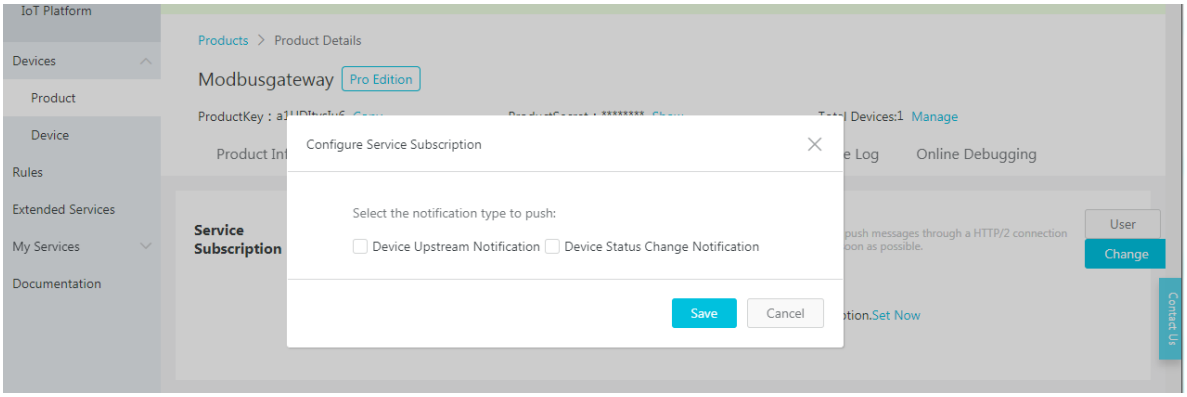
2.10.2 Development guide

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

Configure service subscription

1. Log on to the *IoT Platform console*.

2. In the left-side navigation pane, click **Products**.
3. In the product list, find the product for which you want to configure 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. There are two types: **Device Upstream Notification** and **Device Status Change Notification**.



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

  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 publish messages. In addition, 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, the notifications on devices going online or going 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.

Connect to the SDK

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

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.2</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 Alibaba Cloud account for identity authentication and to build the connection between the SDK and IoT Platform.

See the following example:

```
String accessKey = "xxxxxxxxxxxxxxxx";
String accessSecret = "xxxxxxxxxxxxxx";
String uid = "xxxxxxxxxxxxxxxx";
String region = "cn-shanghai";
String endPoint = "https://${uid}.iot-as-http2.${region}.aliyuncs.com:
443"
Profile profile = new Profile(endPoint, region, accessKey, accessSecr
et);
MessageClient client = MessageClientFactory.messageClient(profile);
client.connect(messageToken -> {
    Message m = messageToken.getMessage();
    System.out.println("receive message from " + m);
    return MessageCallback.Action.CommitSuccess;
});
```

The value of **accessKey** is the AccessKeyID of your Alibaba Cloud account, and the value of **accessSecret** is the AccessKeySecret corresponding to the AccessKeyID. Log on to the *Alibaba Cloud console*, move the pointer to your account image, and click **AccessKey** to view your AccessKey ID and AccessKey Secret; click **Security Settings** to view your account ID.

The value of **region** 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 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 callback before you `connect` the SDK to IoT Platform.

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

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

The method of message receiving interface configuration is as follows:

```
MessageCallback messageCallback = new MessageCallback() {
    @Override
    public boolean consume(MessageToken messageToken) {
        Message m = messageToken.getMessage();
        log.info("receive : " + new String(messageToken.getMessage().
getPayload()));
        return true;
    }
};
messageClient.setMessageListener("/${YourProductKey}/#",messageCal
lback);
```

The parameters are introduced as follows:

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

  A message body contains the following:

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

- For more information about message body, see *message parameters* .
- `messageClient.setMessageListener("/${YourProductKey}/#",messageCal lback);` 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 ranks higher in the dictionary 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("/alEddfaXXXX/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, generic callback is called.

  The method to configure the generic callback:

```
messageClient.setMessageListener(messageCallback);
//When 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 in 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 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 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);
```

Demo

Click *SDK demo* to download the demo.

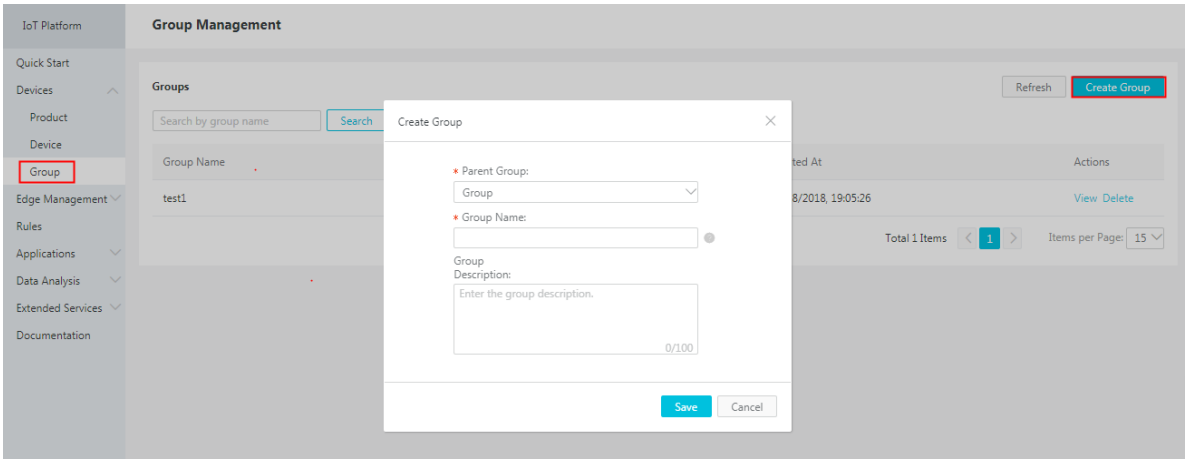
2.11 Device group

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 on the IoT Platform console.

Procedure

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

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



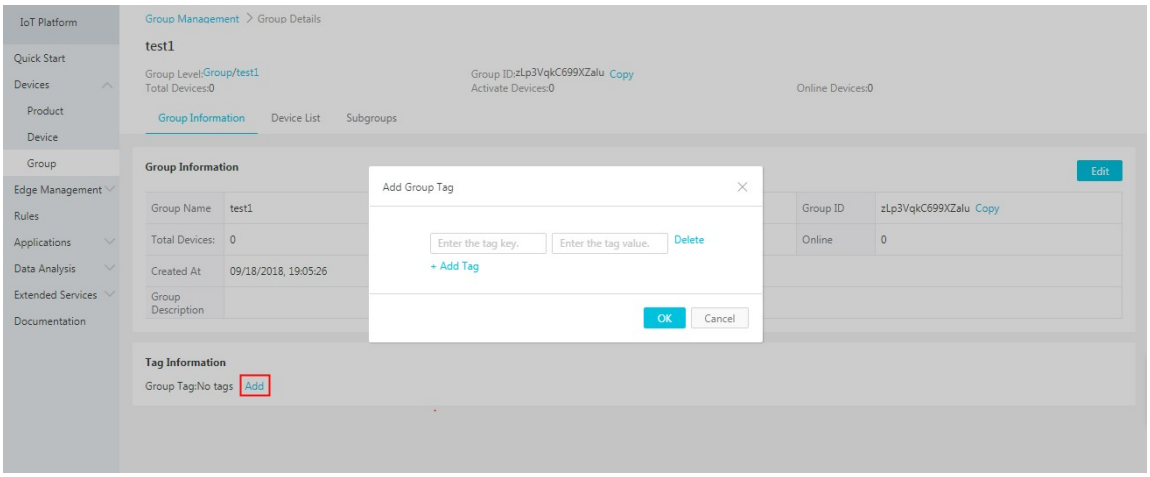
The parameters are as follows:

- **Parent Group**: Select a group type.
  - Group: Indicates that the group to be created is a parent group.
  - Select an existing group: Specifies a group as the parent group and creates a subgroup for it.

- **Group Name**: Enter a name for the group. A group name can be 4 to 30 characters in length and can include Chinese characters, English letters, digits and underscores (_) . The group name must be unique among the groups for an account, and cannot be modified once the group has been created.
- **Group Description**: Describes the group. Can be left empty.

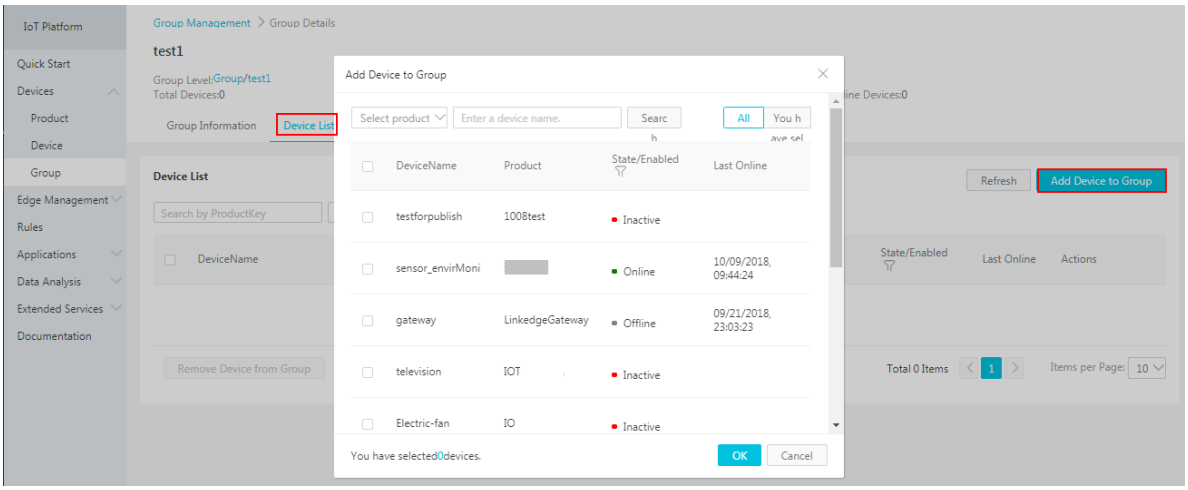
4. On the **Group Management** page, click **View** to view the **Group Details** page of the corresponding group.
5. (Optional) Add tags for the group. Tags can be used as group identifiers when you manage your groups.
   a) Click **Add** under Tag Information, and then enter keys and values of tags.
   b) Click **OK** to create all the entered tags.

> **Note:**
> You can add up to 100 tags for a group.



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

> **Note:**
> - You can add up to 200 devices at a time. You can add up to 20,000 devices for a group in total.
> - A device can be included in a maximum of 10 groups.



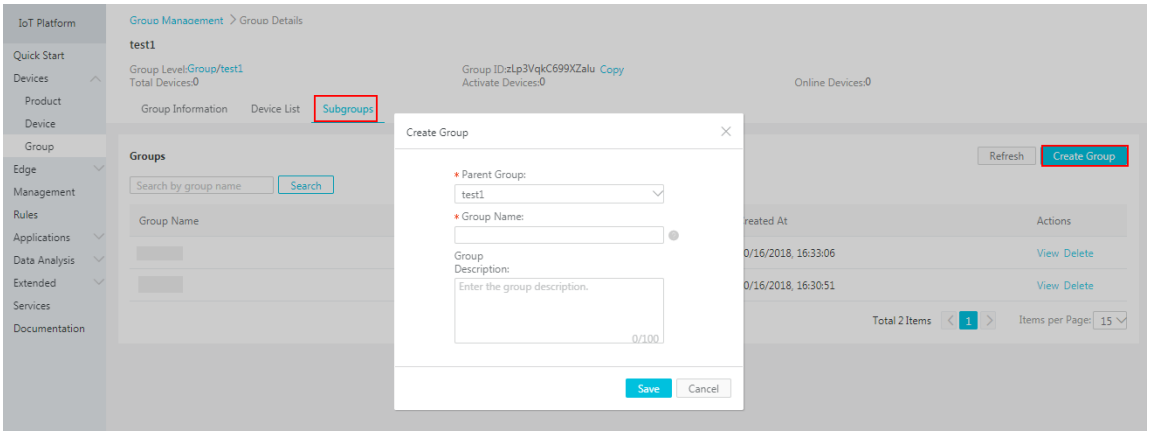
There are two buttons at the upper-right corner of the **Add Device to Group** page.

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

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

   Subgroups are used to manage devices in a more specific manner. For example, you can create subgroups such as "SmartKitchen" and "SmartBedroom" for a parent group " SmartHome", and then you can manage your kitchen devices and bedroom devices separately . The procedure is as follows:

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



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

   c) Click **Device List** > **Add Device to Group**, and then add devices for the subgroup.

   After creating the subgroup and adding devices for it, you can then manage it. You can also create sub-subgroups within the subgroup.

