ALIBABA CLOUD

Alibaba Cloud

物联网平台

Device Access

Document Version: 20210312

Alibaba Cloud

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 because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should 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 this document based on the "status quo", "being defective", and "existing functions" of its products and services. 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 take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, 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 contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, 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 this document 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 directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice 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.
Warning	A warning notice 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 restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Create a product

When you use IoT Platform, you must first create a product in the console. A product consists of multiple devices of the same type. In most cases, these devices have the same features. For example, you can create a product to represent a hardware product model, and create a device of the product model.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Products**. On the page that appears, click **Create Product**.
4. Specify the parameters and then click **OK**.

Parameter	Description
Product	The name of the product. The product name must be unique within the Alibaba Cloud account. For example, you can enter the product model as the product name. The product name must be 4 to 30 characters in length, and can contain letters, digits, underscores (_), hyphens (-), at signs (@), and parentheses ().
Node Type	The type of devices under the product. Valid values: ○ **Directly Connected Device**: Devices can directly connect to IoT Platform. The devices cannot be attached with sub-devices. The devices cannot be attached to gateways as sub-devices. ○ **Gateway Sub-device**: Devices cannot directly connect to IoT Platform. The devices must connect to IoT Platform by using gateways. For more information about gateways and sub-devices, see Gateways and sub-devices. ○ **Gateway Device**: Devices can directly connect to IoT Platform and be attached with sub-devices. You can use a gateway to manage sub-devices. You can maintain the topological relationships of sub-devices, and synchronize the topological relationships to IoT Platform.
Gateway Connection Protocol	This parameter is available if you set the Node Type parameter to **Gateway Sub-device**. This parameter specifies the communication protocol between sub-devices and gateways. Valid values: ○ Custom: Other standard or private protocols are used. ○ Modbus: The Modbus protocol is used. ○ OPC UA: The OPC UA protocol is used. ○ ZigBee: The ZigBee protocol is used. ○ BLE: The BLE protocol is used.

Parameter	Description
Network Connection Method	The network connection method of directly connected devices or gateway devices. ○ Wi-Fi ○ Cellular (2G/3G/4G) ○ Ethernet ○ Others
Checksum Type	After a device submits Thing Specification Language (TSL) data to IoT Platform, IoT Platform verifies the data based on the specified data verification mode. ○ IoT Platform verifies only the identifier and dataType fields of the data. After verification, all data is forwarded. The data that fails to pass the verification is tagged when the data is forwarded. For more information, see Data formats. In the IoT Platform console, the data is displayed on the **TSL Data** tab of the **Device Details** page. ○ IoT Platform does not verify the data. All data is forwarded. In the IoT Platform console, the data is not displayed on the **TSL Data** tab of the **Device Details** page.
Data Type	The format of upstream and downstream data. Valid values: ○ ICA Standard Data Format (Alink JSON): The JSON-based Alink protocol is provided by IoT Platform for communication between devices and IoT Platform. ○ Custom: If you want to use a custom serial data format, set this parameter to Custom. You must submit a data parsing script in the console to convert upstream custom-format data to Alink JSON data and parse downstream Alink JSON data to custom-format data. This way, devices can communicate with IoT Platform.
Product Description	The description of the product. The description can be up to 100 characters in length.

What's next

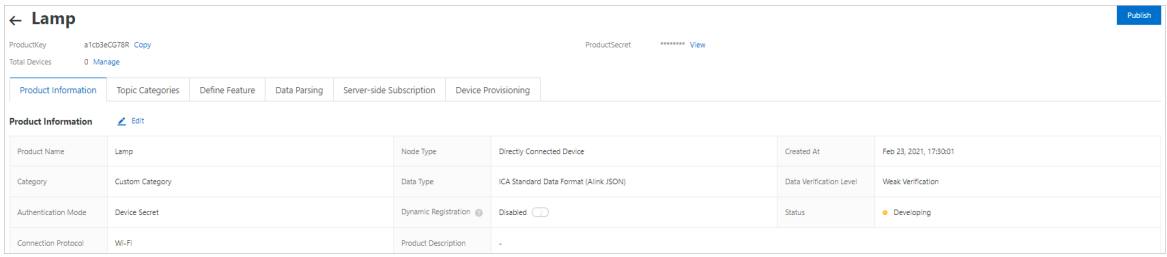
1. Develop the product.

   In the left-side navigation pane, choose **Devices > Products**. In the product list, find the product and click **View** to go to the Product Details page. Click the following tabs to view product details: Product Information, Topic Categories, Edit a topic category, Define Feature, Server-side Subscription, and Data Parsing.

2. Develop a device.

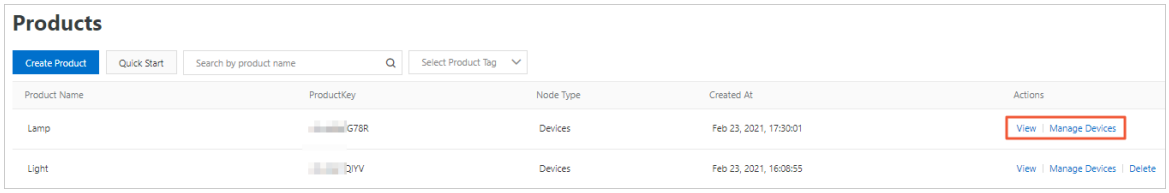
On the Product Details page, click the **Device Provisioning** tab. You can create a device, develop device features, burn a certificate to the device, and authenticate the device when you establish a connection. After you develop the device, you can connect the device to IoT Platform. For more information, see Connect devices to IoT Platform.

3. On the Product Details page, click **Publish** to publish the product.



Before you publish the product, make sure that all parameters of the product are set and devices are developed and debugged.

After you publish the product, the product status changes to **Published**. In this case, you can only view the product information. You cannot modify or delete the product.



You can unpublish the product. To do it, click **Unpublish**.

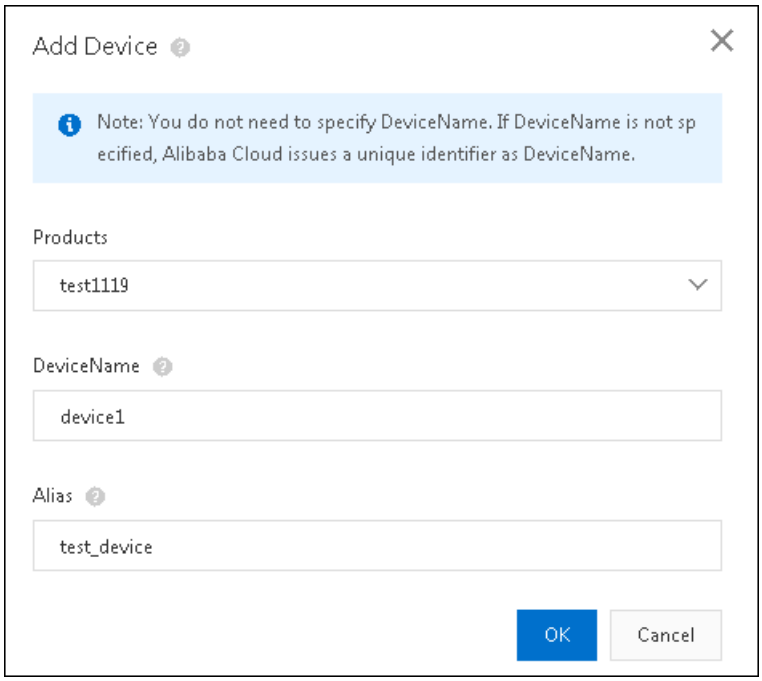
2.Create devices

2.1. Create a device

A product consists of multiple devices of the same type. After you create a product, you must create devices for the product. You can create a device or multiple devices at a time. This article describes how to create a device at a time.

Procedure

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose **Devices > Devices**.
3. On the **Devices** page, click **Add Device**.
4. In the **Add Device** dialog box, set the required parameters, and click **OK**.



Parameter	Description
Products	Select a product. The device derives the features and properties of the specified product. **Note** If the product is registered with another platform, make sure that your account has sufficient activation codes to create the device.

Parameter	Description
DeviceName	The device name. ○ The device name must be unique within the product. ○ The device name must be 4 to 32 characters in length, and can contain letters, digits, hyphens (-), underscores (_), at signs (@), periods (.), and colons (:). **Note** You can leave the DeviceName parameter unspecified. If the parameter is left unspecified, IoT Platform generates a globally unique identifier (GUID) as the device name.
Alias	The alias. The alias must be 4 to 64 characters in length, and can contain letters, digits, and underscores (_).

Result

After a device is created, the **View Device Certificate** dialog box appears. You can view and copy the device certificate information. A device certificate consists of a product key, device name, and device secret. A device certificate is the credential that the device uses to communicate with IoT Platform. We recommend that you keep your device certificates confidential.

Parameter	Description
ProductKey	The key of the product to which the device belongs. It is the GUID that is issued by IoT Platform to the product.
DeviceName	The device name. The identifier of the device. The identifier is unique within the product. DeviceName and ProductKey uniquely identify the device, and are used for authentication with IoT Platform.
DeviceSecret	The device key that is issued by IoT Platform for device authentication and encryption. It must be used in combination with the device name.

What's next

View device information. For more information, see Manage devices.

The status of the created devices is **Not Activated**. Use a Link SDK to implement a device, connect the device to IoT Platform, and then activate the device. For more information, see the Link SDK documentation.

Connect the device to IoT Platform. For more information about best practices, see the following articles:

- 使用MQTT.fx接入物联网平台
- Connect Android Things to IoT Platform

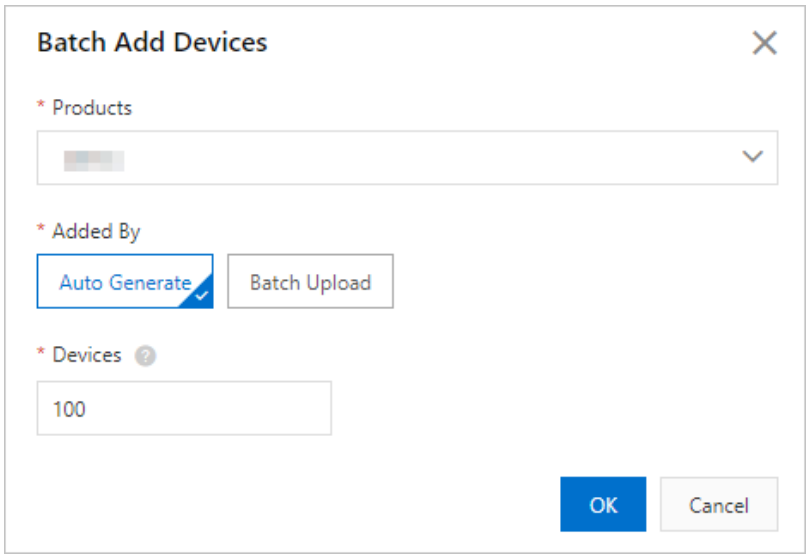
2.2. Create multiple devices at a time

A product consists of multiple devices of the same type. After you create a product, you must create devices for the product. You can create one or more devices at a time. This topic describes how to create multiple devices at a time.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Devices**.
4. On the **Devices** page, click **Batch Add**.
5. Select a product. Each new device derives the features and properties of the product.

> **Note** If the product is associated with another platform, make sure that your account has sufficient activation codes to create the device.



6. Select a method to name devices.
   - **Auto Generate**: You do not need to name devices. After you enter the number of devices, IoT Platform generates names for all devices. Each device name consists of random letters and digits.
   - **Batch Upload**: You must name each device. Click **Download .csv Template** to download a template, specify the DeviceName and Nickname parameters in the template, and then upload the template to the IoT Platform console.

> **Note** You must note the following issues:
>
> - Each device name must be 4 to 32 characters in length and can contain letters, digits, hyphens (-), underscores (_), at sign (@), periods (.), and colons (:). The name of each device that you create for a product must be unique.
> - The Nickname parameter specifies an alias. Each alias must be 4 to 64 characters in length and can contain letters, digits, and underscores (_). This parameter is optional.
> - Each template file can contain a maximum of 1,0000 device names.
> - The maximum size of a template file is 2 MB.

7. Click **OK**.If a template file includes one or more invalid device names, an error occurs. You can click **Download Invalid Device Name List** to download a TXT file. The file includes invalid device names. Modify the invalid device names in the file based on the naming rules of devices. Then,

upload the file again.