Note:

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

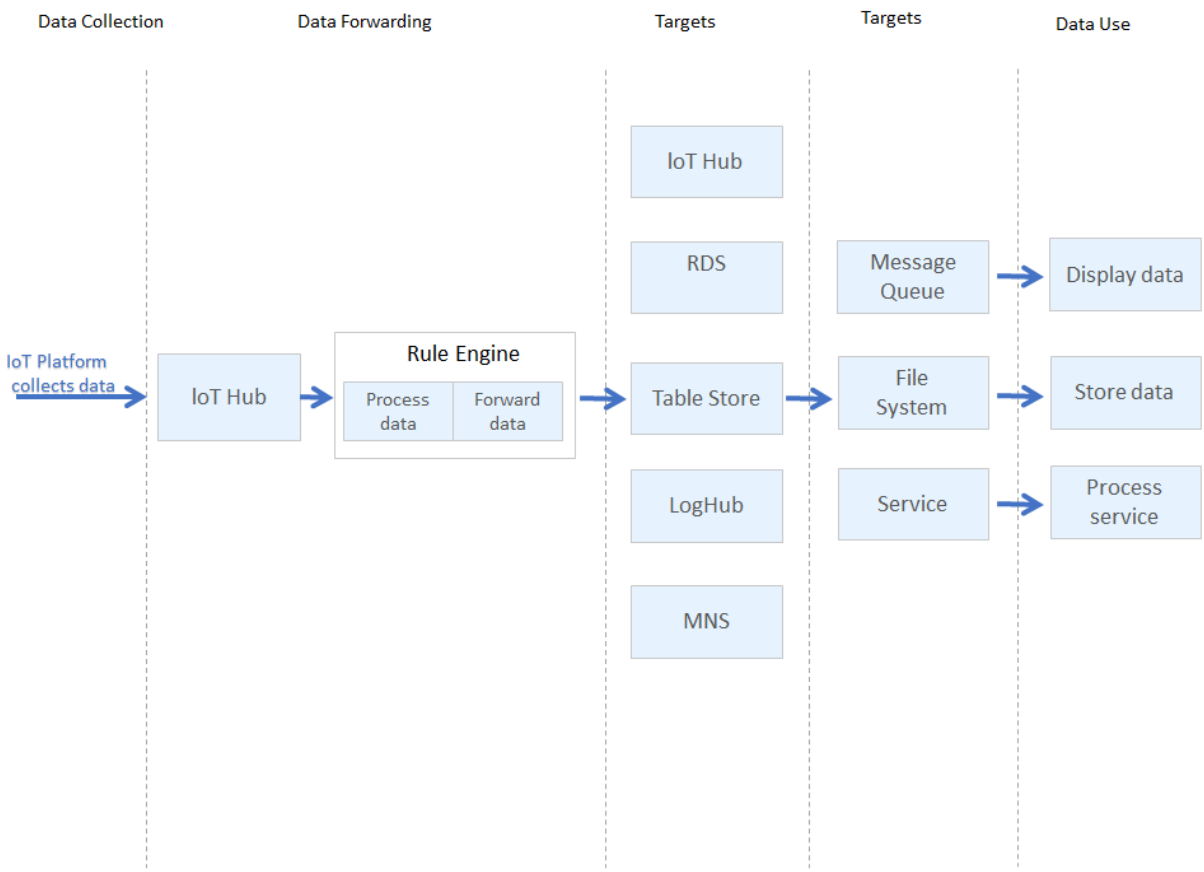
3 Rules engine

3.1 Overview

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

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

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



Note:

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

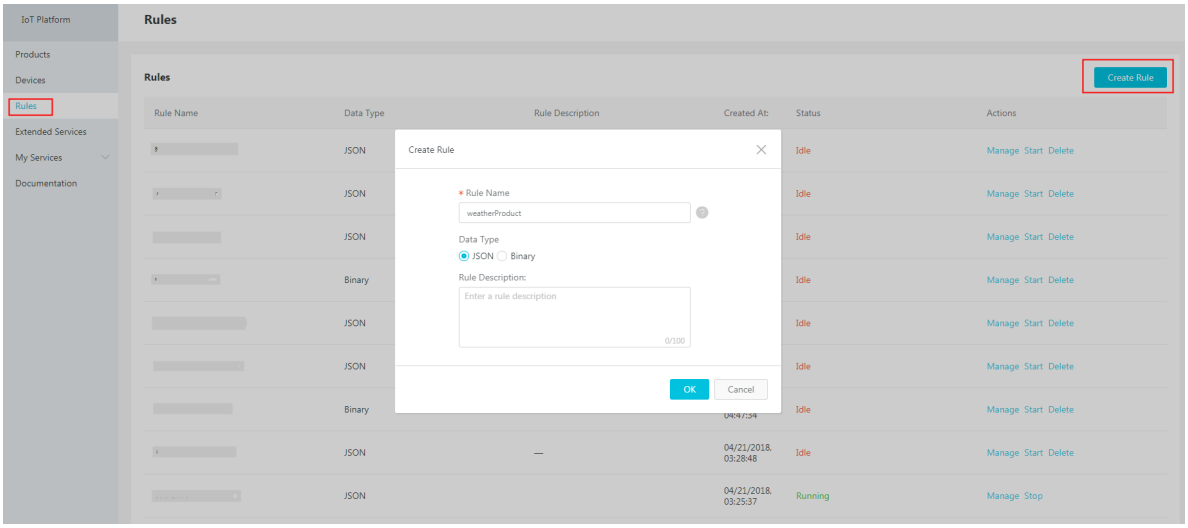
- The rule engine processes data based on topics. You can use the rule engine to process device data only when devices are communicating with each other by using topics.
- The rule engine processes the data in topics using SQL.
- SQL subqueries and the use of the LIKE operator are currently not supported.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

3.2 Create and configure a rule

This topic describes how to create and configure a rule.

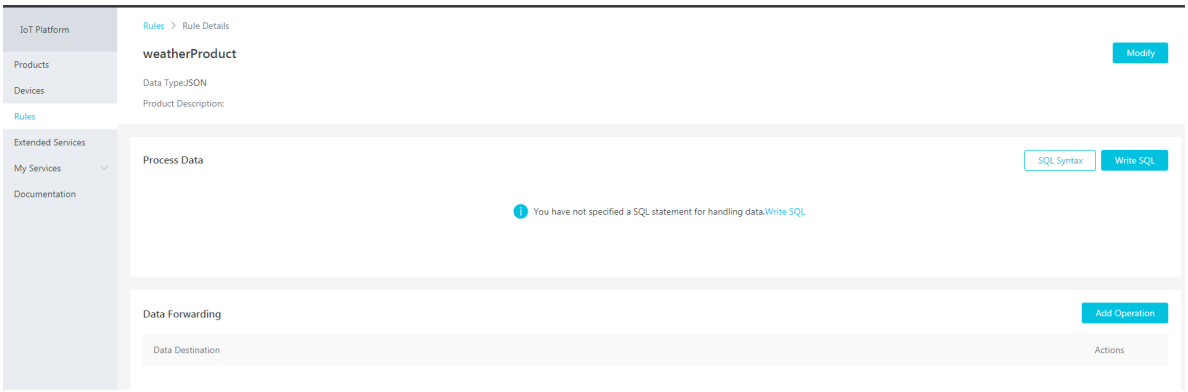
Procedure

1. On the **Rules** page of the IoT Platform console, click **Create Rule**.
2. Specify a **`Rule Name`** and select a **`Data Type`**.



- Rule Name: Enter a unique rule name. Rule names are used to identify rules.
- Data Type: JSON and binary formats are supported. 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.

3. Locate the rule you have created and click **Manage**. On the **Rule Details** page, configure the rule.

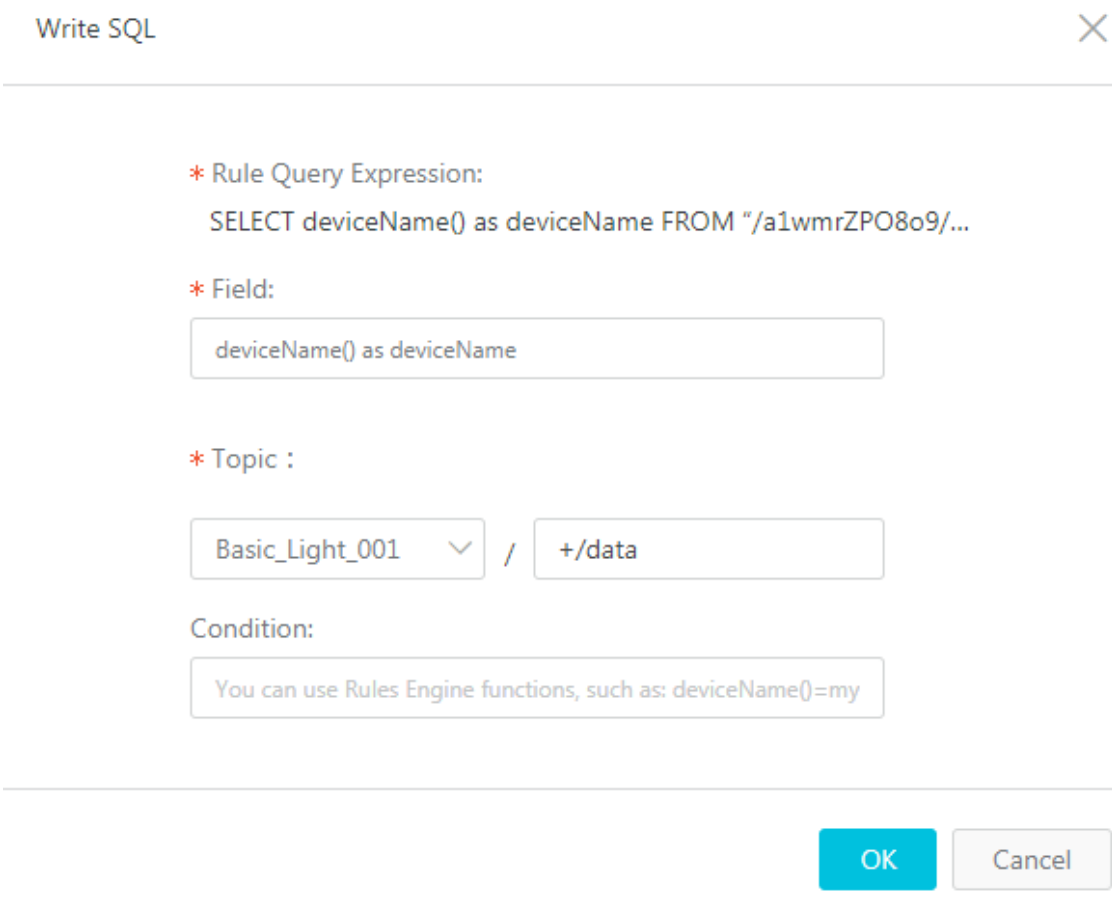


a) Click **Write SQL**, and then write a SQL statement as the detailed rule for data processing.

Note:

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

For example, the following SQL statement can be used to extract the deviceName field from the custom topic category, which ends with the level labeled data, of the product Basic_Light_001.

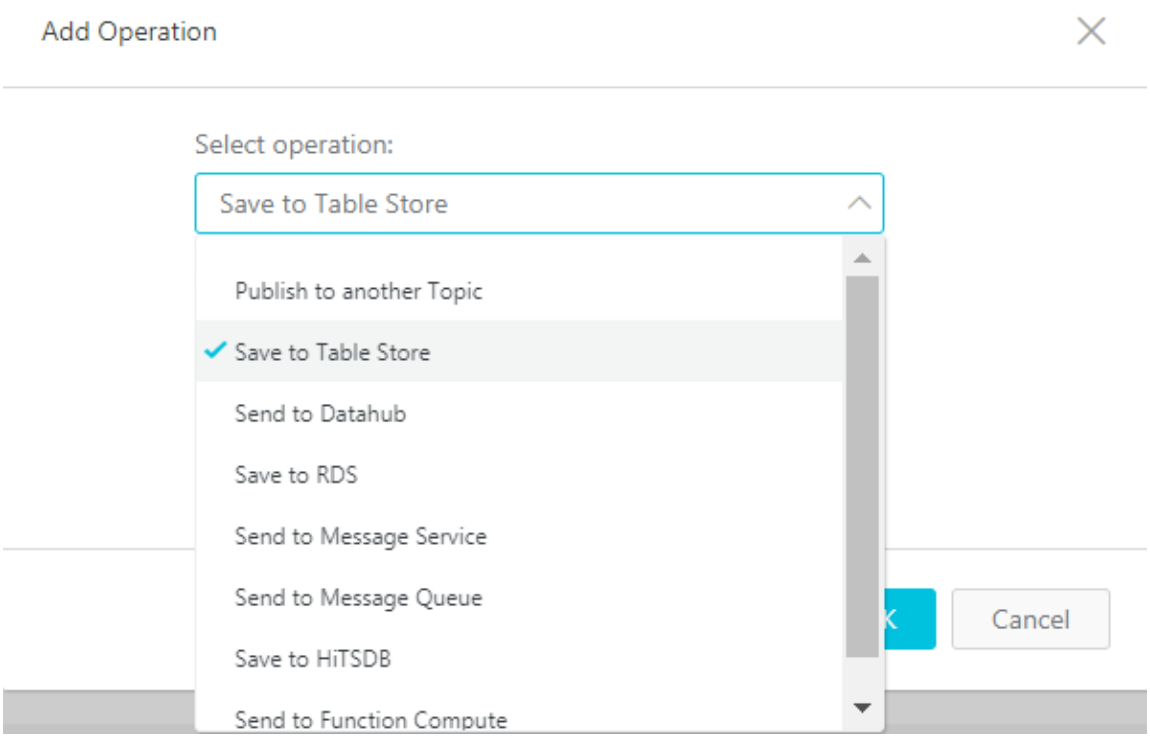


- **`Rule Query Expression`**: You must define the **`Field`**,**`Topic`**, and **`Condition`**. The system will then automatically generate a complete rule query expression.
- **`Field`**: The message content field. For example, `deviceName() as deviceName`.
- **`Topic`**: Select a topic whose messages are to be processed.
  - Custom: Indicates that it is a custom topic. After you select a product, you can enter a custom topic.
  - sys: Indicates that it is a system-defined topic. If you select sys, you must select a product, a device, and a system-defined topic.

  In this example, the custom topic category of the product Basic_Light_001 is set.
- **`Condition`**: The condition by which the rule is triggered.

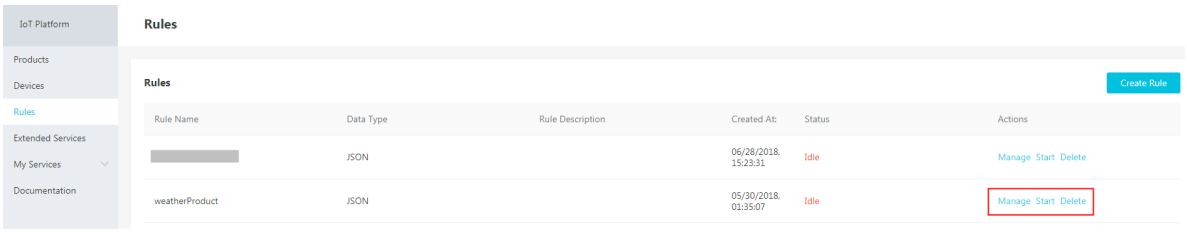
For more information about how to write SQL statements, see *SQL statements*and *Functions*.

b) Click **Add Operation** next to **Data Forwarding**. Select the Alibaba Cloud service to which you want to forward the processed data, and follow the instructions on the page to configure the parameters.



For more information about data forwarding examples, see *Examples*.

4. Go back to the **Rules** page and click **Start**. Data will then be forwarded following this rule.



You can also perform the following operations:

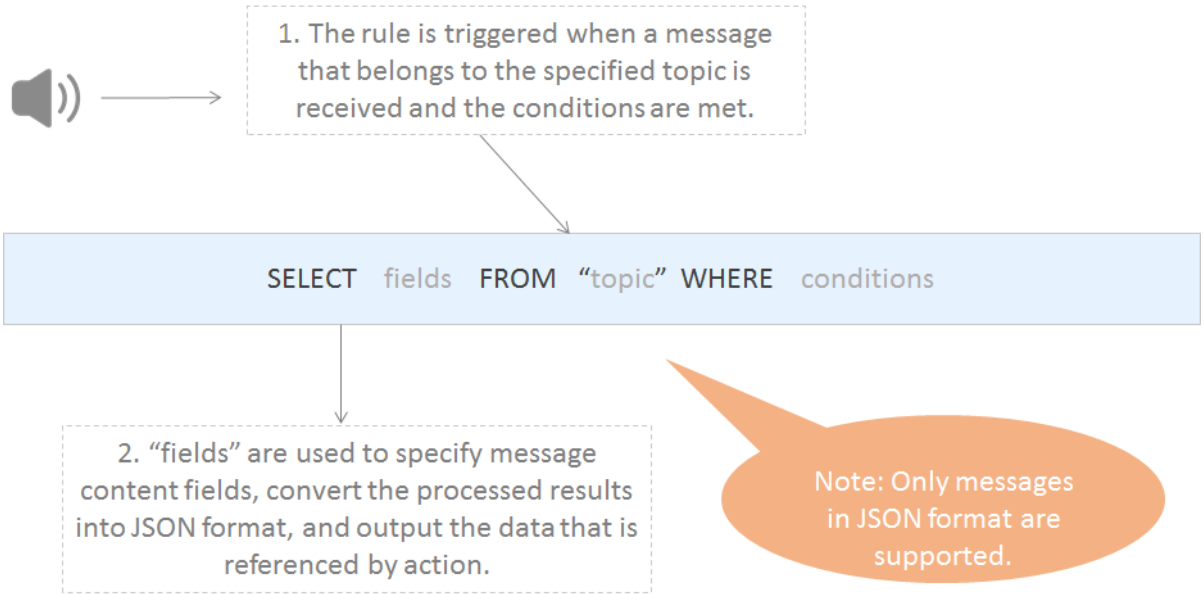
- Click **Manage** to modify the settings of this rule.
- Click **Delete** to delete this rule. Rules that are in a running status cannot be deleted.
- Click **Stop** to disable this rule.