8. After the devices are created, click **Download Activation Credential** to download the certificates of the devices. Then, you can burn the certificates on the devices.

Result

The devices are created. You can view the information about the devices and download the certificates of the devices on the **Batch Management** tab of the **Devices** page.

- Find the batch of the devices, and click **Details** to view the details of the devices.
- Click **DownloadCSV** to download the certificates of the devices.

What's next

View device information. For more information, see Manage devices.

The status of the created devices is **Not Activated**. Use a Link SDK to implement a device, connect the device to IoT Platform, and then activate the device. For more information, see the Link SDK documentation.

Connect the device to IoT Platform. For more information about best practices, see the following articles:

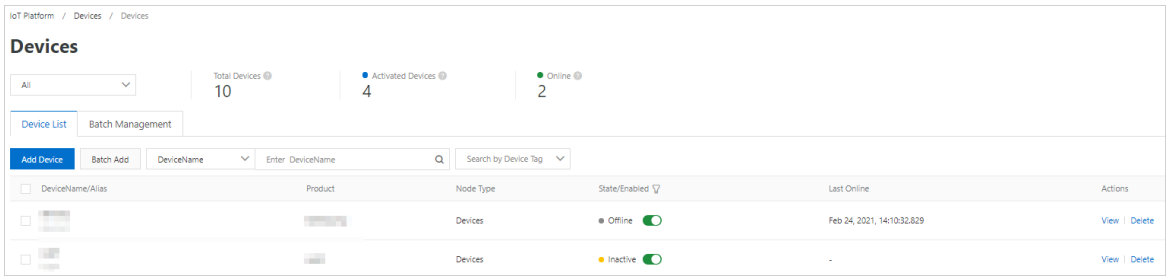
- 使用MQTT.fx接入物联网平台
- Connect Android Things to IoT Platform

2.3. Manage devices

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

Manage the devices of an Alibaba Cloud account

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Devices**. The **Devices** page appears.

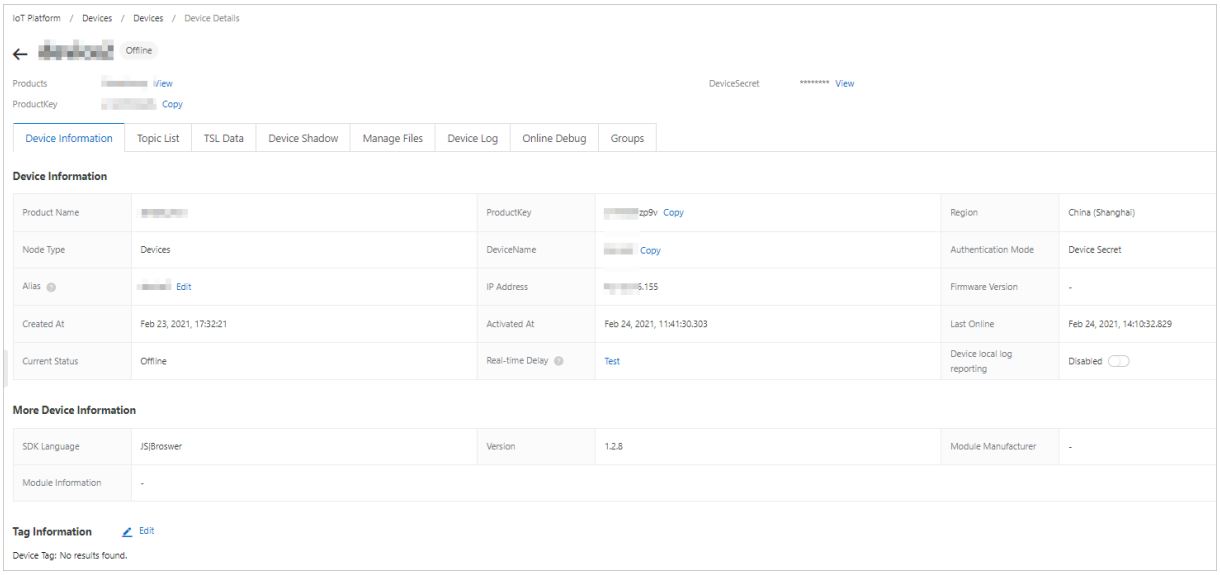


Task	Operation

Task	Operation
View the devices of a product	Select a product in the upper-left corner of the page. You can view the status of each device: ○ Inactive: The device is not connected to IoT Platform. You can download a device SDK to develop the device, connect the device to IoT Platform, and then activate the device. ○ Online: The device is activated and connected to IoT Platform. ○ Offline: The device is activated and disconnected from IoT Platform.
Search for a device	Enter a device name, alias, or tag to search for a device. Fuzzy search is supported.
View detailed information about a device	Find a device and click **View**.
Delete a device	Find a device and click **Delete**. **Note** After a device is deleted, the device certificate becomes invalid and the data about this device is deleted from IoT Platform.

View detailed information about a device

In the device list, find a device and click **View**. The **Device Details** page appears.



Task	Operation
View device information	View the basic information about the device, including device certificate information, firmware information, extended information, and tag information.

Task	Operation
View device data	On the **TSL Data** tab, you can perform the following operations: • Click the **Status** tab to view the current values, data records, and desired values of properties that are submitted by the device. • Click the **Events** tab to view the event records that are submitted by the device. • Click the **Invoke Service** tab to view the service call records of the device.
View device logs	On the **Device Log** tab, view the operational logs of IoT Platform. If you turn on **Device local log reporting**, you can also view the on-premises logs of the device. For more information, see IoT Platform logs and Local device logs.
View device groups	On the **Groups** tab, view the group information of the device. You can click **Add this device to the group** to add the device to an existing group. For more information, see Device group.

Related API operations

Operation	Description
RegisterDevice	Registers a device.
QueryDeviceDetail	Queries the details of a device.
BatchQueryDeviceDetail	Queries the details of multiple devices.
QueryDevice	Queries the devices of a product.
DeleteDevice	Deletes a device.

For more information about API operations related to device management, see API operations for devices.

3.Download device SDKs

IoT Platform provides various device SDKs to simplify the development process and connect devices with IoT Platform.

Prerequisites

The required operations are performed in the IoT Platform console. The device and topic information that is required for device development is obtained.

For more information about the operations, see Create a product, Create a device, Topics, and Add a TSL model.

Develop devices by using device SDKs

To connect a device with IoT Platform, you can integrate an SDK that is provided by IoT Platform in the device. After you develop the device and connect the device to IoT Platform, the device is activated and shows the online status in IoT Platform.

For information about the device SDK for C, see Link SDK for C.

If the provided SDK does not meet your requirements, you can send an email to **linkkitSDK-query@list.alibaba-inc.com**. Use the following template when you write the email:

- Email Subject: *Query about SDK programming language or platform*
- Message Body:

  Company Name:
  Contact:
  Phone Number:
  Programming Language or Platform:
  Requirements:
  Scale of Device Production and Development Plan:

IoT as Bridge SDK

Alibaba Cloud IoT Platform supports communication over MQTT, CoAP, or HTTP. Other types of protocols, such as the fire protection agreement GB/T 26875.3-2011, Modbus, and JT808, are not supported. In some scenarios where devices cannot be directly connected to IoT Platform, you can use the IoT as Bridge SDK to deploy a bridging service and establish connections between the devices and IoT Platform.

For more information, see IoT as Bridge SDK.

Development devices based on the Alink protocol

If the provided device SDK does not meet your requirements, you can develop a custom SDK. For more information, see Alink protocol.

For information about the examples of using open source MQTT clients to access IoT Platform, see the following topics:

- Using Paho MQTT Go client
- Using Paho MQTT C# client
- Using Paho MQTT C client
- Using Paho MQTT Java client

- Using Paho MQTT Android client

4.Authenticate devices

To secure devices, IoT Platform provides certificates for devices, including product certificates (ProductKey and ProductSecret) and device certificates (DeviceName and DeviceSecret). A device certificate is a unique identifier used to authenticate a device. Before a device connects to IoT Hub through a protocol, the device reports the product certificate or the device certificate, depending on the authentication method. The device can connect to IoT Platform only when it passes authentication. IoT Platform supports various authentication methods to meet the requirements of different environments.

IoT Platform supports the following authentication methods:

- Unique-certificate-per-device authentication: Each device has been installed with its own unique device certificate.
- Unique-certificate-per-product authentication: All devices under a product have been installed with the same product certificate.
- Sub-device authentication: This method can be applied to sub-devices that connect to IoT Platform through the gateway.

These methods have their own advantages in terms of accessibility and security. You can choose one according to the security requirements of the device and the actual production conditions. The following table shows the comparison among these methods.

Comparison of authentication methods

Items	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
Information written into the device	ProductKey, DeviceName, and DeviceSecret	ProductKey and ProductSecret	ProductKey
Whether to enable authentication in IoT Platform	No. Enabled by default.	Yes. You must enable dynamic register.	Yes. You must enable dynamic register.
DeviceName pre-registration	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes.
Certificate installation requirement	Install a unique device certificate on every device. The safety of every device certificate must be guaranteed.	Install the same product certificate on all devices under a product. Make sure that the product certificate is safely kept.	Install the same product certificate into all sub-devices. The security of the gateway must be guaranteed.
Security	High	Medium	Medium
Upper limit for registrations	Yes. A product can have a maximum of 500,000 devices.	Yes. A product can have a maximum of 500,000 devices.	Yes. A maximum of 200 sub-devices can be registered with one gateway.

Items	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
Other external reliance	None	None	Security of the gateway.

4.1. Overview

A device must pass identity authentication before it can be connected to IoT Platform. IoT Platform supports device authentication by using device keys.

Use device keys for authentication

When you create a product, set **Authentication Mode** to **Device Secret**. IoT Platform issues a ProductSecret and DeviceSecret or other device keys to each device. When a device is accessing IoT Platform, the device must use the device keys for identity authentication.

IoT Platform supports various authentication methods to meet the requirements of different scenarios.

- Unique-certificate-per-device authentication: A device certificate is burned to each device. The device certificate includes a ProductKey, DeviceName, and DeviceSecret. For more information, see Unique-certificate-per-device authentication.
- Pre-registration unique-certificate-per-product authentication: A product certificate is burned to all devices of a product. The product certificate includes a ProductKey and ProductSecret. For more information, see Unique-certificate-per-product authentication. Enable dynamic registration for the product, and use dynamic registration to obtain a DeviceSecret for a device.
- Preregistration-free unique-certificate-per-product authentication: A product certificate is burned to all devices of a product. The product certificate includes a ProductKey and ProductSecret. For more information, see Unique-certificate-per-product authentication. Enable dynamic registration for the product, and use dynamic registration to obtain a combination of ClientID and DeviceToken instead of a DeviceSecret.
- Sub-device authentication: After a sub-device is connected to IoT Platform by using a gateway, you can use dynamic registration to obtain a DeviceSecret for the sub-device.

These methods have their own benefits in terms of accessibility and security. You can choose a method based on the security requirements of the device and the actual production conditions. The following table shows the comparison among these methods.

Comparison of authentication methods

Item	Unique-certificate-per-device authentication	Pre-registration unique-certificate-per-product authentication	Preregistration-free unique-certificate-per-product authentication	Sub-device authentication
Information burned into the device	ProductKey, DeviceName, and DeviceSecret	ProductKey and ProductSecret	ProductKey and ProductSecret	ProductKey

Item	Unique-certificate-per-device authentication	Pre-registration unique-certificate-per-product authentication	Preregistration-free unique-certificate-per-product authentication	Sub-device authentication
Enable dynamic registration in IoT Platform	Not required. The dynamic registration feature is enabled by default.	Required.	Required.	Required.
Create a device in IoT Platform and register the DeviceName	Required. You must ensure that the specified DeviceName is unique under a product.	Required. You must ensure that the specified DeviceName is unique under a product.	Not required.	Required. You must ensure that the specified DeviceName is unique under a product.
Certificate burning requirement	Burn a unique device certificate to each device. You must ensure the security of each device certificate.	Burn the same product certificate to all devices of a product. You must ensure that the product certificate is safely kept.	Burn the same product certificate to all devices of a product. You must ensure that the product certificate is safely kept.	• A gateway can obtain the ProductKeys of all sub-devices on premises. • Burn the ProductKey of each sub-device on the gateway.
Security	High	Medium	Medium	Medium
Upper limit for registrations	A product can have a maximum of 500,000 devices.	A product can have a maximum of 500,000 devices.	A product can have a maximum of 500,000 devices.	A maximum of 1,500 sub-devices can be registered under a gateway.
Other external reliance	N/A	N/A	N/A	Security of the gateway.

4.2. Unique-certificate-per-device authentication

If the unique-certificate-per-device authentication method is used, you must install a unique device certificate on each device in advance. The device certificate includes a product key, device name, and device secret. When you connect a device to IoT Platform, IoT Platform authenticates the device certificate. After the device passes authentication, IoT Platform activates the device to enable data communication between the device and IoT Platform.

Context

The unique-certificate-per-device authentication method is recommended because of its high level of security.

Procedure:

Procedure

1. Create a product.Create a product in the IoT Platform console. For more information, see Create a product.
2. Add a device.

   Add a device to an existing product and obtain the device certificate information. For more information, see Create a device and Create multiple devices at a time.
3. Burn the device certificate information onto the device.
   i. Download a Link SDK.
   ii. Initialize the Link SDK. Specify the device certificate information in the Link SDK.Initialize the Link SDK in which the unique-certificate-per-device authentication method is specified. For more information, see the device authentication, and authentication and connection articles of language-specific Link SDKs in the Link SDK documentation.
   iii. Develop the device SDK based on your business requirements. For example, you can develop the following features: over-the-air (OTA) update, sub-device connection, Thing Specification Language (TSL) model, and device shadows.
   iv. Burn the developed device SDK to the device on the production line.
4. Connect the device to IoT Platform.After you power on the device and connect the device to IoT Platform, the device submits an authentication request that includes the device certificate information to IoT Platform. For more information, see Establish MQTT connections over TCP, Connect devices to IoT Platform over CoAP, and Establish connections over HTTP.
5. Activate the device in the IoT Platform console.IoT Platform authenticates the device certificate. After the device passes authentication and connects with IoT Platform, the device can publish messages to topics and subscribe to topic messages. This enables messaging between the device and IoT Platform.

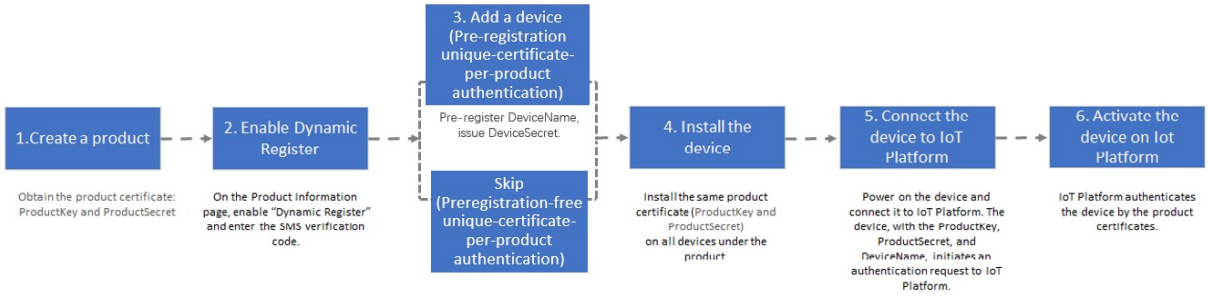
4.3. Unique-certificate-per-product authentication

If you use unique-certificate-per-product authentication, the same firmware is burned to all devices of a product. The firmware includes the same product certificate information (ProductKey and ProductSecret). When a device initiates an activation request, IoT Platform authenticates the device. If the device passes the authentication, IoT Platform sends the information that the device requires to connect with IoT Platform.

Context

> **Note**
>
> - If you use this authentication method, the product certificate may be disclosed because all devices of a product have the same firmware. On the **Product Details** page, you can turn off the Dynamic Registration switch to reject authentication requests from new devices.
> - Transport Layer Security (TLS) encryption must be used if you dynamically register the devices based on unique-certificate-per-product authentication. If your device SDK cannot use TLS encryption, you must use the Unique-certificate-per-device authentication method.

The following figure shows the process of unique-certificate-per-product authentication.



You can use unique-certificate-per-product authentication in the following two methods:

- Pre-registration unique-certificate-per-product authentication

  Before you connect a device to IoT Platform, you must register the DeviceName in IoT Platform. We recommend that you use the MAC address, IMEI number, or serial number (SN) as the DeviceName. Then, IoT Platform issues a DeviceSecret to the device.

  After IoT Platform authenticates the device, the device uses the ProductKey, DeviceName, and DeviceSecret to establish a connection with IoT Platform.

  Pre-registration unique-certificate-per-product authentication supports MQTT-based connections.

- Preregistration-free unique-certificate-per-product authentication

  You do not need to pre-register a device in IoT Platform. Instead, you can use an IoT card number as the DeviceName.

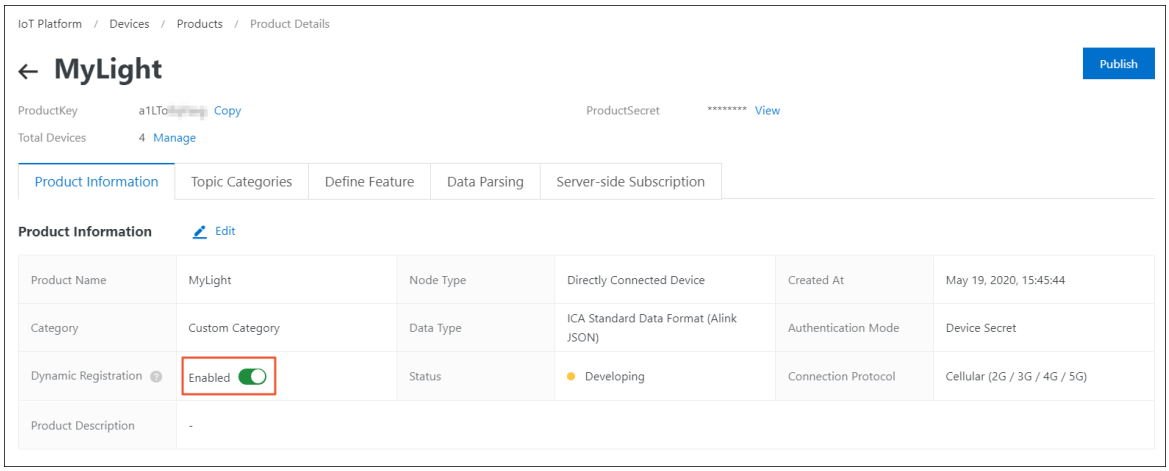
  After IoT Platform authenticates the device, the device uses the ProductKey, DeviceName, ClientID, and DeviceToken to establish a connection with IoT Platform.

  Preregistration-free unique-certificate-per-product authentication supports MQTT-based connections.

Procedure

1. Create a product.Create a product in the IoT Platform console. For more information, see Create a product.
2. Enable dynamic registration.On the **Product Details** page, turn on the Dynamic Registration switch. IoT Platform sends an SMS verification code to confirm your identity.

> **Note** If dynamic registration is disabled when devices initiate activation requests, IoT Platform rejects the requests. Activated devices are not affected.



3. Add a device.
   - Pre-registration unique-certificate-per-product authentication

     Add a device to the created product. For more information, see Create multiple devices at a time or Create a device.

     IoT Platform authenticates the DeviceName when a device initiates an activation request. We recommend that you use an identifier that can be obtained from the device as the DeviceName. The identifier can be the MAC address, IMEI, or SN of the device.

     Then, IoT Platform issues a DeviceSecret to the device. The initial status of the device is Inactive.
   - If you use preregistration-free unique-certificate-per-product authentication, skip this step.
4. Burn the device SDK on the production line.
   i. Download the device SDK.For more information, see Link SDK.
   ii. Initialize the device SDK and enable dynamic registration. In the device SDK, specify the ProductKey and ProductSecret.For more information about how to configure device SDK to use the unique-certificate-per-product method, see the documentations about authentication and connection in Link SDK.
   iii. Develop the device SDK based on your business requirements. For example, you can develop the following features: over-the-air (OTA) update, sub-device connection, Thing Specification Language (TSL) model, and device shadows.
   iv. Burn the developed device SDK to the device on the production line.
5. Connect the device to IoT Platform.Power on the device and connect it to IoT Platform. Then, the device carries the ProductKey, ProductSecret, and DeviceName to initiate an authentication request. For more information, see MQTT-based dynamic registration and HTTP-based dynamic device registration.
6. Activate the device in IoT Platform.
   - Pre-registration unique-certificate-per-product authentication

After IoT Platform authenticates the device, IoT Platform delivers the DeviceSecret that is issued in Step 3 to the device. The device obtains the device certificate (ProductKey, DeviceName, and DeviceSecret). Then, the device can use the certificate to establish a connection with IoT Platform.

> **Note**
> - A device certificate can be used to activate only one physical device.
> - If Device A is activated by using the DeviceName but Device B must use this DeviceName, you can delete Device A from IoT Platform and invalidate the DeviceSecret of Device A. Then, you can use the DeviceName to add and activate Device B.
> - To reactivate a device due to the loss of its DeviceSecret, use the ResetThing API operation to reset the device, and then reconnect the device to IoT Platform. IoT Platform issues the same DeviceSecret to the device.

- Preregistration-free unique-certificate-per-product authentication

  After IoT Platform authenticates the device, IoT Platform issues the ClientID and DeviceToken to the device. Then, the device uses the ProductKey and ProductSecret, ClientID, and DeviceToken to establish a connection with IoT Platform.

  > **Note** A maximum of five physical devices can be activated in IoT Platform with the same ProductKey, ProductSecret, and DeviceName. Each device has a unique ClientID and DeviceToken.

  A DeviceName may be used for multiple physical devices that have different ClientIDs. In this case, the following message appears on the **Product Details** page of the IoT Platform console: The devices of the current product have multiple ClientIDs. You can retain one physical device or clear all physical devices.

  a. On the **Product Details** page, click **View** to view the security-compromised devices of the product.
  b. Choose **Devices > Devices**. On the page that appears, find the device and click **View** to go to the **Device Details** page. The ClientID for the current connection is displayed. Click **Switch** or **Clear** next to the ClientID.
     - **Switch**: Select the ClientID from the drop-down list. Check the first connection time of the device that corresponds to the ClientID, or click **Log Service** and view IoT Platform logs to determine whether the physical device needs to be retained. Then, you can select the ClientID of the physical device that you want to retain, and click **OK**. The physical devices that use other ClientIDs cannot be connected.
     - **Clear**: All physical devices cannot be connected.

5.Devices retrieve certificates

5.1. Overview

Devices can use the following two methods to retrieve device certificates (ProductKey, DeviceName, and DeviceSecret) issued by IoT Platform: 1. Device manufacturers burn certificates on devices. 2. Devices retrieve certificates from the cloud after being powered on and connected to the Internet.

- Burn device certificates

  Device manufacturers retrieve certificates issued by IoT Platform, and then use production lines to burn certificates on devices. After being powered on and connected to the Internet, devices use the certificates to access IoT Platform. The solution requires device manufacturers to transform production lines to burn certificates.

  For more information about this solution, see Burn certificates on devices.

- Retrieve certificates from the cloud

  After being powered on and connected to the Internet, devices automatically retrieve the IP address, and connect to the cloud server of device manufacturers to retrieve certificates, and then connect to IoT Platform. During production, device manufacturers do not need to burn certificates on devices. This solution eliminates the need of certificate burning on production lines and speeds up the mass production of devices.

  For more information about this solution, see Devices retrieve certificates from the cloud.

5.2. Burn certificates on devices

This topic describes how to use production lines to burn certificates (ProductKey, DeviceName, and DeviceSecret) on devices.

The solution requires device manufacturers to transform production lines based on business needs. This topic only describes the burning methods that are available.

Retrieve device certificates

When you create devices, the system automatically generates the device certificates. You can use one of the following methods to retrieve device certificates and write the certificates to databases or files.