3.3 SQL statements

When using the rules engine, if your data is in JSON format, you can write SQL statements to parse the data and process the parsing result. The rules engine does not parse binary data, but passes binary data through directly. This section 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, and values in a JSON data record correspond to the column values. Once mapped to a virtual table, a JSON data record can be processed using SQL. The following section provides an example of abstracting a rule from the rules engine into a SQL statement.




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

```
{
"temperature":25.1
"humidity":65
"pressure":101.5
"location":"xxx,xxx"
}
If you want to set an alarm to trigger when the temperature is higher
 than 38 degrees Celsius and the humidity is less than 40% RH, write
 the following SQL statement as a rule: SELECT temperature as t,
deviceName() as deviceName, location FROM /ProductA/+/update WHERE
temperature > 38 and humidity < 40
Then, if the reported data meets the rule parameters, the rule is
triggered and the temperature, device name, and location in the data
record are parsed for further processing.
```

FROM

To use the FROM statement, you must specify topic wildcards after FROM that are used to match the rule against topics that contain the messages to be processed. This means that when a message that belongs to the specified topics arrives, only the message payload that is in JSON format can be parsed and then processed by the SQL statement that you have defined. If the message format is invalid, the message will be ignored. You can use`topic()` to reference a specific topic.

```
In this example, the "FROM /ProductA/+/update" statement indicates
 that only messages that match /ProductA/+/update are processed. For
 more information about how messages match topics, see Topic.
```

SELECT

- JSON data

  In the SELECT statement, you can specify the parsed result of the payload of the reported message, which represents the keys and values in the JSON data. You can also use built-in functions in SQL statement, such as `deviceName()`. 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 the key values in the reported data record. For example, for a payload `{a :{key1:v1, key2:v2}}`, you can obtain the value of `v2` by specifying `a.key2` as the JSON path. When specifying variables in SQL statements, note the difference between using single quotes (') and double quotes ("). Single quotes (') enclose constants. Double quotes (") enclose variables. Variables may also be written without being enclosed by quotes. For example, if you use single quotes (') around a variable such as `'a.key2'`, `a.key2` will be taken as a constant.

  For more information about built-in functions, see *Functions*.

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

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

- Binary data
  - Enter `*` to pass binary data through directly.
  - Use the function `to_base64(*)` to convert binary data of the original payload to a base64 string. Built-in functions and conditions, such as `deviceName()`, are supported for extracting the required information.

WHERE

- 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 result obtained using the WHERE clause will be used to determine whether a rule will be triggered. For more information about WHERE expressions, see the following table: Supported WHERE expressions.

  ```
  In the previous example, "WHERE temperature > 38 and humidity < 40"
  indicates that the rule is triggered when the temperature is higher
  than 38 degrees Celsius and the humidity is less than 40% RH.
  ```

- Binary data

  If the reported message is composed of binary data, you can only use built-in functions and WHERE 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 you want to forward.

```
For the previous example, when configuring 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 about using arrays

Use double quotes (") when referencing arrays in SQL statements. For example, if a message is `{a:[{v:1},{v:2},{v:3}]}`, the SELECT statement is`select "$.a[0]" data1,".a[1].v" data2,".a[2]" data3`, which indicates `data1={v:1}`, `data2=2`, and `data3=[{v:3}]`.

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’
( )	Parentheses enclose the conditions that will be evaluated 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	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	The LIKE operator is used to search for a specified pattern. When you use a LIKE operator, you can only use the %wildcard character to represent a string of any characters.	For example, you can use the LIKE operator as in WHERE c1 LIKE '% abc' and WHERE c1 not LIKE '%def %'.

3.4 Functions

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

Call functions

You can call 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(temperature) tmr FROM "/topic/#" WHERE
temperature>10 and topic(2)='123'
```

Note:

Exercise caution when you call functions. 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` and `a3`represent a variable, and `a2` represents constant `a`.

Function	Description
abs(number)	Returns the absolute value of a number.
asin(number)	Returns the asin of a number.
attribute(key)	Returns the device tag value that corresponds with the key. If no such device tag key exists, the returned result is empty.
concat(string1, string2)	Concatenates two strings. Example: concat(field,'a').
cos(number)	Returns the cosine of a number.
cosh(number)	Returns the hyperbolic cosine of a number.
crypto(field,String)	Encrypts the value of a 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 no real device is connected, the returned result is empty.
endswith(input, suffix)	Validates whether the input string ends with the suffix string.
exp(number)	Returns a specific 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.
log(n, m)	Returns the logarithm of a number to the base that you have specified . If you do not specify 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 payload of encoding the text in the message 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.
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 current system time. The format parameter is optional. If you do not enter a value for format, the real-time system timestamp in milliseconds will be returned. For example, timestamp() = 1540483200000 , timestamp(' yyyy-MM-dd HH:mm:ss.SSS')=2018-10-26 00:00:00.000.
topic(number)	Returns a segment of a topic. For example, for topic /abcdef/ghi, function topic () returns " /abcdef/ ghi". Function topic (1) returns "abcdef". Function topic(2) returns "ghi ".

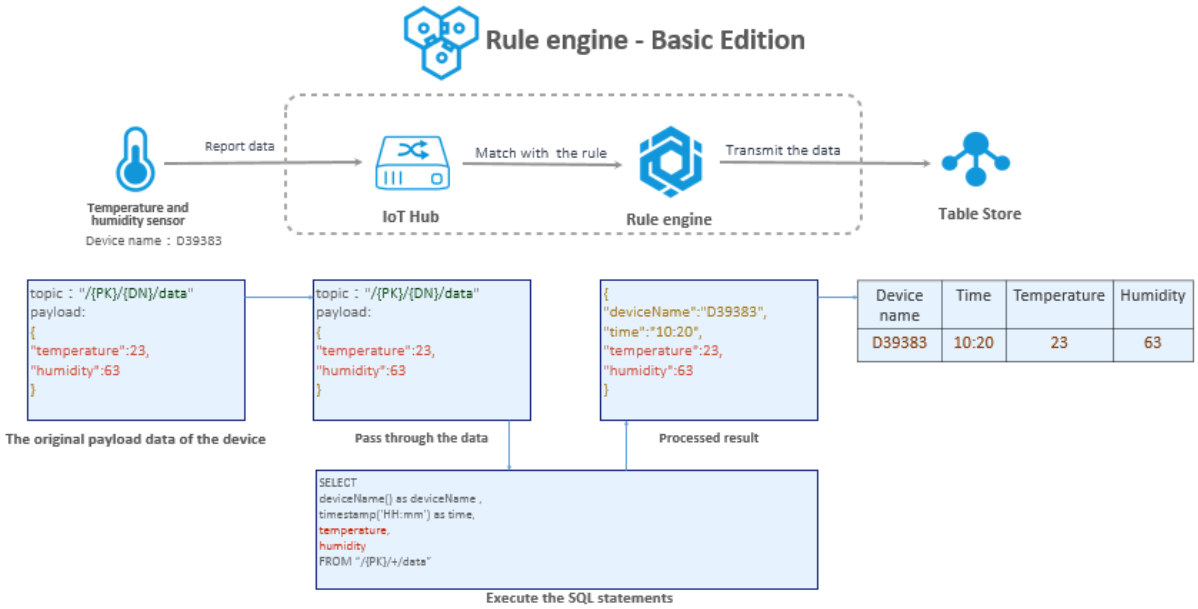
upper(string)	Returns an upper-case string.
to_base64(*)	If the original payload data is binary data, you can call this function to convert the binary data to a base64 string.

3.5 Data forwarding route

The rule engine only can process device data that is sent to topics. The data forwarding route is different for Basic Edition and Pro Edition devices.

Data forwarding route of Basic Edition devices

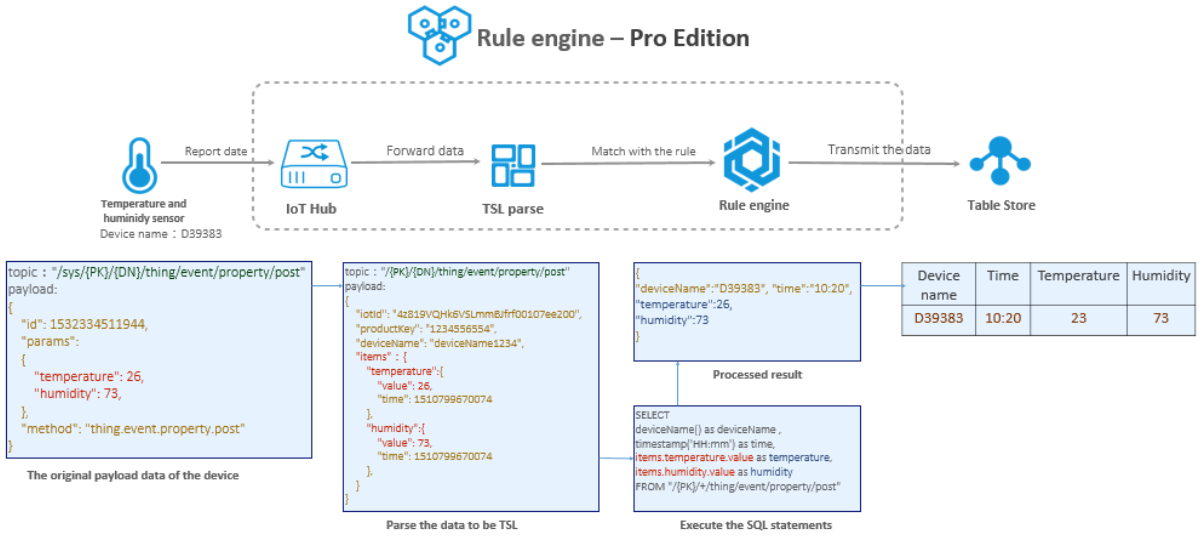
The device data of Basic Edition devices passes unaltered to IoT Hub. An example of a data forwarding route is shown in the following figure.



Data forwarding route of Pro Edition devices

When you create a Pro Edition product, you are required to select the data type as either Do not parse/Custom (indicating binary data) or Alink JSON.

- Do not parse/Custom: The rule engine does not parse binary data and the data passes to the targets. The data forwarding route is the same as that of Basic Edition devices.
- Alink JSON: Data is first parsed to be Thing Specification Language (TSL), and then the rule engine executes the SQL statements for the parsed data.



3.6 Data format in topics

Using rules engine, you need to write SQL statement to process data stored in topics. In what format data is stored in these topics is therefore important for writing the SQL statement. For IoT Platform Basic topics, data format is defined by yourself. For IoT Platform Pro topics, some are customized and some are pre-defined. For those whose data format is pre-defined, you should process data strictly accord to the format. This article explains the pre-defined data format.

Report device properties

Get device property from this topic.

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

Data format:

```
{
"iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
"productKey": "1234556554",
"deviceName": "deviceName1234",
"gmtCreate":1510799670074,
"deviceType":"Ammeter",
"items": {
"Power": {
"value": "on",
"time": 1510799670074
},
"Position":{
"time":1510292697470,
"value":{
"latitude":39.90,
"longitude":116.38
}
}
}
```

```
}
```

Parameter description:

Parameter	Type	Description
iotId	String	The unique identifier of the device within IoT Platform
productKey	String	The unique identifier of the product
deviceName	String	The name of the device
deviceType	String	The type of the device
items	Object	Device Data
Power	String	A property name. Refer to TSL for all this product's property names.
Position	String	A property name. Refer to TSL for all this product's property names.
value	Defined in TSL	Property values
time	Long	Property generated time. Use Cloud end time if not reported by device.
gmtCreate	Long	The time when message data starts to flow

Report device event

Get device event from this topic.

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

Data format:

```
{
"identifier":"BrokenInfo",
"Name": "Damage rate report ",
"type":"info",
"iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
"productKey":"X5eCzh6fEH7",
"deviceName":"5gJtxDVeGAkaEztpisjX",
"gmtCreate":1510799670074,
"value":{
"Power": "on",
"Position":{
"latitude":39.90,
"longitude":116.38
}
},
"time":1510799670074
```

```
}
```

Parameter description:

Parameter	Type	Description
iotId	String	The unique identifier of the device within IoT Platform
productKey	String	The unique identifier of the product
deviceName	String	The name of the device
type	String	Event type. Refer to TSL for details.
value	Object	Parameters for the event
Power	String	A parameter name for the event
Position	String	A parameter name for the event
time	Long	Use Cloud end time if not reported by device.
gmtCreate	Long	The time when message data starts to flow

Gateway discovers sub-devices

In some cases, the gateway can discover sub-devices and report their information. The sub-devices' information is reported using this topic.

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

Data format:

```
{
"gwIotId":"4z819VQHk6VSLmmBJfrf00107ee200",
"gwProductKey":"1234556554",
"gwDeviceName":"deviceName1234",
"devices":[{
"productKey":"12345565569",
"deviceName": "deviceName1234",
"iotId":"4z819VQHk6VSLmmBJfrf00107ee201"
}]
}
```

Parameter description:

Parameter	Type	Description
gwIotId	String	The unique identifier of the gateway device within the Platform
gwProductKey	String	The unique identifier of the gateway product

Parameter	Type	Description
gwDeviceName	String	The name of the gateway device
iotId	String	The unique identifier of the sub-device within IoT Platform
productKey	String	The unique identifier of the sub-device product
deviceName	String	The name of the sub-device

Instruction execution result

Obtain the device's instruction execution result from this topic when instructions were sent asynchronously from the Cloud. If an error occurs when sending the instructions, you can get error message from this topic.