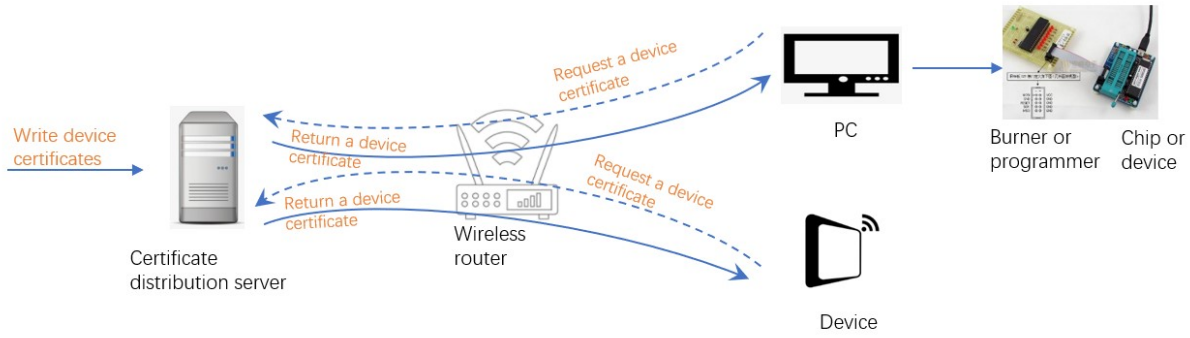
- Use the IoT Platform console to create a device and view the device certificate.
  - After a device is created, the **The devices have been added.** dialog box automatically appears. Click **Learn More** or **Copy Device Certificate** to retrieve the device certificate.
  - On the **Device List** tab, find the required device and click **View**. On the **Device Details** page, click the **Device Information** tab to view the device information.
- Use the IoT Platform console to create multiple devices and view the device certificates.
  - After devices are created, the **The devices have been added.** dialog box appears. Click **Download Device Certificate** to download the device certificates.
  - On the **Devices** page, click the **Batch Management** tab. On this tab, click **DownloadCSV** to download the certificates of all devices under the product.
- Call API operations to create devices. IoT Platform returns the generated device certificates to your application.

> **Note** For more information about how to create devices, see the following topics:
>
> 1. For information about how to create a product, see Create a product.
> 2. Topics about how to create devices:
>   - For information about how to create a device in the IoT Platform console, see Create a device.
>   - For information about how to create multiple devices in the IoT Platform console, see Create multiple devices at a time.
>   - You can call API operations to create devices. IoT Platform provides the RegisterDevice operation to create a single device and the BatchRegisterDevice and BatchRegisterDeviceWithApplyId operations to create multiple devices. For information about how to retrieve the SDK and call the API operations, see Download SDKs.

Burn certificates

After retrieving device certificates, you can start a server on your production line to distribute the device certificates. Programmers, burners, or devices can apply for certificates to the certificate distributor and burn the certificates on the NVRAM or Flash of the devices.

Two certificate burning methods are available. You can use either of the burning methods based on your needs. The following figure shows the procedure.



Two burning methods are described as follows:

- Use programmers or burners to burn device certificates.

  You need to modify the existing programmers or burner programs. Use PCs to apply for device certificates to the certificate distributor and then use programmers or burners to burn the certificates on chips or devices.

  In this solution, multiple burners or programmers must be deployed on a production line to burn certificates. You can increase or decrease the number of burners or programmers based on the scale of device production.

- Enable devices to directly retrieve certificates.

  You need to enable device firmware to automatically detect whether valid certificates exist after devices are powered on. If no invalid certificates exist, devices apply for certificates to the certificate distributor and then write the certificates into the NVRAM or Flash.

  In this solution, you do not need to deploy burners or programmers on your production line. In addition, multiple devices can apply for certificates to the certificate distributor at the same time.

5.3. Devices retrieve certificates from the cloud

In this solution, you do not need to burn certificates on devices. Instead, devices send requests to the server to retrieve certificates (ProductKey, DeviceName, and DeviceSecret) after being powered on and connected to networks.

Procedure

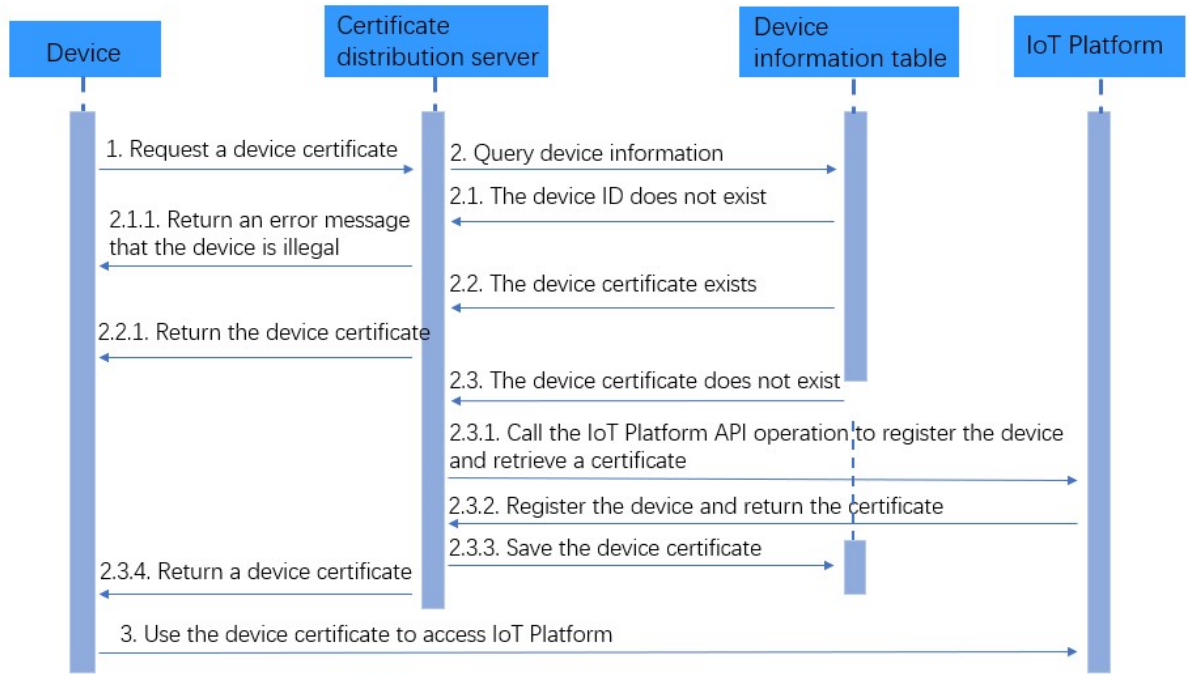
In this solution, you must deploy a device certificate distribution server, and develop the corresponding server API and device information table.

The certificate distribution server calls the API when receiving requests from devices to retrieve certificates. The business logic of this API is described as follows: queries the device information table based on the device ID that is specified in a request, and performs the following operations based on the query result.

- Returns an error message if the device ID is not found in the table. The error message indicates that the device is invalid.
- Returns a device certificate if the device ID is found in the table and the corresponding certificate exists.
- Calls the RegisterDevice API operation of IoT Platform to register the device and returns a device certificate if the device ID is found in the table but the corresponding certificate does not exist.

After retrieving the certificate, the device can use the certificate to connect to IoT Platform.

The following figure shows the procedure.



> **Note**
>
> - Devices must be able to automatically retrieve the IP address and connect to your certificate distribution server.
> - You can develop the certificate distribution server based on your needs.
> - You must ensure the security and reliability of the connections from devices to the certificate distribution server.

Server API

We recommend that you develop the API as follows.

- Request parameters

Parameter	Description
deviceId	The ID of the device. You can specify a MAC address or series number (SN) for this parameter.

- Response parameters

Parameter	Description
productKey	The ProductKey in the device certificate that is issued by IoT Platform.
deviceName	The DeviceName in the device certificate that is issued by IoT Platform.
deviceSecret	The DeviceSecret in the device certificate that is issued by IoT Platform.

Device information table

We recommend that you create the device information table as follows.

Table properties

Table property	Recommended value
Table name	device_table
Time to live (TTL)	-1
Maximum data version	1
Maximum time offset	86400
Primary key	deviceId

Fields

Field	Description

Field	Description
deviceId	The ID of the device. You can specify an MAC address or series number (SN) for this parameter.
registerTime	The time when the device was registered.
activateTime	The time when the device was activated.
productKey	The ProductKey in the device certificate that is issued by IoT Platform.
deviceName	The DeviceName in the device certificate that is issued by IoT Platform.
deviceSecret	The DeviceSecret in the device certificate that is issued by IoT Platform.
IotId	The device ID issued by IoT Platform. This parameter uniquely identifies the device in IoT Platform.

6.Topics

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

6.1. What is a topic?

IoT Platform communicates with devices based on topics. Topics are associated with devices, and topic categories are associated with products.

Topic categories

Topic categories are used to simplify authorization and improve topic-based communication between devices and IoT PlatformA topic category indicates a set of topics. For example, a custom topic category named `/${YourProductKey}/${YourDeviceName}/user/update` includes the `/${YourProductKey}/device1/user/update` and `/${YourProductKey}/device2/user/update` topics.

Log on to the IoT Platform console.Choose **Devices > Products**. Find the product, and click **View** to go to the **Product Details** page. Click the **Topic Categories** tab, and then click **Topics for Basic Communications**, **Topics for TSL Communications**, or **Custom Topics** to view topic categories. For more information about the three types of topic categories, see Topic types.

When you use a topic category, take note of the following items:

- A topic category consists of several fields. Separate these fields with forward slashes (/). A topic category contains the following pre-defined fields: *${YourProductKey}* and *${YourDeviceName}*. Replace the ${YourProductKey} variable with your ProductKey. Replace the ${YourDeviceName} variable with your DeviceName.
- Permissions:
  - **Publish**: Devices can publish messages to the topic.
  - **Subscribe**: Devices can subscribe to the topic to receive messages.

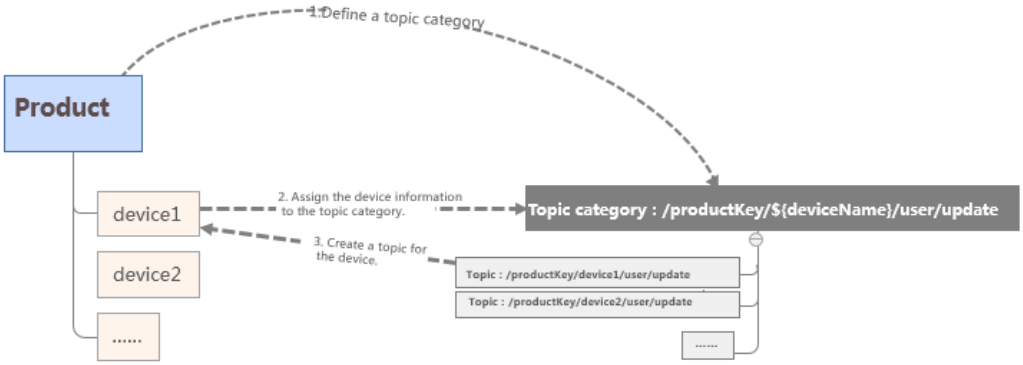
Topics

Topic categories are used to define topics. Topic categories cannot be used for communication. Only topics can be used for communication.

> **Notice** When you debug upstream and downstream communication, make sure that specified topics have the required permissions.

After you create a device under a product, topic categories of the product are automatically mapped to the device to generate topics. You do not need to create topics for the device. Topics are in the same format as topic categories. The difference between topics and topic categories is that the *${YourDeviceName}* variable in topic categories is replaced with an actual DeviceName in topics.

Process of generating topics



The topics of a device can be used for communication only by the device. For example, the `/${YourProductKey}/device1/user/update` topic belongs to a device named Device 1. Only Device 1 can publish messages and subscribe to this topic. Other devices cannot use this topic.

After the device sends a **SUB** command to subscribe to a topic, you can perform the following steps to view the subscribed topic. Log on to the IoT Platform console,and choose **Devices > Devices**. Find the device and click **View**. On the **Device Details** page, click the **Topic List** tab. All subscribed topics are displayed in the **Subscribed Topics** section. IoT Platform can send upstream messages to these topics.

You can click **Post Message** next to a topic of a device and send a message to the device by using the topic. The Post Message option is unavailable for topics that include wildcards. For more information, see A topic that includes one or more wildcards.

A device can send a **UNSUB** request to unsubscribe from a topic. After the topic is unsubscribed, it is removed from the **Subscribed Topics** section.

If you need to disable message sending and receiving features of a device, log on to the IoT Platform console,and disable the device on the **Devices** page. You can also call the DisableThing operation on the server to disable the device. If you do not want to disable the device, you can control the messages that are sent to the device.

Topic types

Topics are categorized into the following three types.