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

Data format:

```
{
"gmtCreate": 1510292739881,
"iotId": "4z819VQHk6VSLmmBJfrf00107ee200",
"productKey": "1234556554",
"deviceName": "deviceName1234",
"requestId": 1234,
"code": 200,
"message": "success",
"topic": "/sys/1234556554/deviceName1234/thing/service/property/set",
"data": {}
}
```

Parameter description:

Parameter	Type	Description
gmtCreate	Long	UTC timestamp
iotId	String	The unique identifier of the device within IoT Platform
productKey	String	The product key.
deviceName	String	The name of the device
RequestId	Long	The identifier of message between Alibaba Cloud and devices
code	Integer	The code for result message
message	String	The description of the result

Parameter	Type	Description
data	Object	The result reported by the device. For passthrough communication, this result should be converted by script.

Response messages

Parameter	Type	Description
200	success	The request is successful.
400	request error	Internal service error occurs when executing the instructions.
460	request parameter error	Request parameter error. Verification of input parameter failed.
429	too many requests	Too many requests in a short time.
9200	device not actived	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.

Online offline status

Obtain the online and offline status of devices from this topic.

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

Data format:

```
{
"productKey": "1234556554",
"deviceName": "deviceName1234",
"gmtCreate":1510799670074,
"deviceType":"Ammeter",
"iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
"action":"online",
"status"{
"value": "14",
"time":1510292697471
}
}
```

Parameter description:

Parameter	Type	Description
iotId	String	The unique identifier of the device within IoT Platform
productKey	String	The unique identifier of the product
deviceName	String	The name of the device
status	Object	The status of the device
value	String	1 represents online and 0 offline.
time	Long	The time when the device got online or offline
gmtCreate	Long	The time when the message starts to flow
action	String	online or offline

3.7 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 3-1: 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	√	√	√	√	-	-	-	-	-	-

3.8 Examples

3.8.1 Forward data to another topic

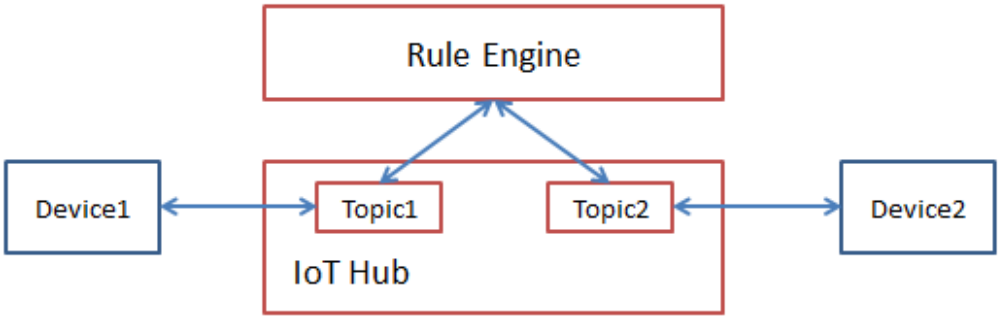
You can forward the data that is processed based on SQL rules to another topic for machine-to-machine (M2M) communication and other applications.

Prerequisites

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

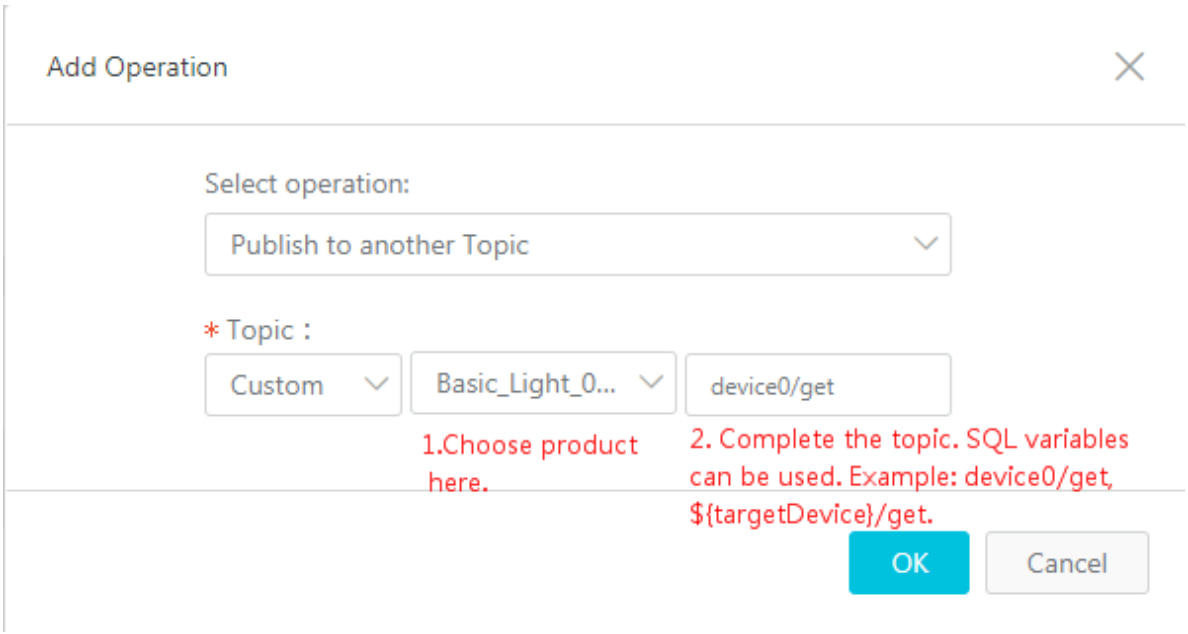
Context

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.



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.

3.8.2 Forward data to Table Store

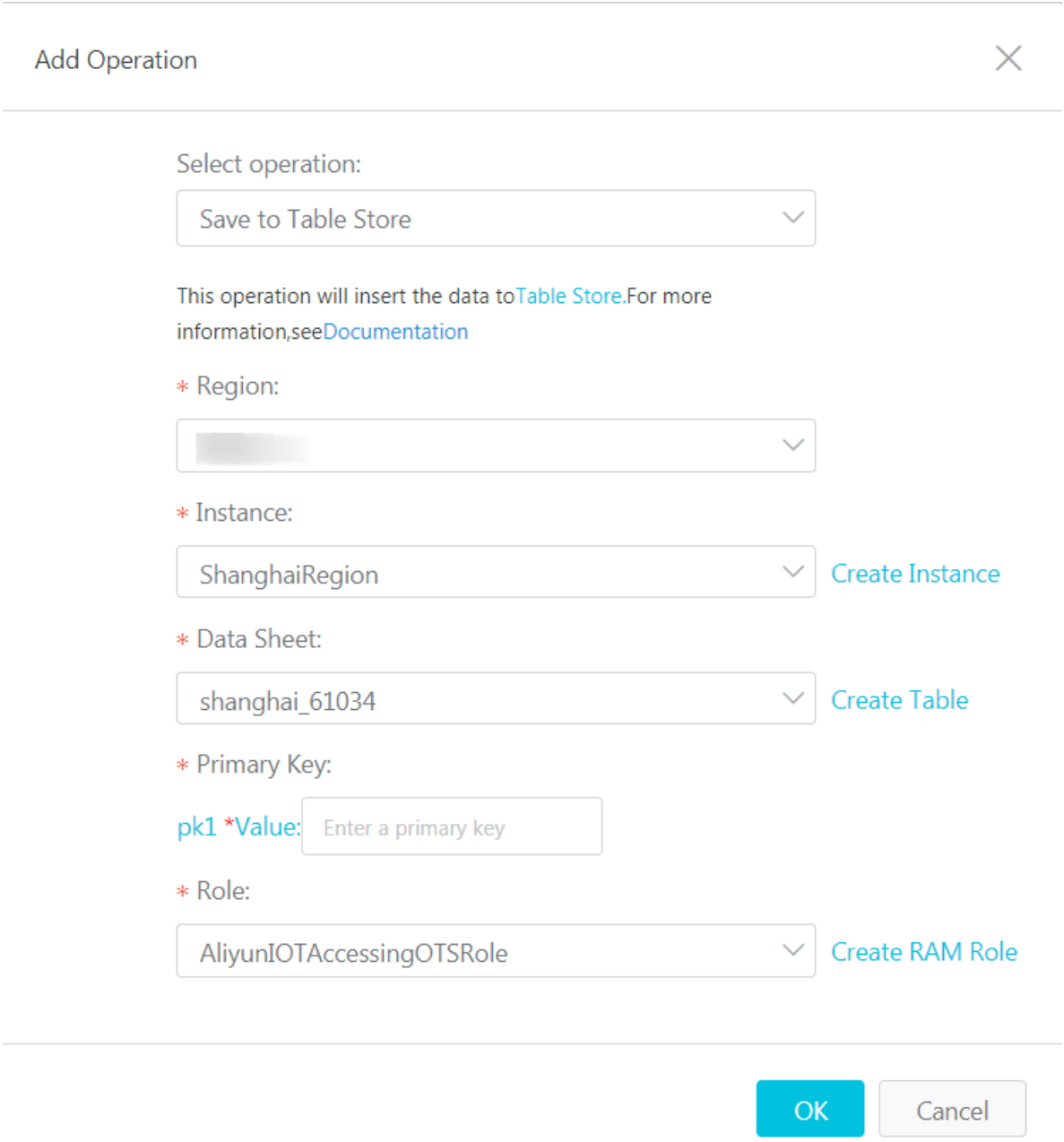
You can configure the rules engine to forward the processed data to Table Store.

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** .



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.

3.8.3 Forward data to RDS

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

Restrictions

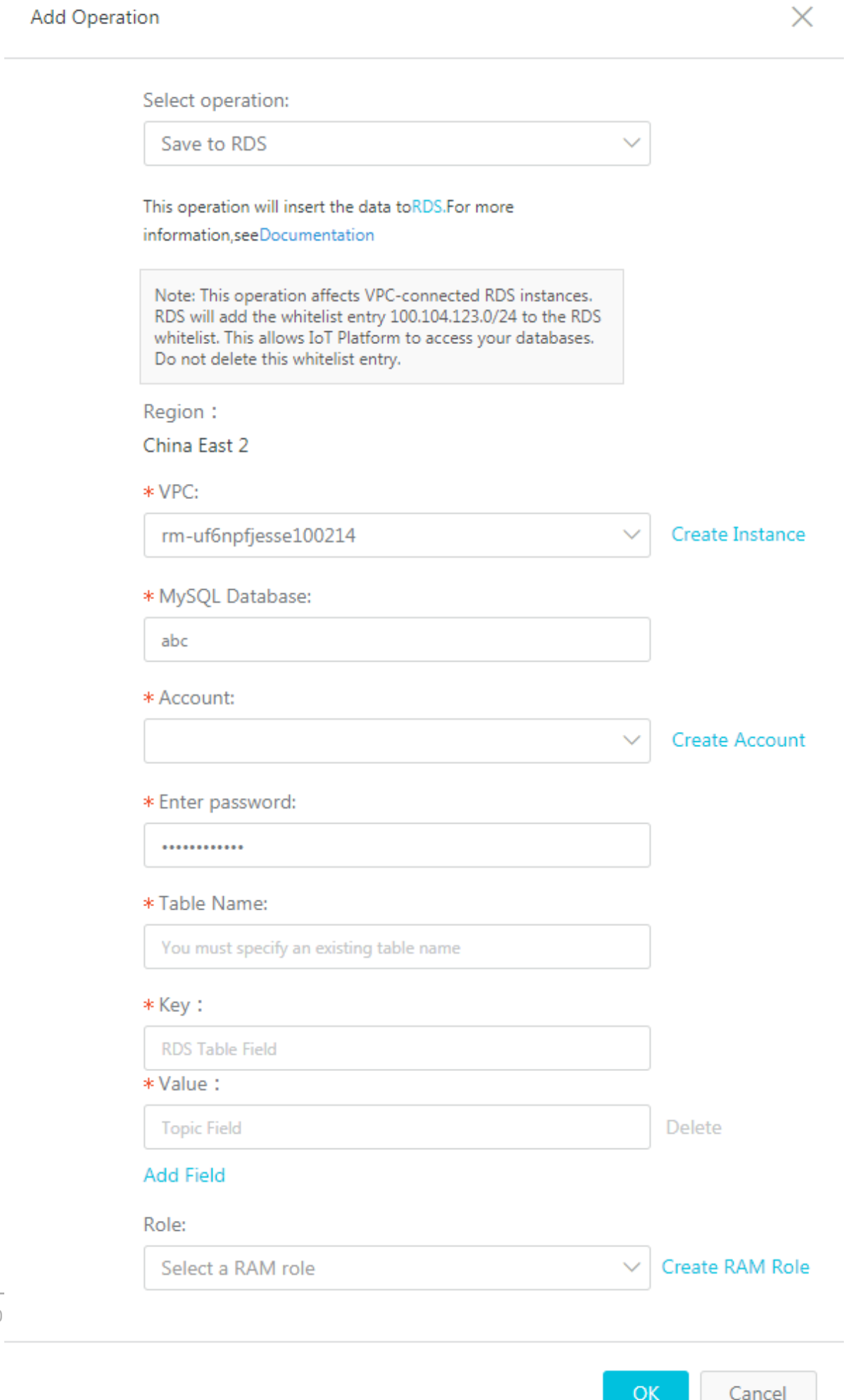
- Data can be forwarded only from IoT Platform to RDS within the same region but cannot be forwarded from IoT Platform to RDS in a different region. For example, for an IoT Platform on China East 2, data can only be forwarded to a RDS in China East 2.
- Only forwarding to RDS instances in VPCs is supported.
- Only MySQL instances are supported.
- Supports forwarding to databases in classic and master modes.

Preparation

Before you configure a forwarding rule, you need to follow the instructions in *Create and configure a rule* to write a SQL script and process the data.

Procedure

1. Click **Add Operation** next to **Data Forwarding** to open the **Add Operation** dialog box. Select **Save to RDS**


Add Operation
Select operation:
Save to RDS
This operation will insert the data toRDS.For more information,seeDocumentation
Note: This operation affects VPC-connected RDS instances. RDS will add the whitelist entry 100.104.123.0/24 to the RDS whitelist. This allows IoT Platform to access your databases. Do not delete this whitelist entry.
Region :
China East 2
* VPC:
rm-uf6npfjesse100214
Create Instance
* MySQL Database:
abc
* Account:
Create Account
* Enter password:
* Table Name:
You must specify an existing table name
* Key :
RDS Table Field
* Value :
Topic Field
Delete
Add Field
Role:
Select a RAM role
Create RAM Role
OK
Cancel

2. Configure the following parameters as prompted:

   - Select Operation: Save to RDS.
   - VPC Instance, MySQL Database: Select the VPC instance and MySQL database in the current region based on your business requirements.

     Note:

     If your database is in the master mode, you need to manually enter the database name.
   - Account, Password: Enter the account and password to log on to the database. This account should have the permissions to read and write data to the database. Otherwise, rules engine cannot write data to RDS.