Parameter	Description

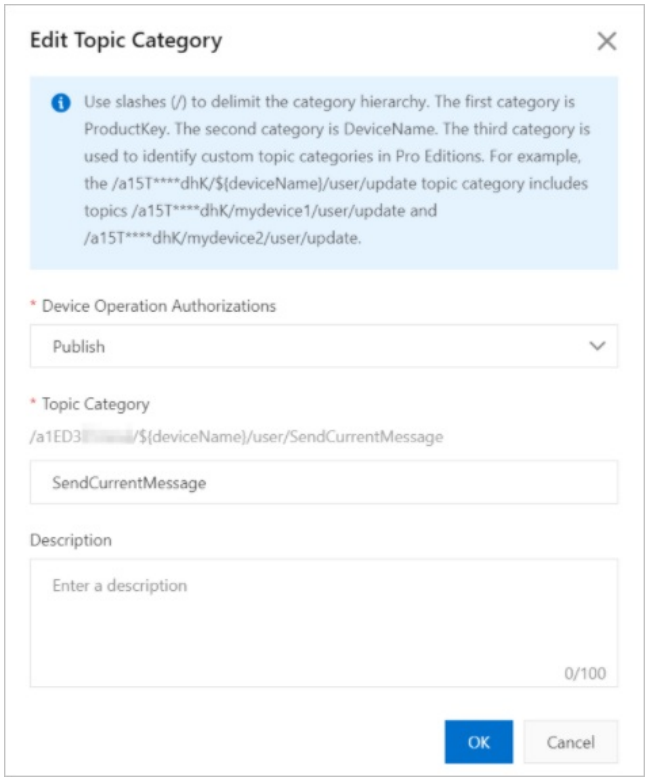
Parameter	Description
Topics for basic communications	Topics for basic communications are predefined in IoT Platform. Topics of this type include the following topics: • OTA update-related topics. For more information about the purpose and data format of each topic, see OTA update. • Device tag-related topics. For more information about the purpose and data format of each topic, see Device tags. • Clock synchronization-related topics. Clock synchronization is implemented by using the Network Time Protocol (NTP) service. For more information, see Configure the NTP service. • Device shadow-related topics. For more information about the purpose and data format of each topic, see Device shadow data stream. • Configuration update-related topics. For more information about the purpose and data format of each topic, see Remote configuration. • Broadcast topics. You can call the PubBroadcast API operation of IoT Platform to broadcast messages to devices that subscribe to a broadcast topic. This allows you to batch control the devices.
Topics for TSL communications	Topics for Thing Specification Language (TSL) communications are predefined in IoT Platform. Topics of this type include the following topics: For more information about the data format of each TSL-based topic, see Device properties, events, and services. **Note** You cannot call the Pub API operation of IoT Platform to send messages to a TSL-based topic. IoT Platform allows you to perform the following operations to control remote devices by using TSL-based topics: 1. Call the SetDeviceProperty or SetDevicesProperty API operation to set the values of device properties. 2. Call the InvokeThingService or InvokeThingsService API operation to invoke device services.
Edit a topic category	You can customize Topic categories on the **Topic categories** page of a product based on your business requirements. For more information, see custom Topic categories. A topic category is a configuration template that is used to generate topics. Each topic category of a product is used by all devices within the product. If you edit and update a topic category, the change applies to all topics that are generated by the topic category. This may affect messaging between the devices that subscribe to these topics and IoT Platform. We recommend that you complete the configuration of topic categories when you implement devices. We also recommend that you do not modify or adjust the topic categories after these devices are connected to IoT Platform.

6.2. Edit a topic category

This article describes how to edit the topic category of a product. Each topic category of a product is used by all devices of the product.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Products**.
4. On the **Products** page, find the product for which you want to edit a topic category, and click **View**.
5. On the **Product Details** page, choose **Topic Categories > Topic Category > Edit Topic Category**.
6. Set the required parameters, and click **OK**.



Parameter	Description
Device Operation Authorizations	The permission of the device for the topic category. Valid values: **Publish**, **Subscribe**, and **Publish and Subscribe**.
Topic Category	The name of the topic category. The name can contain letters, digits, and underscores (_). Each field of the topic category cannot be empty. **Note** If you set the Device Operation Authorizations parameter of a topic category to **Subscribe**, you can specify the + and # wildcards in the topic category. These wildcards allow devices to subscribe to multiple topics at a time. For more information about how to use wildcards, see A topic that includes one or more wildcards.

Parameter	Description
Description	The description.

A topic that includes one or more wildcards

If you set the Device Operation Authorizations parameter to **Subscribe**, IoT Platform allows you to specify two wildcards in a topic. This feature allows a device to subscribe to multiple topics at a time.

Wildcard	Description
#	This wildcard must be specified for the last field in a topic and can match all field values at the current level and sub-levels. For example, you create the `/a1aycMA****/${deviceName}/user/#` topic category. If Device 1 subscribes to the `/a1aycMA****/device1/user/#` topic, the device subscribes to all topics that start with `/a1aycMA****/device1/user/`, for example, `/a1aycMA****/device1/user/update` and `/a1aycMA****/device1/user/update/error`.
+	This wildcard can match all field values at the current level. For example, you create the `/a1aycMA****/${deviceName}/user/+/error` topic category. If Device 1 subscribes to the `/a1aycMA****/device1/user/+/error` topic, the device subscribes to multiple topics, such as `/a1aycMA****/device1/user/get/error` and `/a1aycMA****/device1/user/update/error`.

A topic that includes one or more wildcards represents a set of topics. A device can subscribe to the topic. However, on the **Topic List** page of the device, the **Post Message** action is unavailable for the topic. You cannot send messages to the device by using the topic.

Topic-based communication

IoT Platform calls the Pub API operation to publish messages to a specified topic. Devices receive messages from IoT Platform by subscribing to the topic.

For more information about topic-based communication, see Use custom topics for communication.

6.3. Automatic topic subscription

If you connect devices to IoT Platform over MQTT, you must subscribe to topics before you can receive messages from IoT Platform. Link SDK for C versions 3.1, 3.2, and 4.x that are provided by IoT Platform support automatic topic subscription. This article describes topics that are automatically subscribed by devices.

Background information

IoT Platform communicates with devices based on topics. If you want to use multiple features of IoT Platform, you must subscribe to feature-specific topics. A large amount of time is required to subscribe to topics and then enable devices to work properly. To shorten the time, IoT Platform provides the automatic topic subscription feature. You can use specified topics to send messages from devices without sending requests to subscribe to the topics.

> **Note**
>
> - After devices are connected to IoT Platform by using Link SDK for C version 3.1, 3.2, or 4.x, IoT Platform can send downstream messages by using automatically subscribed topics.
> - If devices are deleted and destroyed, or devices call the aiot_mqtt_unsub operation to unsubscribe from the topics, IoT Platform no longer sends messages to the devices.

Topics

Feature	Topic
TSL-based communication	/sys/${productKey}/${deviceName}/thing/model/down_raw
	/sys/${productKey}/${deviceName}/thing/model/up_raw_reply
	/sys/${productKey}/${deviceName}/thing/event/+/post_reply
	/sys/${productKey}/${deviceName}/thing/deviceinfo/update_reply
	/sys/${productKey}/${deviceName}/thing/deviceinfo/delete_reply
	/sys/${productKey}/${deviceName}/thing/dynamicTsl/get_reply
	/sys/${productKey}/${deviceName}/thing/dsltemplate/get_reply
	/sys/${productKey}/${deviceName}/rrpc/request/+
	/sys/${productKey}/${deviceName}/thing/service/property/set
	/sys/${productKey}/${deviceName}/thing/service/property/get
	/sys/${productKey}/${deviceName}/thing/event/property/history/post_reply
	/sys/${productKey}/${deviceName}/thing/service/+
Sub-device management	/sys/${productKey}/${deviceName}/thing/gateway/permit
	/sys/${productKey}/${deviceName}/thing/topo/change
	/sys/${productKey}/${deviceName}/thing/sub/register_reply
	/sys/${productKey}/${deviceName}/thing/sub/unregister_reply
	/sys/${productKey}/${deviceName}/thing/topo/add_reply
	/sys/${productKey}/${deviceName}/thing/topo/delete_reply
	/sys/${productKey}/${deviceName}/thing/disable_reply
	/sys/${productKey}/${deviceName}/thing/topo/get_reply
	/ota/device/upgrade/${productKey}/${deviceName}

Firmware update Feature	Topic
	/ota/device/request/${productKey}/${deviceName}
Remote configuration	/sys/${productKey}/${deviceName}/thing/config/push
	/sys/${productKey}/${deviceName}/thing/config/get_reply
On-premises communication	/sys/${productKey}/${deviceName}/thing/lan/prefix/get_reply
	/sys/${productKey}/${deviceName}/thing/lan/blacklist/update_reply
	/sys/${productKey}/${deviceName}/thing/lan/prefix/update
Device shadows	/sys/${productKey}/${deviceName}/thing/property/desired/get_reply
	/sys/${productKey}/${deviceName}/thing/property/desired/delete_reply
	/shadow/get/${productKey}/${deviceName}
Reset of device response	/sys/${productKey}/${deviceName}/thing/reset_reply
Response of IoT Platform to device connection	/sys/${productKey}/${deviceName}/thing/awss/enrollee/match_reply
	/sys/${productKey}/${deviceName}/thing/awss/enrollee/checkin
Response of IoT Platform to sub-device connection	/sys/${productKey}/${deviceName}/thing/awss/enrollee/found_reply
	/sys/${productKey}/${deviceName}/thing/cipher/get_reply
	/sys/${productKey}/${deviceName}/thing/awss/device/switchap
Unique-certificate-per-product authentication of sub-devices	/sys/${productKey}/${deviceName}/thing/proxy/provisioning/product_register_reply
Activation of global devices	/sys/${productKey}/${deviceName}/thing/bootstrap/config/push
Downstream notifications	/sys/${productKey}/${deviceName}/_thing/event/notify
Response to devices after data submission	/sys/${productKey}/${deviceName}/_thing/service/post_reply
Task management	/sys/{productKey}/{deviceName}/thing/job/notify
	/sys/{productKey}/{deviceName}/thing/job/get_reply
	/sys/{productKey}/{deviceName}/thing/job/update_reply

7.Protocols for connecting devices

7.1. Use MQTT protocol

7.1.1. MQTT standard

Message Queuing Telemetry Transport (MQTT) is an asynchronous communication protocol based on the TCP/IP protocol stack. MQTT is a lightweight protocol that is used for message transmission in the publish/subscribe mode. MQTT is scalable in unreliable network environments. It is applicable in scenarios where the storage space of device hardware or network bandwidth is limited. The sender and receiver of a message that is transmitted over MQTT are not restricted by time or space. You can connect a device to IoT Platform by using the MQTT protocol.

Supported versions

IoT Platform supports device connection over MQTT. Valid MQTT versions include 5.0, 3.1.1, and 3.1. For more information, see MQTT 5.0, MQTT 3.1.1, and MQTT 3.1.

> **Note** To use the MQTT 5.0 protocol, you must buy an Enterprise Edition instance, and then submit a ticket to apply for the whitelist permission for the instance.

Differences of IoT Platform-based MQTT from standard MQTT

- Supports MQTT messages including PUB, SUB, PING, PONG, CONNECT, DISCONNECT, and UNSUB.
- Supports the clean session flag.
- Does not support will and retained messages.
- Supports quality of service (QoS) 0 and QoS 1 messages and does not support QoS 2 messages.
- Does not support setting the QoS on subscriber clients. Only the QoS that is set by publisher clients is valid.
- Supports the synchronous RRPC mode based on native MQTT topics. A server can call a device service and obtain a response at the same time.

Supported MQTT 5.0 features

Compared with the previous version, MQTT 5.0 provides a large number of new features to improve the performance and ease of use. For more information, see Appendix C. Summary of new features in MQTT 5.0.

IoT Platform supports the following new features of MQTT 5.0:

- Set the maximum length of messages on clients and servers to filter messages.

```
MqttConnectionOptions connOpts = new MqttConnectionOptions();
connOpts.setMaximumPacketSize(1024L);
```

- Set the maximum number of QoS 1 messages per second.

```
MqttConnectionOptions connOpts = new MqttConnectionOptions();
connOpts.setReceiveMaximum(5);
```

- Use the UserProperty parameter to specify a list of properties. This parameter is used to transmit

additional property data. Each property consists of a key and a value.

> **Note** You can add up to 20 properties. Each key cannot start with an underscore (_). The total length of each key and value cannot exceed 128 characters.

```
MqttProperties properties = new MqttProperties();
List<UserProperty> userPropertys = new ArrayList<>();
userPropertys.add(new UserProperty("key1","value1"));
properties.setUserProperties(userPropertys);
```

After a device is connected to IoT Platform by using the MQTT 5.0 protocol, you can view the submitted UserProperty parameter in IoT Platform logs.

- Add the ResponseTopic and CorrelationData parameters to achieve communication in a mode that is similar to HTTP-based request/response.

  For example, the requester is a device, and the responder is your business server. After you use the AMQP subscription or data forwarding feature, you can parse the ResponseTopic and CorrelationData parameters from the property data in the message. Then, you can call the Pub operation to send a response to the device.

  ```
  MqttProperties properties = new MqttProperties();
  properties.setCorrelationData("requestId12345".getBytes());
  properties.setResponseTopic("/" + productKey + "/" + deviceName + "/user/get");
  ```

  > **Note**
  > - The parsed CorrelationData parameter must be Base64 decoded to the byte array that is submitted by the device.
  > - The ResponseTopic or CorrelationData parameter cannot exceed 128 characters in length.

- Add response codes that devices can use to identify request status and problems.

  For more information, see Troubleshooting.

- Scale down message communication topics to integers to reduce the size of MQTT messages. This saves bandwidth resources.

Security levels

- TLS-based TCP connection: high security

  > **Note**
  > - TLS 1.0, 1.1, and 1.2 are supported. We recommend that you use TLS 1.2 for encryption. TLS 1.0 and 1.1 are previous versions and may pose security risks.
  > - Link SDK is configured with TLS 1.2. You do not need to configure the TSL protocol if you use Link SDK.

- Unencrypted TCP connection: low security
- IoT Internet Device ID-based TCP connection (IoT Internet Device ID is a chip-level encryption service): high security

Topic specifications

For more information about the definitions and types of topics, see What is a topic?.

You can view system topics on the Device Details page of the IoT Platform console. For information about feature-specific topics, see the documentation about specific features.

7.1.2. Establish MQTT connections over TCP

This article describes how to establish MQTT connections over TCP by using an MQTT client.

Context

When you configure an MQTT CONNECT message, take note of the following issues:

- If a device certificate (ProductKey, DeviceName, and DeviceSecret) or a combination of ProductKey, DeviceName, ClientID, and DeviceToken is used to connect multiple physical devices, clients may frequently go online and offline. This is because when a new device initiates an authentication request to IoT Platform, the original device is forced to go offline. After the device goes offline, it automatically tries to re-establish a connection.
- In MQTT connection mode, Link SDK automatically tries to re-establish a connection after the device is disconnected. You can view device behaviors by using Log Service.

Connect an MQTT client to IoT Platform

1. Optional. We recommend that you use the TLS protocol for encryption.
   - Link SDK integrates the TLS encryption feature, which eliminates your needs of configuration.
   - If you are developing your own device SDK, you must download the root certificate. For more information about how to use the root certificate, see mbed TLS.
2. Connect the MQTT client to the server. For more information about the connection method, see Open-source MQTT client. For more information about the MQTT protocol, see MQTT documentation.

   > **Note** Alibaba Cloud does not provide technical support for third-party code.

3. Establish an MQTT connection.

   We recommend that you use Link SDK to connect the device to IoT Platform. If you use your own device SDK for connection, you must specify the following parameters.

Endpoint	 The endpoint of a public instance is `${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com:1883`. *${YourRegionId}*: Replace this variable with the region ID of your product. For more information, see Regions and zones.

Variable header: Keep Alive	The CONNECT message must include a keep-alive period. Valid values of the keep-alive time: 30 to 1,200 seconds. Otherwise, IoT Platform rejects the connection. We recommend that you set the keep-alive period to a value that is greater than 300 seconds. If the network connection is unstable, we recommend that you set the keep-alive period to a higher value.
Parameters in an MQTT CONNECT message	◦ Unique-certificate-per-device authentication and pre-registration unique-certificate-per-product authentication: Use the device certificate (ProductKey, DeviceName, and DeviceSecret) to connect the device to IoT Platform. **mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=132323232|"** **mqttUsername: deviceName+"&"+productKey** **mqttPassword: sign_hmac(deviceSecret,content)** ■ mqttClientId: Extended parameters are placed between vertical bars (|). ■ clientId: the ID of the client. We recommend that you use the MAC address or serial number (SN) of the device as the client ID. The client ID cannot exceed 64 characters in length. ■ securemode: the current security mode. Valid values: 2 (direct TLS connection) and 3 (direct TCP connection). ■ signmethod: the signature algorithm. Valid values: hmacmd5, hmacsha1, hmacsha256, and sha256. Default value: hmacmd5. ■ timestamp: the current time, in milliseconds. This parameter is optional. ■ mqttPassword: the password. Calculation method: Alphabetically sort the parameters that are submitted to the server and encrypt the parameters based on the specified signature algorithm. For more information about the signature calculation example, see Examples of signing MQTT connections. ■ content: a concatenated string of the parameters that are submitted to the server. These parameters include productKey, deviceName, timestamp, and clientId. The parameters are sorted in alphabetical order and concatenated without delimiters. Example: Assume that the following values are specified: **clientId=12345, deviceName=device, productKey=pk, timestamp=789, signmethod=hmacsha1, deviceSecret=secret** . The following code shows the parameters in an MQTT CONNECT message that is sent over TCP: **mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=789|** **mqttUsername=device&pk** **mqttPassword=hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();** The encrypted password is a hexadecimal string that is converted from a binary string. The following code shows the result: **FAFD82A3D602B37FB0FA8B7892F24A477F85******

- Preregistration-free unique-certificate-per-product authentication: Use ProductKey, DeviceName, ClientID, and DeviceToken to connect the device to IoT Platform.

  ```
  mqttClientId: clientId+"|securemode=-2,authType=connwl|"
  mqttUsername: deviceName+"&"+productKey
  mqttPassword: deviceToken
  ```

  - mqttClientId: Extended parameters are placed between vertical bars (|).
  - clientId, deviceToken: the ClientID and DeviceToken that are obtained when the device is dynamically registered. For more information, see MQTT-based dynamic registration.
  - securemode: the current security mode. If you use preregistration-free unique-certificate-per-product authentication, set the value to -2.
  - authType: the authentication method. If you use preregistration-free unique-certificate-per-product authentication, set the value to connwl.

Examples

For information about examples of using open-source MQTT clients to access IoT Platform, see the following articles:

- Using Paho MQTT Go client
- Using Paho MQTT C# client
- Using Paho MQTT C client
- Using Paho MQTT Java client
- Using Paho MQTT Android client

MQTT keep-alive

In a keep-alive interval, the device must send at least one message, including ping requests.

If IoT Platform does not receive a message in a keep-alive interval, the device is disconnected from IoT Platform and needs to reconnect to the server.

Valid values of the keep-alive time: 30 to 1,200 seconds. We recommend that you set the keep-alive period to a value that is greater than 300 seconds.

7.1.3. MQTT-based dynamic registration

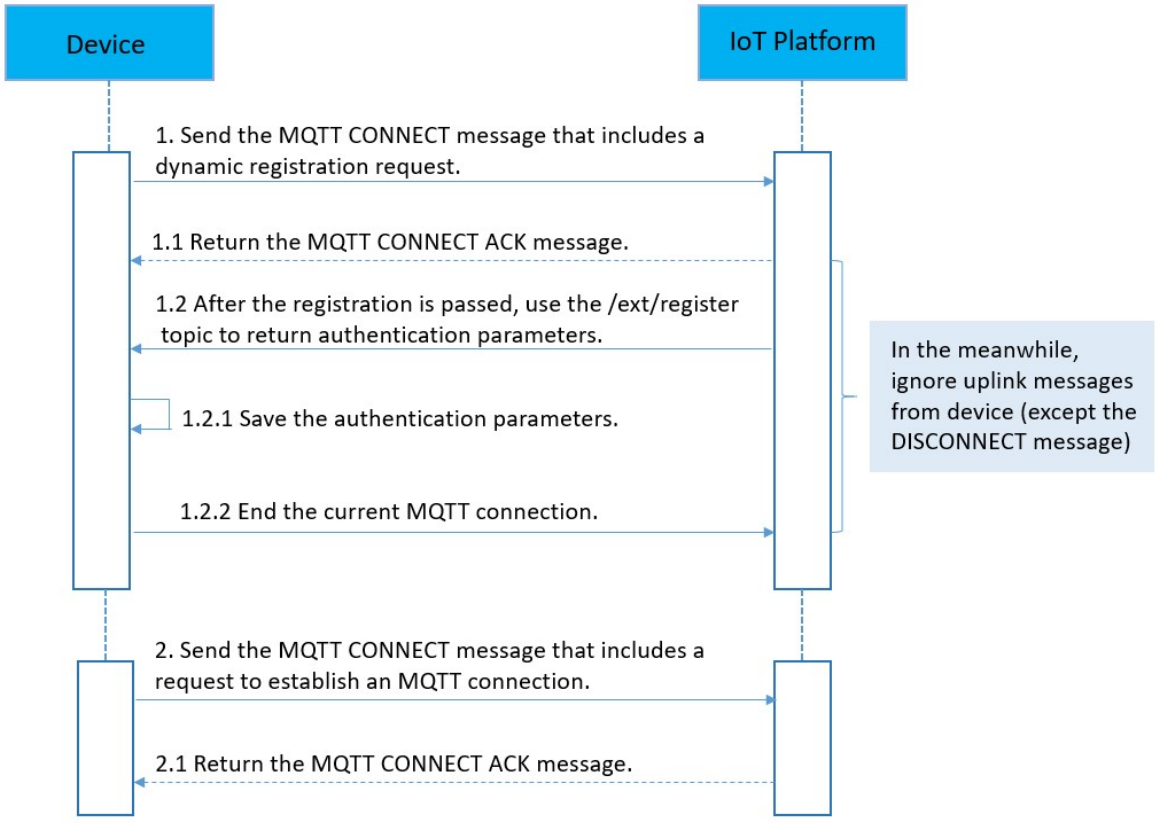
If a device is directly connected to IoT Platform, you can dynamically register the device by using the MQTT protocol. You can use the unique-certificate-per-product authentication method to connect the device with IoT Platform. The device establishes a Transport Layer Security (TLS) connection with IoT Platform to obtain the information that is required for a TCP connection. Then, the device ends the TLS connection and establishes the TCP connection for communication. This article describes the dynamic registration process.

Prerequisites

The following steps that are specified in the Unique-certificate-per-product authentication topic are performed:

1. Create a product.
2. Enable dynamic registration.
3. Add a device.
4. Burn the device certificate to the device.

Dynamic registration process



1. The device sends a CONNECT message that includes dynamic registration parameters to establish a connection.

   > **Note** Dynamic registration supports only TLS-based connections. It does not support direct TCP connections. During dynamic registration, IoT Platform does not verify the keep-alive time of the MQTT connection. Therefore, you do not need to set the keep-alive time.

   - MQTT endpoint:
     -
     - The endpoint of a public instance is `${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com:1883`.
       -
       - Replace the *${YourRegionId}* variable with your region ID. For more information about region IDs, see Regions and zones.
   - Dynamic registration parameters of the CONNECT message:

- If the device belongs to an Enterprise Edition instance and uses the preregistration-free Unique-certificate-per-product authentication method, the dynamic registration parameters in the following example are used:

```
mqttClientId: clientId+"|securemode=-2,authType=xxxx,random=xxxx,signmethod=xxxx,instanceId
=xxxx|"
mqttUserName: deviceName+"&"+productKey
mqttPassword: sign_hmac(productSecret,content)
```

- If the device belongs to a public instance and uses the pre-registration Unique-certificate-per-product authentication method, the dynamic registration parameters in the following example are used:

```
mqttClientId: clientId+"|securemode=2,authType=xxxx,random=xxxx,signmethod=xxxx|"
mqttUserName: deviceName+"&"+productKey
mqttPassword: sign_hmac(productSecret,content)
```

Parameters:

- mqttClientId

  The following table describes the parameters that are included in the mqttClientId parameter.