     Note:

     After rules engine obtains the account, rules engine only writes data that matches the rule to the database.
   - Table Name: Enter the name of the table that has been built in the database. Rules engine writes the data to the database table.
   - Field: Enter the field name of the database. Rules engine writes the processed data to the field.
   - Value: Enter the value of the field for the database table. You can use the escape character `$`. The format is `${key}`, indicating that the value of key selected from the topic is used as the input value.

     For example, if the SQL statement for rules engine is `SELECT key FROM mytopic`, and RDS has a table that includes a `String` type field with the value `tem`.

     on IoT Platform, enter **`tem`** into **Filed**, and **`${key}`** (the JSON field selected from rules engine) into**Value**.

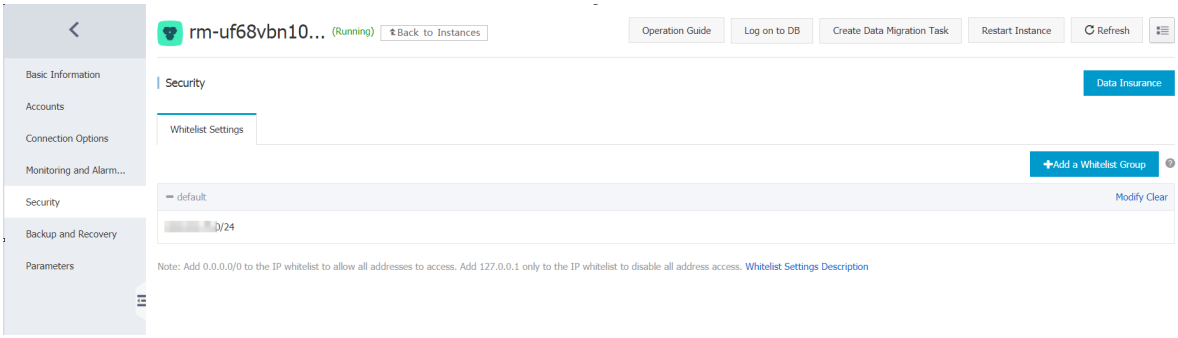
     Note:

     - Make sure that the value is set in the correct format: `${}`. Otherwise, a constant is written to the table.
     - Make sure that the data type of the field is consistent with its value. Otherwise, the data cannot be stored in the database.

3. Once the configuration is complete, rules engine will add the following IP addresses to the whitelist to connect to RDS. If the following IP addresses are not listed, manually add them to the whitelist:

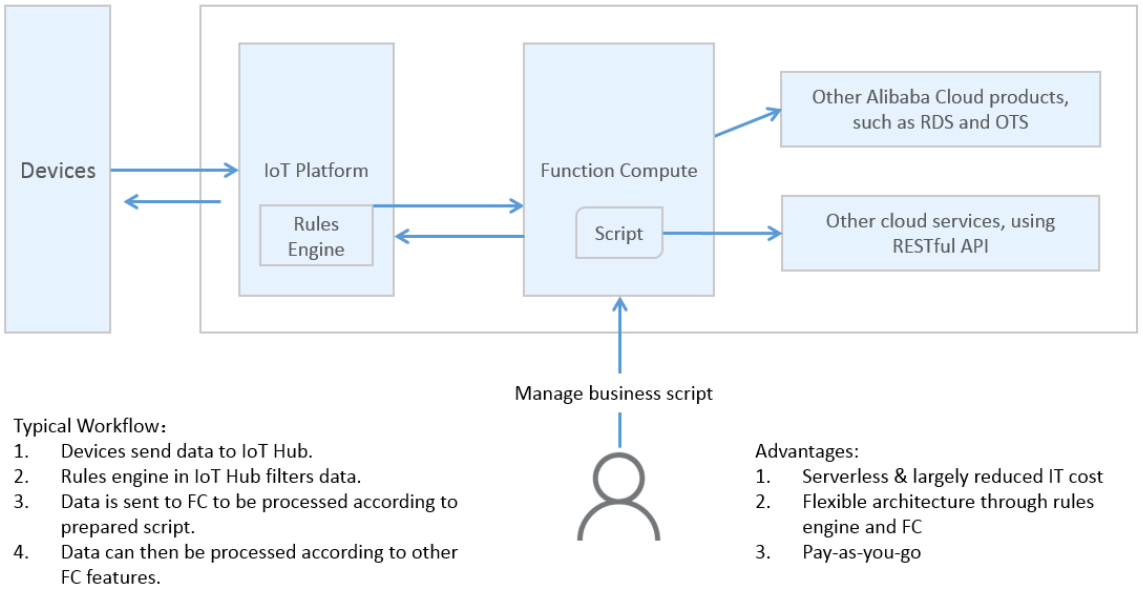
   - China East 2: 100.104.123.0/24
   - Asia Pacific SE 1 (Singapore): 100.104.106.0/24
   - US West 1 (Silicon Valley): 100.104.8.0/24
   - EU Central 1 (Frankfurt): 100.104.160.192/26
   - Asia Pacific NE 1 (Tokyo): 100.104.160.192/26

   The whitelist (example) for the RDS console is as follows:



3.8.4 Forward data to Function Compute

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



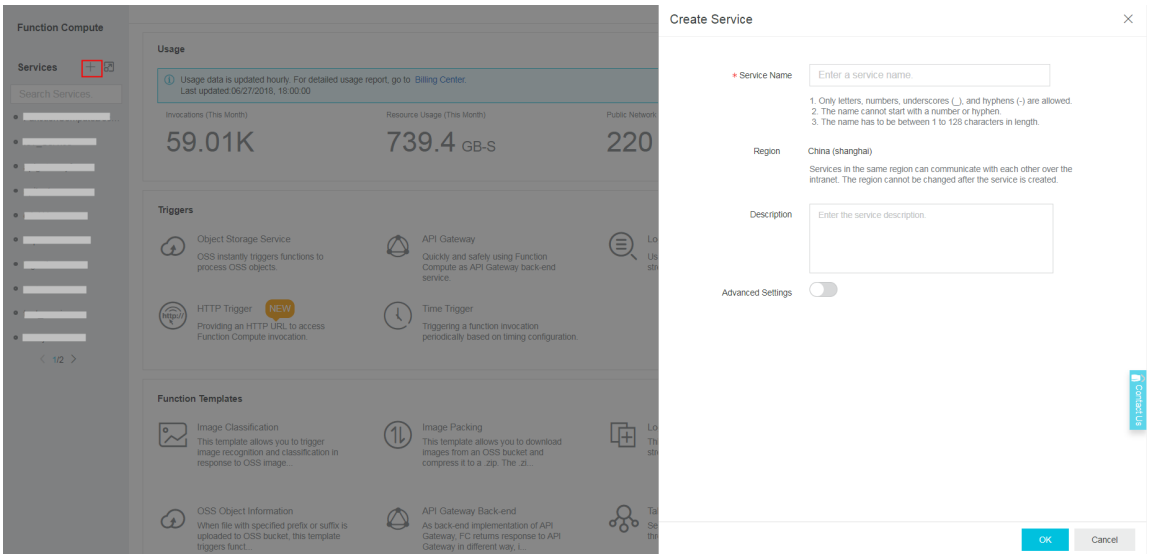
Procedure:

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

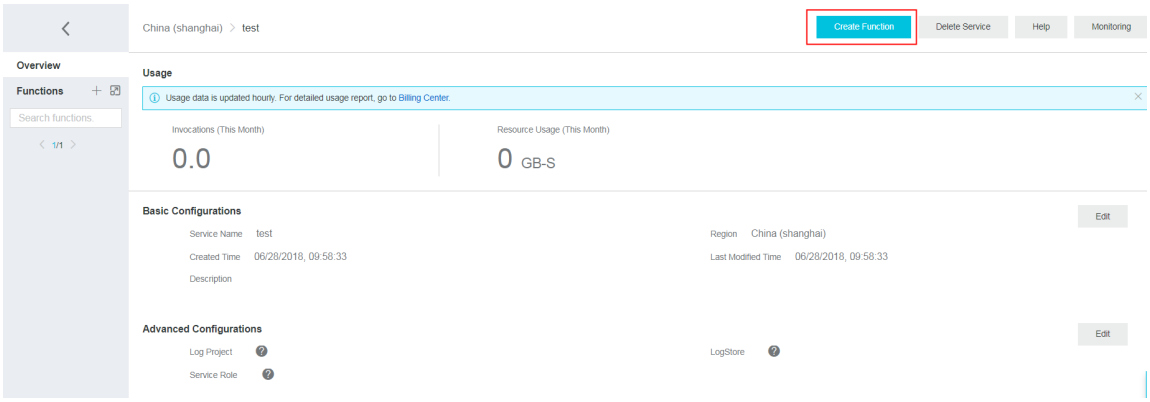
Procedure

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

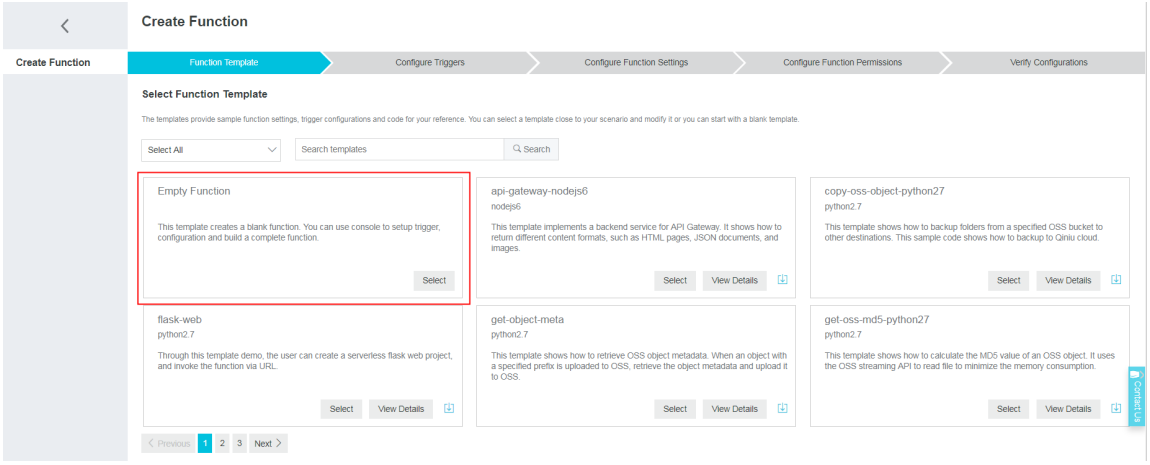
   a. Create a service. **`Service Name`** is required. Configure other parameters as required.



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

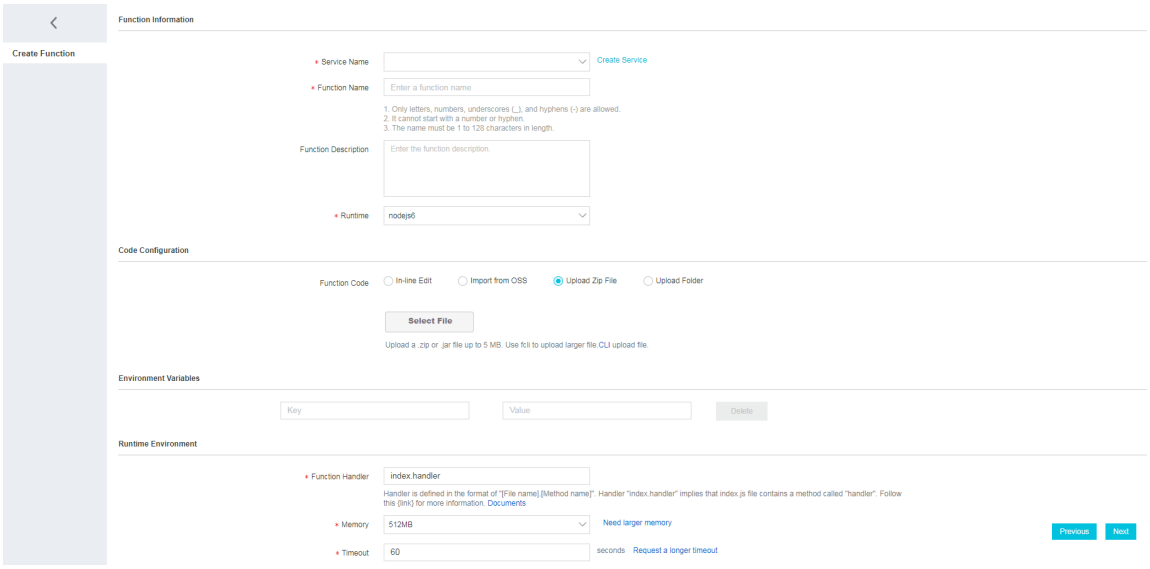


   c. Select a function template. A blank template is used as an example.



d. Set parameters for the function.

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



In the proceeding parameters,

Service Name: Select the service created in *1.a*.

Function Name: Specify the name of your function.

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

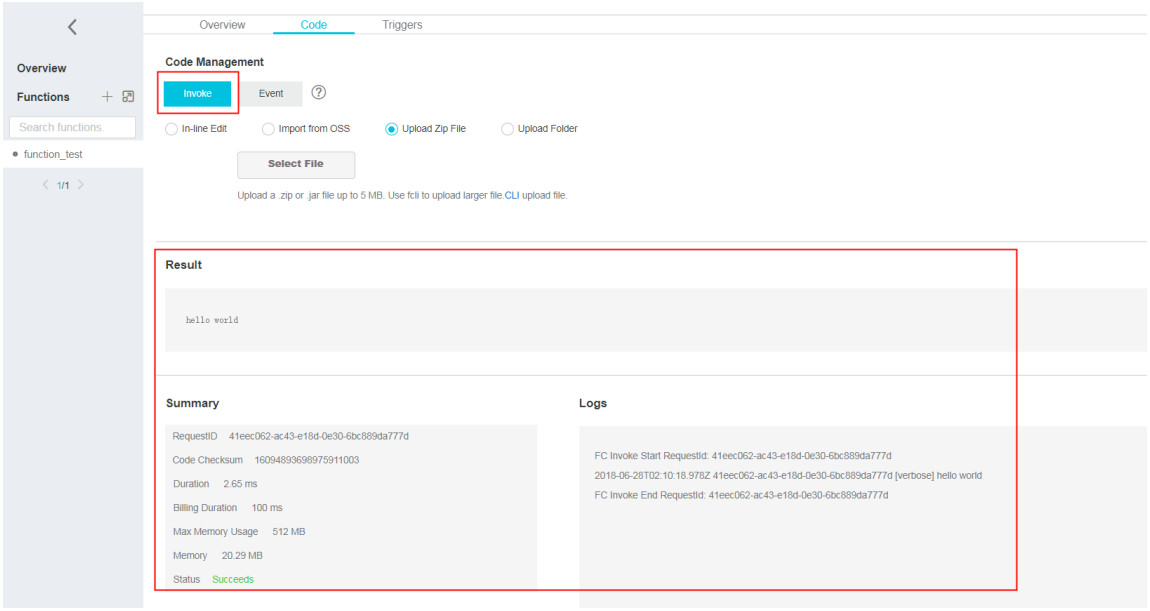
Code Configuration: Upload your code.