Parameter	Description
clientId	The ID of the client. The client ID must be 1 to 64 characters in length. We recommend that you use the MAC address or serial number (SN) of the device as the client ID.
securemode	The mode of security. Valid values: ■ 2: the pre-registration unique-certificate-per-product authentication method. For more information, see Unique-certificate-per-product authentication. ■ -2: the preregistration-free unique-certificate-per-product authentication method. For more information, see Unique-certificate-per-product authentication.
authType	The authentication method. Different parameters are returned based on the authentication method. Valid values: ■ register: the pre-registration unique-certificate-per-product authentication method. For more information, see Unique-certificate-per-product authentication. If you set the parameter to this value, DeviceSecret is returned. ■ regnwl: the preregistration-free unique-certificate-per-product authentication method. For more information, see Unique-certificate-per-product authentication. If you set the parameter to this value, DeviceToken and ClientID are returned.
random	The random number. You can specify a random number.
signMethod	The signature algorithm. Valid values: hmacmd5, hmacsha1, and hmacsha256.
instanceId	The ID of the instance. You can log on to the IoT Platform console console, and view the instance ID on the **Instance Overview** page.

- mqttUserName

  Format: `deviceName+"&"+productKey`

  Example: `device1&al123456789`

- mqttPassword

  Calculation method: `sign_hmac(productSecret,content)`

  The value of the content parameter is a concatenated string of the parameters and their values that must be submitted to IoT Platform. These parameters include the deviceName, productKey, and random. These parameters are sorted in alphabetical order and concatenated without using concatenation operators. Then, the value of the content parameter is encrypted based on the algorithm that is specified by signMethod in the mqttClientId parameter. The ProductSecret of the product is used as the secret key of the algorithm.

  Example: `hmac_sha1(h1nQFYPZS0mW****, deviceNamedevice1productKeyal123456789random123)`

2. IoT Platform returns a CONNECT ACK message.

   If 0 is returned, the connection is established and the dynamic registration is successful.

   If the connection fails, you must identify the cause based on the error code that is returned in the ACK message.

   The following table describes the response codes that may be returned after the device sends a connection request to IoT Platform.

Response code	Message	Description
0	CONNECTION_ACCEPTED	The dynamic registration is successful.
2	IDENTIFIER_REJECTED	One or more parameters are invalid. This error may occur due to one of the following causes: ◦ One or more required parameters are not specified or are in invalid formats. ◦ You have established a direct TCP connection for registration. Dynamic registration supports only TLS-based connections.
3	SERVER_UNAVAILABLE	An error has occurred in IoT Platform. Try again later.
4	BAD_USERNAME_OR_PASSWORD	Dynamic registration has failed. The device is not authenticated. Check whether the values of the mqttUserName and mqttPassword input parameters are valid.

3. After the connection is established, IoT Platform uses the `/ext/register` topic to return different authentication parameters based on the authType parameter in the CONNECT message.

   > **Note** The device does not need to subscribe to the topic that is used to push the certificate.

- If you set the authType parameter to register, DeviceSecret is returned.

  The message payload that is pushed by IoT Platform is in the following format:

  ```
  {
   "productKey" : "xxx",
   "deviceName" : "xxx",
   "deviceSecret" : "xxx"
  }
  ```

- If you set the authType parameter to regnwl, ClientID and DeviceToken are returned.

  The message payload that is pushed by IoT Platform is in the following format:

  ```
  {
   "productKey" : "xxx",
   "deviceName" : "xxx",
   "clientId" : "xxx",
   "deviceToken" : "xxx"
  }
  ```

4. The device receives and saves the DeviceSecret or a combination of ClientID and DeviceToken, and ends the current MQTT connection.

   The device can end the current connection by sending a DISCONNECT message or directly ending the TCP connection.

   If the device does not end the connection, IoT Platform disconnects the device after 15 seconds.

   If you are using the Eclipse Paho MQTT client, use the `MqttConnectOptions.setAutomaticReconnect(false)` function to disable automatic reconnection. Otherwise, after the registration succeeds and the TCP connection is ended, a new request of dynamic registration is generated based on the reconnection logic.

5. The device uses the DeviceSecret or a combination of ClientID and DeviceToken to re-initiate a request to establish an MQTT connection between the device and IoT Platform for message communication. For more information, see Establish MQTT connections over TCP.

7.1.4. Establish MQTT over WebSocket connections

IoT Platform supports MQTT over WebSocket connections. You can first use the WebSocket protocol to establish a connection, and then use the MQTT protocol to communicate over the WebSocket connection.

Context

WebSocket provides the following benefits:

- Allows browser-based applications to establish persistent connections with the server.
- Uses port 433, which allows messages to pass through most firewalls.

Procedure

1. Prepare a certificate.

The WebSocket protocol includes WebSocket and WebSocket Secure. WebSocket and WebSocket Secure are used for unencrypted and encrypted connections, respectively. Transport Layer Security (TLS) is used in WebSocket Secure connections. Like a TLS connection, a WebSocket Secure connection requires a root certificate.

2. Develop a client.

   IoT Platform provides MQTT SDK for Java. You can use this client SDK and replace the URL with a URL that is used by WebSocket. For information about how to obtain MQTT SDKs for other programming languages or customize MQTT SDKs, see Open source MQTT client. Before you use MQTT SDKs, read the instructions and check whether WebSocket is supported.

3. Establish a connection with IoT Platform.

   An MQTT over WebSocket connection has a different protocol and port number in the URL from an MQTT over TCP connection. An MQTT over WebSocket connection has the same parameters as an MQTT over TCP connection. Set the securemode parameter to 2 when you use WebSocket Secure. Set the securemode parameter to 3 when you use WebSocket.

   - Endpoint:
     -
     - The endpoint of a public instance is `${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com` format.
       -
       - *${YourRegionId}*: Replace this variable with your region ID. For more information, see Regions and zones.
   - Port number: 443.
   - Variable header: Keep Alive.

     The Keep Alive parameter must be included in the CONNECT packet. Valid values of the keep-alive time: 30 to 1,200 seconds. If the value of the Keep Alive parameter is not in this range, IoT Platform rejects the connection. We recommend that you set the keep-alive period to a value that is greater than 300 seconds. If the network connection is unstable, we recommend that you set the keep-alive period to a higher value.

     In a keep-alive interval, the device must send at least one message, including ping requests.

     If IoT Platform does not receive a message in a keep-alive interval, the device is disconnected from IoT Platform and must reconnect to the server.

   - An MQTT Connect packet contains the following parameters:

     ```
     mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=132323232|"
     mqttUsername: deviceName+"&"+productKey
     mqttPassword: sign_hmac(deviceSecret,content)sign. Sort the content parameters in alphabetical order and sign them by using the signature method.
     content=Parameters sent to the server (productKey,deviceName,timestamp,clientId). Sort these parameters in alphabetical order and splice the parameters and parameter values.
     ```

     Parameters:

     - clientId: the ID of the client. The client ID can be up to 64 characters in length. We recommend that you use a MAC address or serial number (SN).
     - timestamp: optional. The current time in milliseconds.

- mqttClientId: Parameters within the vertical bars (||) are extended parameters.
- signmethod: the signature algorithm.
- securemode: the secure mode. Valid values: 2 (WebSocket Secure) and 3 (WebSocket).

The following examples show MQTT Connect packets with predefined parameter values:

```
clientId=12345, deviceName=device, productKey=pk, timestamp=789, signmethod=hmacsha1, deviceSecret=secret
```

- For a WebSocket connection:
  - Endpoint

    ```
    ws://pk.iot-as-mqtt.cn-shanghai.aliyuncs.com:443
    ```

  - Connection parameters

    ```
    mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=789|
    mqttUsername=device&pk
    mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789
    ").toHexString();
    ```

- For a WebSocket Secure connection:
  - Endpoint

    ```
    wss://pk.iot-as-mqtt.cn-shanghai.aliyuncs.com:443
    ```

  - Connection parameters

    ```
    mqttclientId=12345|securemode=2,signmethod=hmacsha1,timestamp=789|
    mqttUsername=device&pk
    mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789
    ").toHexString();
    ```

We recommend that you use Link SDK to connect devices to IoT Platform. For information about how to develop a custom device SDK for connection, see Examples of creating signatures for MQTT connections.

7.1.5. Examples of creating signatures for MQTT connections

This article provides sample signature code for you to develop your device. This way, your device can communicate with IoT Platform over MQTT without using a device SDK that is provided by IoT Platform.

Description

We recommend that you use a device SDK that is provided IoT Platform. If a device SDK for any programming language is used, you do not need to configure your own signature mechanism. For information about how to obtain the SDK download URL, see Download device SDKs.

If you use other methods to connect your devices with IoT Platform, take note of the following information:

- You must ensure connection stability and maintain the keepalive and reconnection mechanisms for

MQTT connections.

- Alibaba Cloud does not provide technical support for any possible connection issues.
- If you want to use IoT Platform features, such as OTA, TSL models, and unique-certificate-per-product authentication, you must compile your own code to implement these features. This may consume a lot of development time and bug fixing time.

Sample code for signature calculation

If you do not use a device SDK that is provided by IoT Platform, click the following links to view the sample code.

- sign_mqtt.c: the sample code that is used to implement the signature function.
- sign_api.h: the sample code that defines the data structure of the signature function.
- sign_sha256.c: the sample code that defines the algorithm of the signature function. If you already implement the HMACSHA256 algorithm on your own platform, you do not need to compile this code file. However, you must provide the `utils_hmac_sha256()` function that can be called by the *sign_mqtt.c* function.
- Sample code: that sample code that is used to test the signature function.

Description of the signature function

Syntax	`int32_t IOT_Sign_MQTT(iotx_mqtt_region_types_t region,`
	`iotx_dev_meta_info_t *meta,`
	`iotx_sign_mqtt_t *signout);`

Description	Obtains the information that are required to connect a device with IoT Platform based on the specified device identity information. The connection information includes the endpoint, MQTT client ID, MQTT username, and MQTT password. Then, you can provide the information for the MQTT client to connect with IoT Platform.

Input parameters	The following input parameters are included:
	• region: the IoT Platform endpoint to which the device connects.
	Example:
	`typedef enum {`
	`  IOTX_CLOUD_REGION_SHANGHAI,  /* Shanghai */`
	`  IOTX_CLOUD_REGION_SINGAPORE,  /* Singapore */`
	`  IOTX_CLOUD_REGION_JAPAN,  /* Japan */`
	`  IOTX_CLOUD_REGION_USA_WEST,  /* America */`
	`  IOTX_CLOUD_REGION_GERMANY,  /* Germany */`
	`  IOTX_CLOUD_REGION_CUSTOM,  /* Custom setting */`
	`  IOTX_CLOUD_DOMAIN_MAX  /* Maximum number of domain */`
	`} iotx_mqtt_region_types_t;`
	• meta: the identity information of the device.
	Note You must allocate memory for the meta parameter when you call the function.
	Example:
	`typedef struct _iotx_dev_meta_info {`
	`  char product_key[IOTX_PRODUCT_KEY_LEN + 1];`
	`  char product_secret[IOTX_PRODUCT_SECRET_LEN + 1];`
	`  char device_name[IOTX_DEVICE_NAME_LEN + 1];`
	`  char device_secret[IOTX_DEVICE_SECRET_LEN + 1];`
	`} iotx_dev_meta_info_t;`
	Parameters:
	◦ product_key: the ProductKey of the product to which the device belongs.
	◦ product_secret: the ProductSecret of the product to which the device belongs.
	◦ device_name: the DeviceName of the device.
	◦ device_secret: the DeviceSecret of the device.

Output parameters	signout: the output data. The data is used to establish an MQTT connection.
	Example:
	`typedef struct {`
	`  char hostname[DEV_SIGN_HOSTNAME_MAXLEN];`
	`  uint16_t port;`
	`  char clientid[DEV_SIGN_CLIENT_ID_MAXLEN];`
	`  char username[DEV_SIGN_USERNAME_MAXLEN];`
	`  char password[DEV_SIGN_PASSWORD_MAXLEN];`
	`} iotx_sign_mqtt_t;`
	Parameters:
	• hostname: the complete endpoint of the IoT Platform server.
	• port: the port number of the IoT Platform server.
	• clientid: the client ID that must be specified to establish an MQTT connection. We recommend that you use the MAC address or serial number (SN) of the device. The client ID can be a maximum of 64 characters in length.
	• username: the username that must be specified to establish an MQTT connection. The username consists of the DeviceName, Ampersand (&), and ProductKey. Format: `deviceName+"&"+productKey`. Example: `Device1&alSsels****`.
	• password: the password that must be specified to establish an MQTT connection. After you sort the parameters that are submitted to the server by using a dictionary and splice the parameters, use the hmacsha256 method and the DeviceSecret of the device to generate a password.
	For more information, see Establish MQTT connections over TCP.