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

Configure other parameters as required. For more information, see configurations in *Function Compute*.

e. Verify whether the function runs as intended.

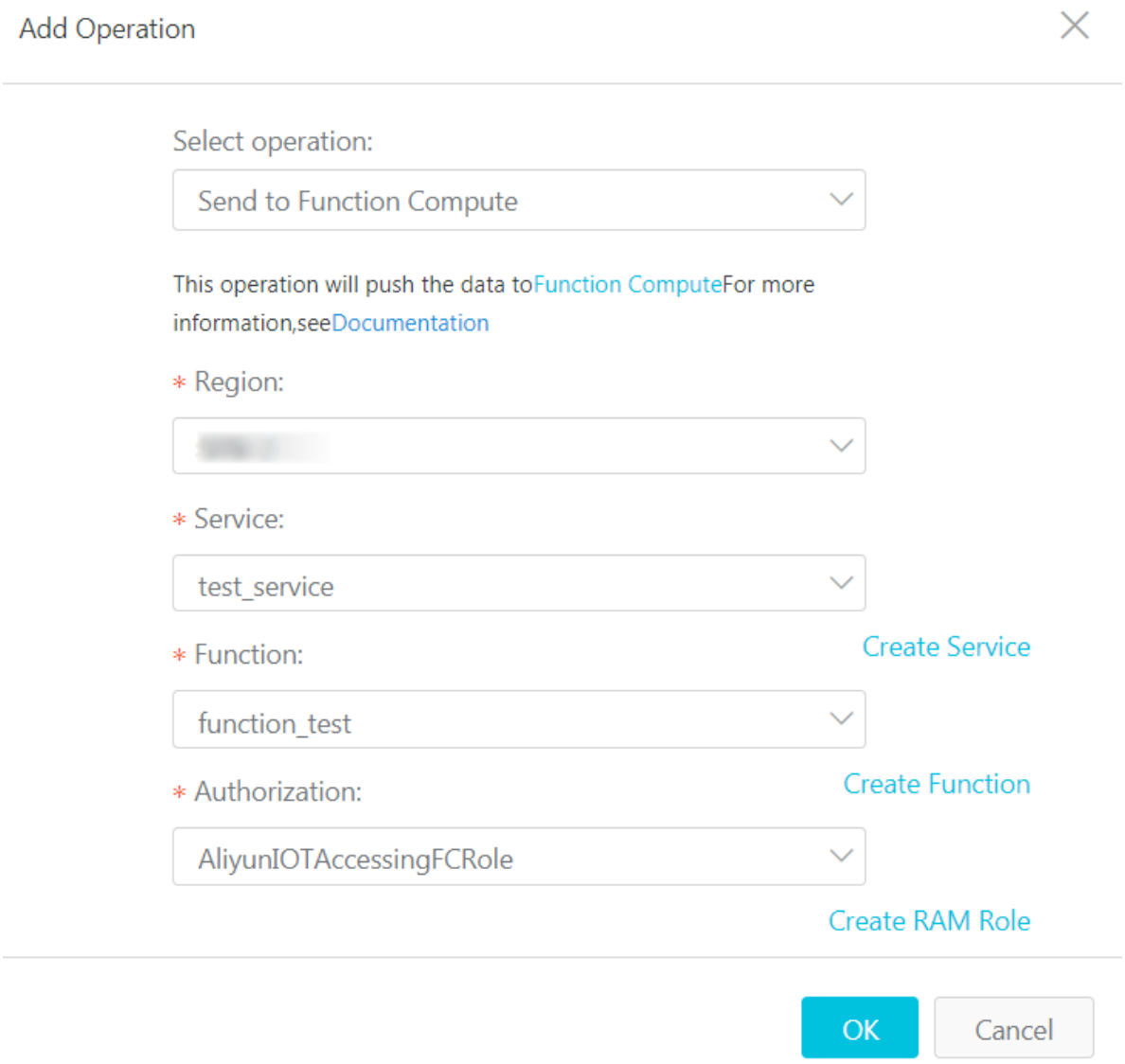
After you create a function, you can run it on the Function Compute console for verification . FC will display information about function output and requests on the Function Compute console.



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

> **Note:**
> Data in JSON and binary formats can be forwarded to FC.

4. Click a rule name to go to the **Rule Details** page.
5. Select **Data Forwarding Add Operation**. On the **Add Operation** page, configure parameters:



- 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 only supported in China East 2.

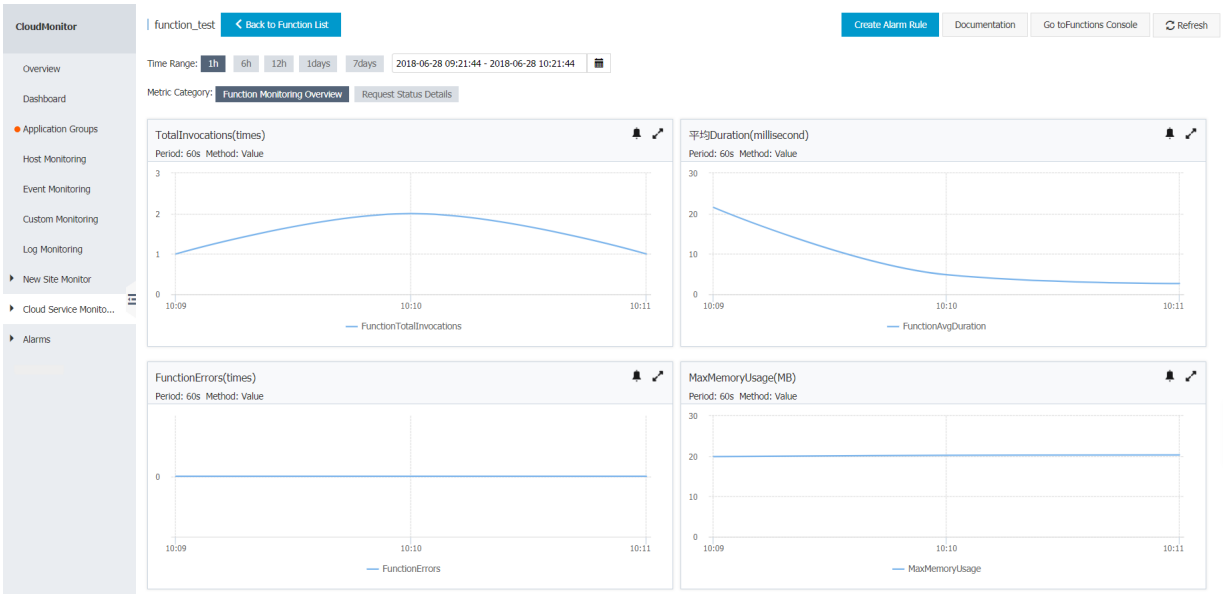
- Service: Select a service based on your region. If there are no services available, click **Create Service**.
- Function: Select a function based on your region. If there are no functions available, click **Create Function**.

- 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.



4 Extended services

4.1 Firmware update

This topic describes how to use the firmware update service in the IoT Platform console.

Currently, the firmware update service is only available in the China (Shanghai) region.

Prerequisites

- Make sure that you have enabled the firmware update service. If you have not enabled the service, log on to the IoT Platform console, select **Extended Services**, and click **Enable Service** under Firmware update.
- By default, the device SDK has enabled the firmware update service.

Context

The firmware update process includes the following tasks:

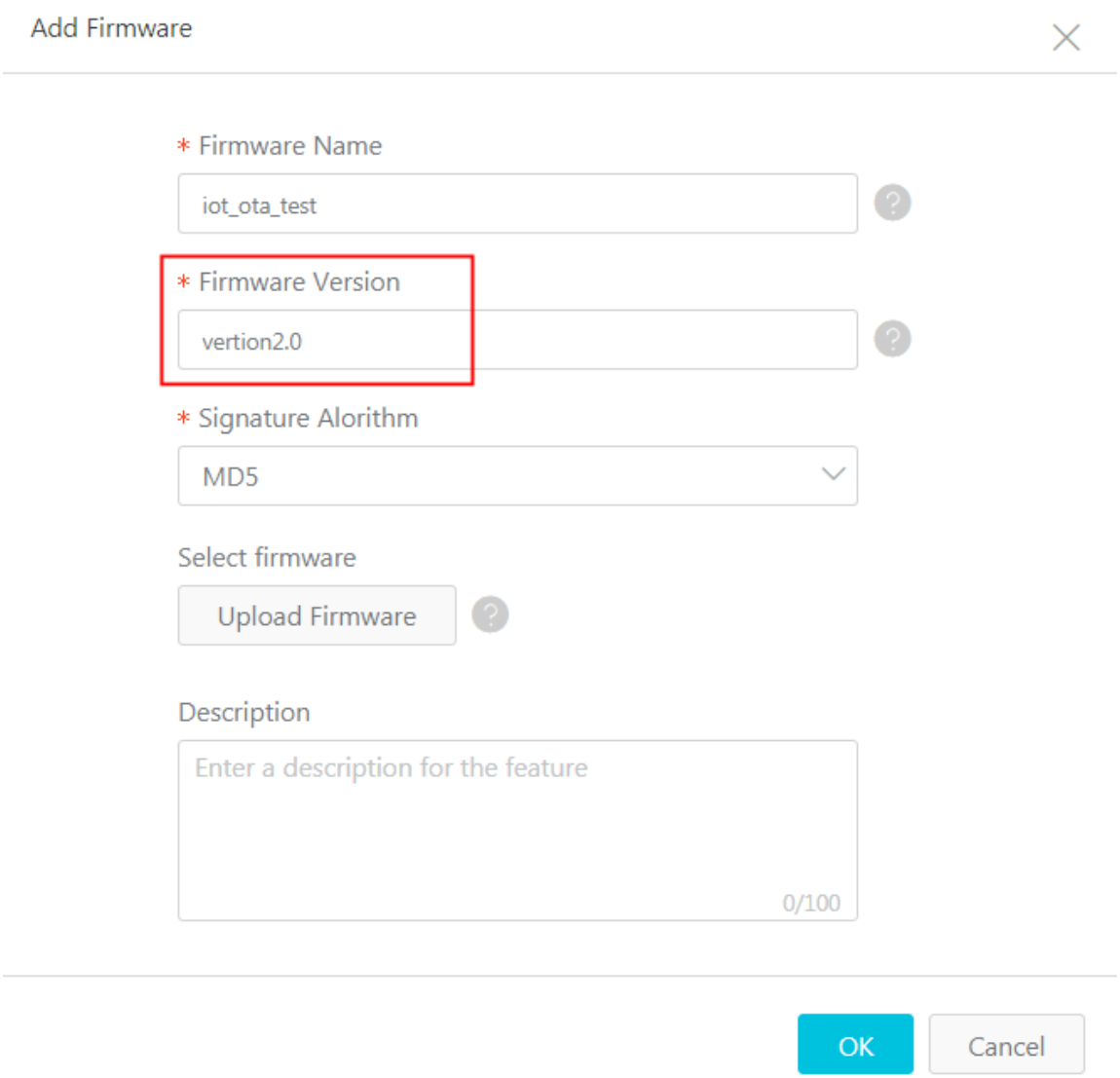
1. Add a firmware.
2. Validate a firmware.
3. Batch update.
4. Update another firmware.

Follow these steps to update a firmware:

Procedure

1. Log on to the IoT Platform console.
2. Select the China (Shanghai) region.
3. Add a firmware.
   a) Select **My Services** > **Firmware update**.
   b) Click **New Firmware** on the **Update Firmware** page, as shown in figure *Figure 4-1: Add a firmware.*.

Figure 4-1: Add a firmware.



c) Set firmware parameters.

- The firmware file name can only contain Chinese characters, English letters, numbers, and underscores (_) and must be 1 to 32 characters in length.
- The size of each firmware file must be no larger than 10 MB.
- You can upload up to 100 firmware files.

> **Note:**
> The 100 firmware files also include firmware files that have been uploaded and then deleted.

4. Validate the firmware.

After you have added the firmware, you must test the firmware on a small number of devices to check whether the firmware runs correctly. If the firmware runs correctly, you can then push the firmware to all devices.

Select a firmware from the firmware list, and click **Validate Firmware**.

- The system then sends a firmware update notification to all devices connected through Message Queuing Telemetry Transport (MQTT). Only online devices will receive the update notification. For offline devices, the system will resend an update notification to these devices when they come online.

  Connections established by using other protocols, such as CoAP and HTTPS, are transient connections. Devices using transient connections cannot receive the firmware update notification when they are offline.
- Firmware verification is to test the firmware on a number of devices. You can validate a firmware multiple times.
- Once you have verified a firmware, the status of the firmware is set to verified, regardless of the update results on the devices.
- The system records all update operations on the devices after the firmware verification process. The system determines that the update process has started only after a device has received the update notification and updated the update progress to the system. The system then changes the update status to Upgrading. After the update process begins, you can view the update progress on the firmware details page.

5. After you have verified and confirmed that the firmware can run correctly, you can then update the firmware on all devcies. Select the firmware from the firmware list, and click **Batch Update**.

Batch update is to push the firmware update notification to a large number of devices.

- You cannot use a firmware that has not been verified to perform a batch update.
- An update is progressive, from the reception of the update notification to the completion of the update. The devices automatically send update information to the OTA system to update their update progress.
- During a batch update, a device may fail to update its firmware if it has not finished the last update task.
- If an update error occurs on a device during the update process, the device sends a notification to the OTA system. The system then sets the update status to Completed

and determines that the device has failed to update the firmware. update errors include downloading failure, verification failure, and extraction failure.

- You can view information about batch upgrades on the firmware details page. The update failure list shows brief information about the cause of the update failures.

When creating a batch update task, if you specify a firmware version that has already been specified in another batch update task for the same product, the system displays a message indicating that an update task conflict has occurred.

For example, you have added firmware versions B and C to the IoT Platform console. You want to update a device with firmware version A for a product. You have also created a batch update task to update the firmware from version A to B. If you try to create another batch update task to update the firmware from version A to C, an update task conflict occurs in the console.

6. If a device failed an update task, you can use the update operation records to re-initiate the update task.

4.2 Remote configuration

Prerequisite

- Make sure that you have enabled Remote Configuration. If you have not enabled this service, log on to the IoT Platform console, select **Extended Services**, and click **Enable Service** under Remote Configuration.
- By default, the device SDK has enabled Remote Configuration. You need to define `FEATURE_SERVICE_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).

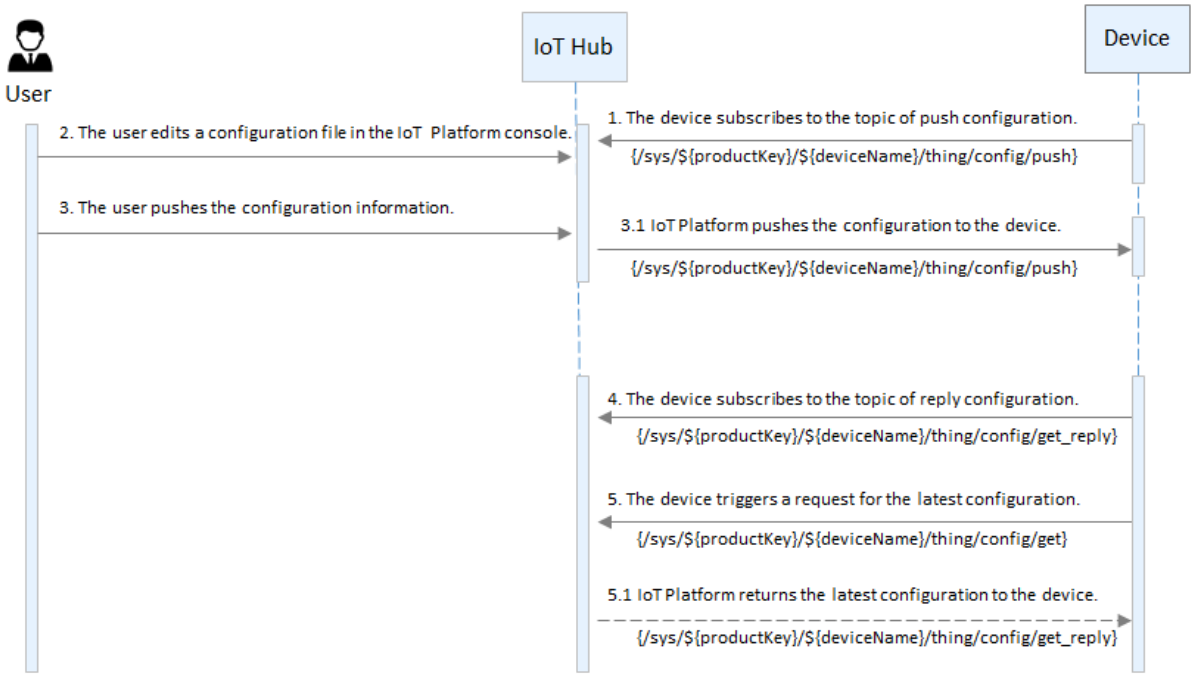
Introduction to Remote Configuration

In many scenarios, developers need to update the device configuration, such as the system parameters, network parameters, and security policies of the devices. Usually, a firmware update is used to complete device configuration update. However, this approach requires more work for firmware version maintenance, and the device must stop running in order to install the update. To fix these issues, IoT Platform provides the Remote Configuration service. This service enables you to complete configuration updates without the need for device restart or service interruption.

With the Remote Configuration service, you can perform the following operations:

- Enable or disable Remote Configuration.
- Edit configuration files online and perform version management.
- Update the configuration information of multiple devices.
- Enable the device to send configuration update requests.

Remote Configuration flow chart:



Remote Configuration consists of the following parts:

- A user edits and saves configuration in the IoT Platform console.
- The user then pushes configuration updates to multiple devices that then update their local configuration file after receiving these updates.
- The device can also send configuration update requests to IoT Platform and perform updates.

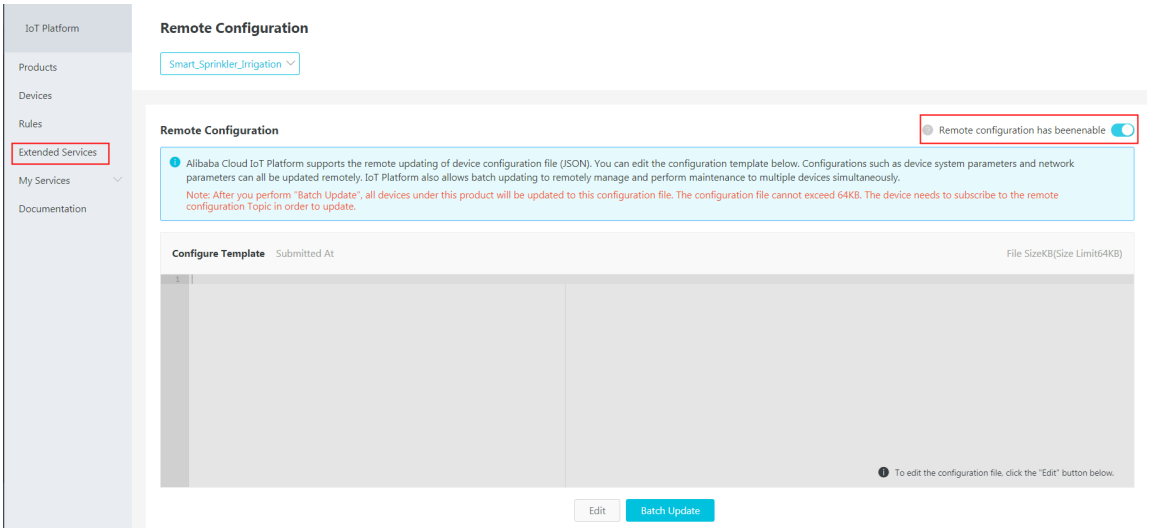
Enable Remote Configuration

Two scenarios are involved when you enable Remote Configuration. One scenario is that IoT Platform sends configuration updates to the device. The other scenario is that the device sends queries about configuration information. The steps to enable Remote Configuration vary based on different scenarios.

Scenario one

If devices receive configuration information from the IoT platform, use the following steps to enable Remote Configuration.

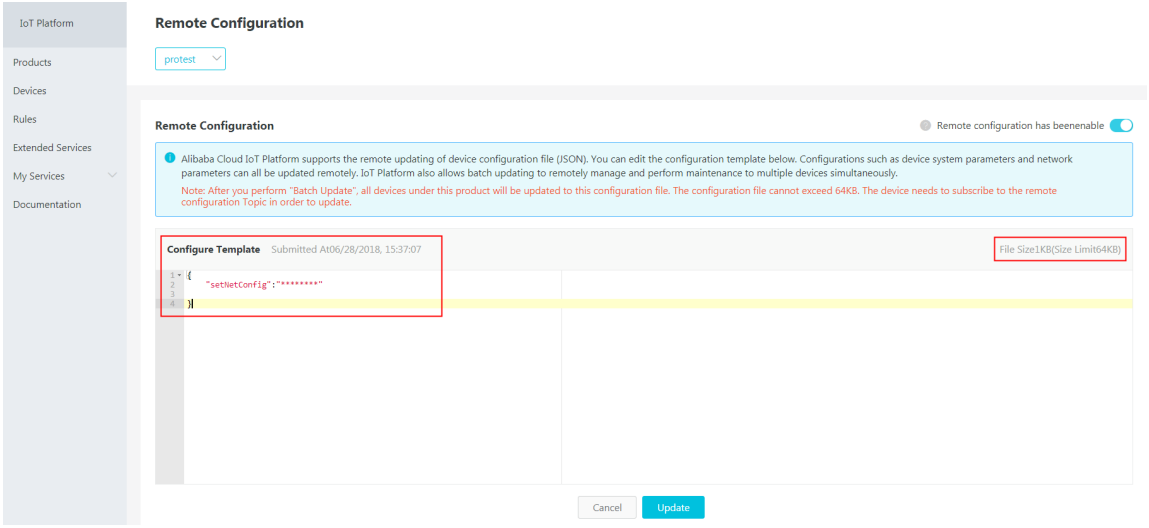
1. When the device is online, configure the device to subscribe to topic (/sys/${productKey}/${deviceName}/thing/config/push) that pushes configuration information.
2. Enable Remote Configuration in the IoT Platform console.
   a. Log on to the IoT Platform console, and select **Extended Services**.
   b. Click **Remote Configuration** to enter the detail page, and click **Enable Service**.
   c. Select a product and click to enable **Remote Configuration**.



Note:

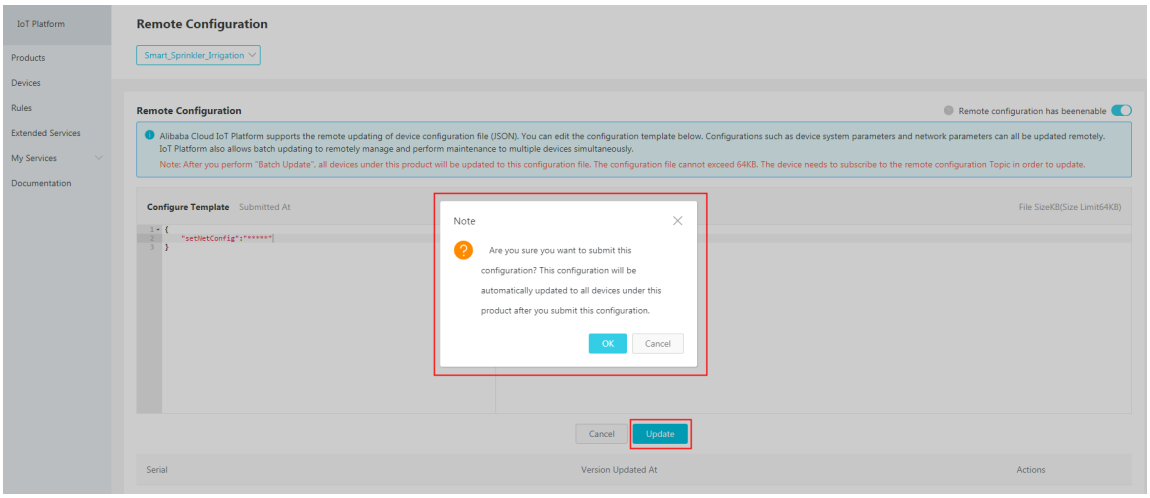
- You must enable Remote Configuration to edit configuration information.
- You can also disable Remote Configuration here.

   d. In the editing area, click **Edit** to edit configuration information. You can also copy configuration information to the editing area. The product configuration template is applicable to all devices under this product. Currently, you cannot update the configuration of individual devices.

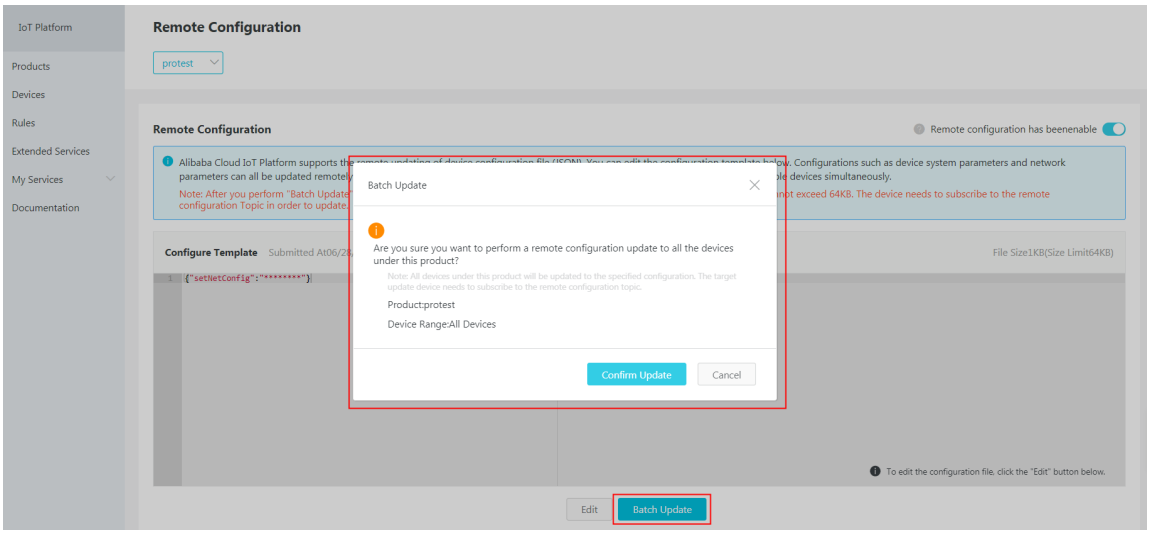


- Remote Configuration supports JSON files. IoT Platform does not have requirements for the configuration content. The system only checks the format of the data when you submit the configuration file. This prevents configuration errors that are caused by format errors.
- The configuration file can be up to 64 KB in size. The file size is dynamically displayed in the upper-right corner of the editing area. Configuration files larger than 64 KB cannot be submitted.

e. After you have finished editing the configuration information, click **Update** to create the configuration file. This allows devices to send requests to update configuration information.



f. After the configuration file has been submitted, IoT Platform does not push updates to devices immediately. You must click **Batch Update** so that the system pushes the updated configuration file to all devices.

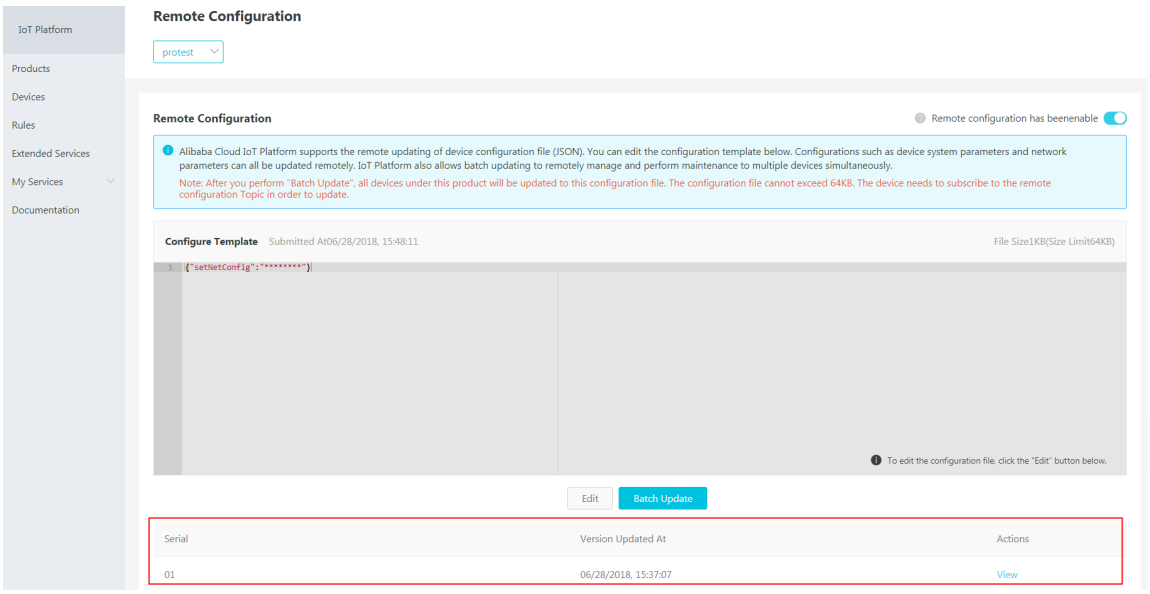


Note:

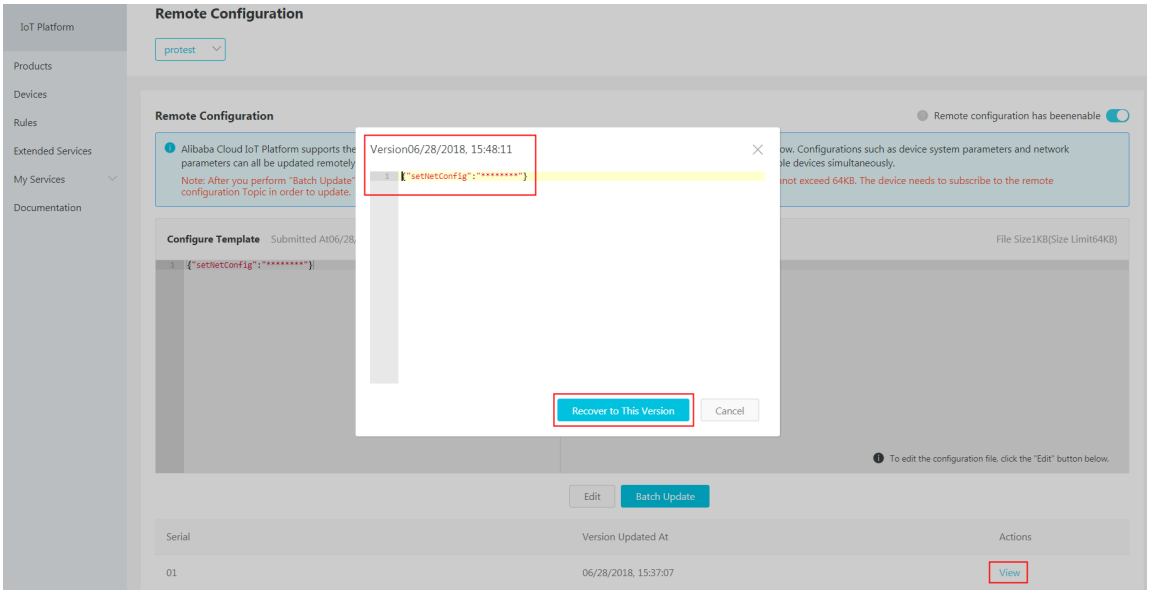
- You can only perform batch update once in an hour. Do not perform batch updates frequently.
- If you want to stop pushing configuration updates, you need to disable Remote Configuration. The system then stops pushing all updates and denies update requests from devices.

g. You can view configuration change history.

Remote Configuration saves the latest five configuration changes by default. After you have submitted a configuration change, the latest configuration is displayed in the version records. You can then view the configuration information and time of update, providing high traceability of records.



Click **View** to view the configuration information of the specified version. Click **Restore to This Version** to copy the configuration information into the editing area so that you can edit and update the configuration.



3. The device automatically updates the configuration after receiving the configuration updates from IoT Platform.

Scenario two

If devices need to send queries about configuration information, use the following steps to enable Remote Configuration.

1. Configure the device to subscribe to topic (/sys/${productKey}/${deviceName}/thing/config/get_reply).
2. Enable Remote Configuration in the IoT Platform console. For more information, see 2.
3. The device call the **`linkkit_invoke_cota_get_config`** operation to trigger the request for remote configuration.
4. The device sends queries about the latest configuration updates through topic (/sys/${productKey}/${deviceName}/thing/config/get).
5. IoT Platform returns the latest configuration information to the device after receiving the queries .
6. The device use the **`cota_callback`** callback to process the configuration file that is sent through Remote Configuration.

5 Log service

This topic describes three types of logs and log details in Log Service.

Usage

There are three types of logs:

- *Device behavior analytics*
- *Upstream analytics*
- *Downstream analytics*

This following table describes the methods for using IoT Platform to filter logs.

Filter method	Description
DeviceName	Specifies the device name. It is the unique identifier of a device for a product. You can filter logs by deviceName.
MessageId	Specifies the message ID. It is the unique identifier of a message on IoT Platform. You can use the messageId to track the entire process of message forwarding.
Status	A log entry has two statuses: success and failure.
Time range	Filters logs based on the time range specified.

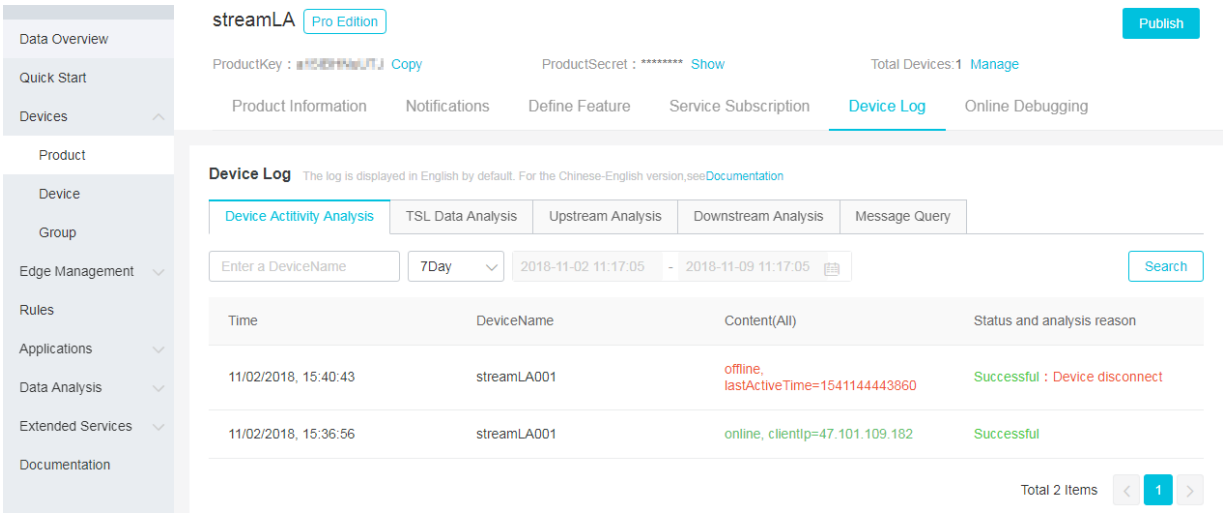
Note:

- `{}` indicates variables. The system will display logs based on the actual running.
- Logs are in English only.
- When logs about failures are displayed, all errors except `system error` are caused by improper use or violations of product restrictions. Troubleshoot these errors carefully..

Device behavior analytics

Device behavior analytics includes the analytics of the online and offline logs for a device.

You can filter logs by DeviceName and time range, as shown in the following figure.



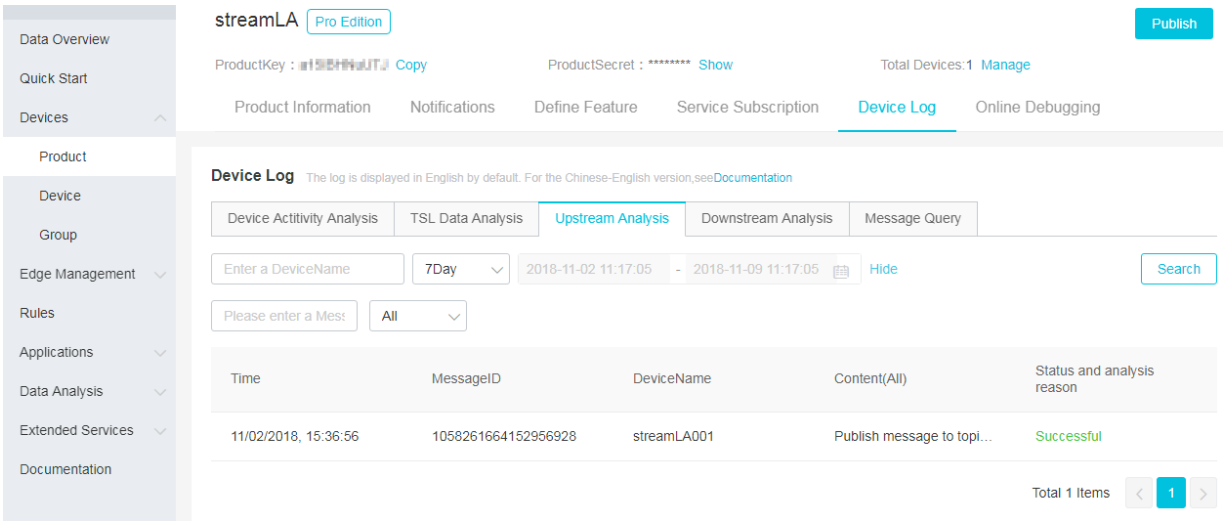
Device connection failure causes

Detail	Description
Kicked by the same device	Another device used the same combination of ProductKey, DeviceName, and ProductKey to come online, and the current device is forced offline.
Connection reset by peer	TCP connection is reset by peer.
Connection occurs exception	Connection exception. IoT server disconnected itself.
Device disconnect	Device sent MQTT disconnection request.
Keepalive timeout	Keepalive timeout. IoT server disconnected.

Upstream analytics

Upstream analytics indicates the analytics of the following processes: A device sends messages to a topic; the topic forwards the messages to rules engine; rules engine forwards the messages to a cloud service.

You can filter logs by DeviceName, MessageId, status, and time range, as shown in the following figure.



Upstream analytics (English and Chinese)

Note:

Upstream analytics includes the context, failure of causes, and cause descriptions.

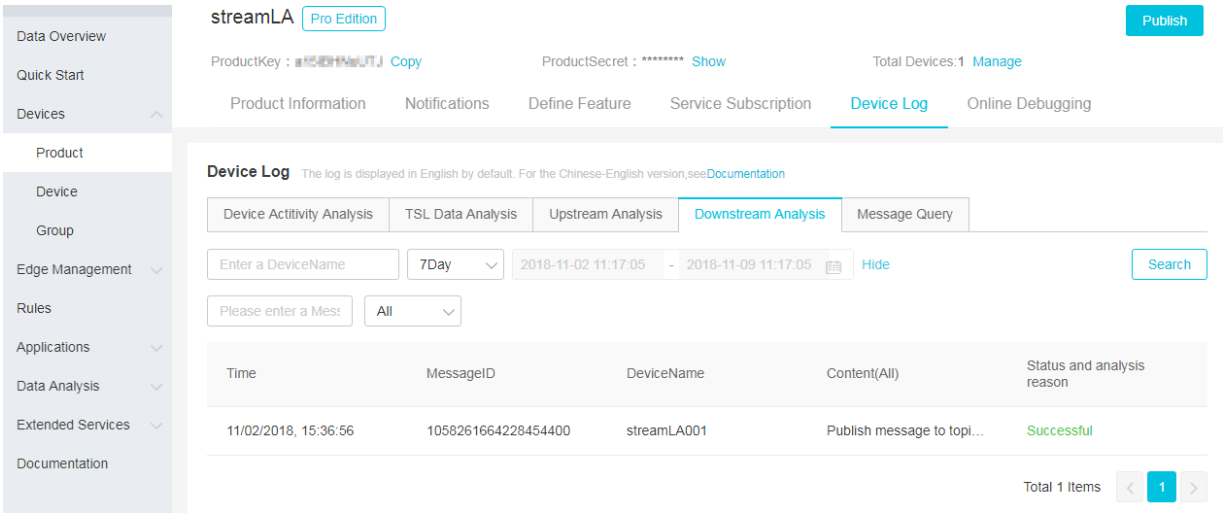
Context	Cause of failure	Cause description
Device publish message to topic:{},QoS={},protocolMessageId:{}	Rate limit:{maxQps},current qps:{}	Restriction violations
	No authorization	No authorization
	System error	System error
send message to RuleEngine ，topic:{} protocolMessageId:{}	{eg ，too many requests}	Causes of connection failure , for example, too many requests for query.
	System error	System error
Transmit data to DataHub, project:{},topic:{},from IoT topic :{}	DataHub Schema:{} is invalid!	Data type mismatch
	DataHub IllegalArgumentException:{}	Parameter exception
	Write record to DataHub occurs error! errors:[code:{}, message:{}]	An error that occurs when data is written to DataHub
	Datahub ServiceException:{}	DataHub exception
	System error	System errors
Transmit data to MNS,queue:{},theme:{},from IoT topic:{}	MNS IllegalArgumentException :{}	MNS parameter exception

	Message Service (MNS) ServiceException:{}	MNS service exception
	MNS ClientException:{}	MNS client exception
	System error	System error
Transmit data to MQ,topic:{}, from IoT topic:{}	MQ IllegalArgumentException :{}	MQ parameter exception
	MQ ClientException:{}	Message Queue (MQ) client exception
	System error	System error
Transmit data to TableStore, instance:{},tableName:{},from IoT topic:{}	TableStore IllegalArg umentException:{}	Table Store parameter exception
	TableStore ServiceException:{}	Table Store exception
	TableStore ClientException:{}	Table Store client exception
	System error	System error
Transmit data to RDS, instance:{},databaseName:{}, tableName:{},from IoT topic:{}	RDS IllegalArgumentException :{}	RDS parameter exception
	RDS CannotGetConnectionE xception:{}	RDS failure of connecting to IoT Hub
	RDS SQLException:{}	RDS SQL statement exception
	System error	System error
Republish topic, from topic:{} to target topic:{}	System error	System error
RuleEngine receive message from IoT topic:{}	Rate limit:{maxQps},current qps:{}	Restriction violations
	System error	System error
Check payload, payload:{}	Payload is not json	Illegal JSON format of Payload

Downstream analytics

Downstream analytics are the logs about messages sent from IoT Hub to your device.

You can filter logs by DeviceName, MessageId, execution status, and time range, as shown in the following figure.



Downstream analytics

Note:

Downstream analytics includes the context, causes of failure, and cause descriptions.

Context	Cause of failure	Cause description
Publish message to topic:{}, protocolMessageId:{}	No authorization	No authorization
Publish message to device, QoS={}	IoT hub cannot publish messages	The server keeps sending messages until QPS reaches the threshold of 50, because it does not receive puback packets from the device.
	Device cannot receive messages	The device fails to receive messages or the server fails to send messages possibly due to the slow network transmissi on speed, or because the server QPS has reached its limit.
	Rate limit:{maxQps},current qps:{}	Restriction violations
Publish RRPC message to device	IoT Hub cannot publish messages	The device does not respond to the server, but the server keeps sending messages until its QPS reaches its limit. Consequently, the server fails to send new messages.

	Response timeout	Response timeout
	System error	System error
RRPC finished	{e.g rrpcCode}	Printed RRPCCode such as UNKNOW, TIMEOUT, OFFLINE and HALFCONN.
Publish offline message to device	Device cannot receive messages	The device fails to receive messages or the server fails to send messages possibly due to the slow network transmissi on speed, or because the server QPS has reached its limit.