Response codes	• 0: indicates that the call was successful. • -1: indicates that the call failed because the input parameters are invalid.

Example of using the signature function

In this example, the sample code in sign_test.c is used.

```
#include <stdio.h>
#include <string.h>
#include "sign_api.h"  //Defines all data structures that are used in the signature function.
//The following macros are used to define the device identity information: ProductKey, ProductSecret, DeviceName, and DeviceSecret.
//In actual product development, the device identity information must be encrypted by the device manufacturer and stored in the flash of the device or a file.
//These macros are read and used after the device is powered on.
#define EXAMPLE_PRODUCT_KEY    "a1X2bEn****"
#define EXAMPLE_PRODUCT_SECRET  "7jluWm1zql7b****"
#define EXAMPLE_DEVICE_NAME    "example1"
#define EXAMPLE_DEVICE_SECRET   "ga7XA6KdlEeiPXQPpRbAjOZXwG8y****"
int main(int argc, char *argv[])
{
  iotx_dev_meta_info_t meta_info;
  iotx_sign_mqtt_t sign_mqtt;
  memset(&meta_info, 0, sizeof(iotx_dev_meta_info_t));
  //Use the following code to copy the previously defined device identity information to meta_info.
  memcpy(meta_info.product_key, EXAMPLE_PRODUCT_KEY, strlen(EXAMPLE_PRODUCT_KEY));
  memcpy(meta_info.product_secret, EXAMPLE_PRODUCT_SECRET, strlen(EXAMPLE_PRODUCT_SECRET));
  memcpy(meta_info.device_name, EXAMPLE_DEVICE_NAME, strlen(EXAMPLE_DEVICE_NAME));
  memcpy(meta_info.device_secret, EXAMPLE_DEVICE_SECRET, strlen(EXAMPLE_DEVICE_SECRET));
  //Call the signature function to generate the data that is required to establish an MQTT connection.
  IOT_Sign_MQTT(IOTX_CLOUD_REGION_SHANGHAI, &meta_info, &sign_mqtt);
  ...
}
```

7.1.6. IPv6-based MQTT connections

You can establish IPv6-based MQTT connections to connect devices to IoT Platform.

Context

- Only the China (Shanghai) region supports IPv6-based MQTT connections.
- During environment tests, you can use the following domain name and port to establish MQTT connections with IoT Platform.

  Domain name: `ipv6.itls.cn-shanghai.aliyuncs.com`

  Port: 1883

  Encryption protocol: TLSv1.2

  > **Note** Do not use the test domain name in production environment.

Connect devices to IoT Platform

In production environment, you must use the official MQTT domain name of a product to connect the devices of the product to IoT Platform.

1. Log on to the ticketing system, and submit a ticket to activate the AAAA record of the official MQTT domain name.

The official MQTT domain name of a product: `${YourProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com`. Replace *${YourProductKey}* with the PorductKey of your product.

2. Download the root certificate that is used for TLS encryption.
3. Develop your device to configure an MQTT connection.

   We recommend that you use the device SDKs provided by Alibaba Cloud to connect to IoT Platform. If you use custom device SDKs, you must configure a signature mechanism. For more information, see Examples of creating signatures for MQTT connections.

   The following table lists the fields to be specified.

Field	Description
Domain name and port	`${YourProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com:1883` Replace *${YourProductKey}* with the PorductKey of your product.
Variable header: keep-alive	The CONNECT command must include a keep-alive time. Valid values of the keep-alive time: 30 to 1,200 seconds. If no response is received from a device before the keep-alive time expires, IoT Platform rejects the connection request. We recommend that you set a value that is greater than 300 seconds. If a network is intermittent, set the keep-alive time to a value that is close to 1,200 seconds.

Field	Description
Parameters in an MQTT CONNECT packet	**mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=132323232|"** **mqttUsername: deviceName+"&"+productKey** **mqttPassword: sign_hmac(deviceSecret,content)** mqttPassword: the password. Calculation method: Alphabetically sort the parameters that are submitted to the server and encrypt the parameters based on the specified signature algorithm. content: a concatenated string of the parameters that are submitted to the server. These parameters include productKey, deviceName, timestamp, and clientId. The parameters are sorted in alphabetical order and concatenated without delimiters. ○ clientId: the ID of the client. We recommend that you use the MAC address or serial number (SN) of the device as the client ID. The client ID cannot exceed 64 characters in length. ○ timestamp: the current time, in milliseconds. This parameter is optional. ○ mqttClientId: Extended parameters are placed between vertical bars (|). ○ signmethod: the signature algorithm. Valid values: hmacmd5, hmacsha1, hmacsha256, and sha256. Default value: hmacmd5. ○ securemode: the current security mode. Valid values: 2 (direct TLS connection) and 3 (direct TCP connection). Example Assume that the following values are specified: **clientId=12345, deviceName=device, productKey=pk, timestamp=789, signmethod=hmacsha1, deviceSecret=secret**. The following code shows the parameters in an MQTT CONNECT message that is sent over TCP: **mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=789|** **mqttUsername=device&pk** **mqttPassword=hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();** The encrypted password is a hexadecimal string that is converted from a binary string. The following code shows the result: **FAFD82A3D602B37FB0FA8B7892F24A477F85******

For information about how to establish TCP-based MQTT connections, see Establish MQTT connections over TCP.

7.2. Use CoAP protocol

7.2.1. CoAP standard

This article describes Constrained Application Protocol (CoAP) that is supported by IoT Platform.

Protocol version

IoT Platform supports RFC 7252 Constrained Application Protocol. For more information, see RFC 7252.

Channel security

IoT Platform uses Datagram Transport Layer Security (DTLS) Version 1.2 to ensure channel security. For more information, see DTLS v1.2.

Open-source clients

For more information, see coap technology.

> **Note** Alibaba Cloud does not provide technical support for third-party code.

Limits

- The communication over CoAP feature is available only in the China (Shanghai) regions.
- Resource discovery is not supported.
- Only UDP is supported. DTLS and symmetric encryption are used to ensure data security.

Description

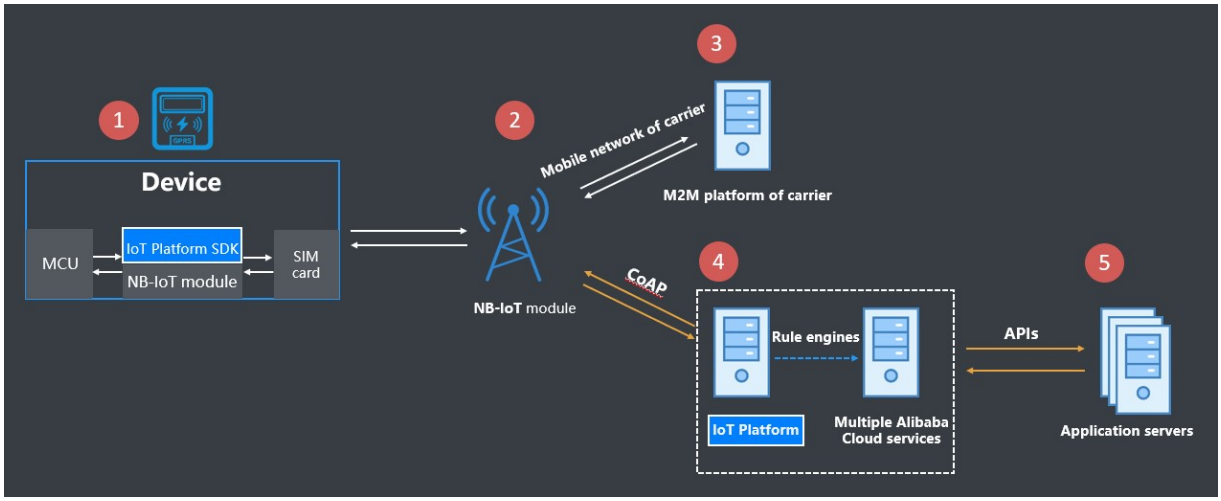
- You can use the Uniform Resource Identifier (URI) resources of CoAP in the same way as that for the URI resources of Message Queuing Telemetry Transport (MQTT). For more information, see MQTT standard.
- You can use CoAP topics in the same way as that for MQTT topics. Replace `${topic}` in the `coap://host:port/topic/${topic}` topic syntax with a real topic name. This topic name can also be used for message communication over MQTT.
- If a client passes authentication, IoT Platform returns a token. The client caches the token and uses the token to make requests.
- The size of transmitted data changes based on the specified maximum transmission unit (MTU). We recommend that set the MTU to a maximum of 1 KB.
- If IoT Platform identifies that a device reports data once or more over CoAP in the last 10 minutes, the device is in the Online state. The status appears in the IoT Platform console.

7.2.2. Connect devices to IoT Platform over CoAP

You can connect devices to IoT Platform over the Constrained Application Protocol (CoAP). CoAP is used for low-power and resource-constrained devices, such as NB-IoT devices. This article describes how to connect a device to IoT Platform over CoAP. It also describes how to authenticate the device by using Datagram Transport Layer Security (DTLS) or symmetric encryption.

Procedure

The following figure shows how to connect an NB-IoT device to IoT Platform.



Procedure:

1. Integrate an IoT Platform SDK into the NB-IoT module of a device. A device provider applies for the device certificate in the IoT Platform console and burns the device certificate to the device.
2. Connect the NB-IoT device to IoT Platform over the mobile network of a carrier. You must contact your local carrier to make sure that the NB-IoT network is available in the region where your device resides.
3. Use the machine-to-machine (M2M) platform of the carrier to manage data traffic and fees. The M2M platform capabilities are provided by the carrier.
4. Collect data in real time and submit the data to IoT Platform over CoAP or UDP. IoT Platform allows you to establish secure connections with hundreds of millions of devices and manage a large amount of device data. IoT Platform also allows you to transmit data to multiple Alibaba Cloud services for further processing. These services include big data services, database services, and Tablestore.
5. Use the data access-related API operations and message pushing services that are provided by IoT Platform to forward data to business servers and integrate devices and applications.

Connect devices by using symmetric encryption

1. Connect to the CoAP server.Endpoint:
   -
   - The endpoint of the public instance in the China (Shanghai) region is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`.
     -
     - Replace the `${port}` variable with your port number. Port 5682 is used for symmetric encryption.
2. Authenticate the device.

   Sample request:

```
POST /auth
Host: ${YourEndpoint}
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"a1NUjcV****","deviceName":"ff1a11e7c08d4b3db2b1500d8e0e55","clientId":"
a1NUjcV****&ff1a11e7c08d4b3db2b1500d8e0e55","sign":"F9FD53EE0CD010FCA40D14A9FE******", "seq
":"10"}
```

Parameters of device authentication

Parameter	Description
Method	The request method. Valid value: POST.
URL	The URL. Valid value: /auth.
Host	The endpoint.
Port	The port number. Valid value: 5682.
Accept	The MIME type of the data that is received by the device. Valid values: *application/json* and *application/cbor*.
Content-Format	The MIME type of the data that the device submits to IoT Platform. Valid values: *application/json* and *application/cbor*.
payload	The JSON-formatted device information for authentication. For more information, see the following table.

Parameters of device information

Field	Required	Description
productKey	Yes	The ProductKey in the device certificate. The ProductKey is a globally unique identifier (GUID) that is issued by IoT Platform to the product. You can log on to the IoT Platform consoleand view the ProductKey on the **Device Details** page.
deviceName	Yes	The DeviceName in the device certificate. The DeviceName is the system-defined or custom device name when you register the device. You can log on to the IoT Platform consoleand view the DeviceName on the **Device Details** page.
ackMode	No	The communication mode. Valid values: ◦ *0*: IoT Platform returns response data and an ACK message at the same time. ◦ *1*: IoT Platform returns an ACK message and then returns response data. Default value: 0.

Field	Required	Description
sign	Yes	The signature. You can use the `signmethod(DeviceSecret,content)` function to calculate a signature. Then, you can specify the signature for the sign parameter. The hmacmd5 and hmacsha1 signature methods are supported. Required parameters: ◦ signmethod: the signature algorithm. The value must be the same as the value of the specified signmethod parameter. ◦ DeviceSecret: the DeviceSecret of the device. You can log on to the IoT Platform consoleand view the DeviceSecret on the **Device Details** page. ◦ content: all parameters that are submitted to IoT Platform, except for the version, sign, resources, and signmethod parameters. The values are spliced in sequence based on the alphabetical order of these parameters. No splicing symbol is used to separate these values. **Note** The parameter values that are used to calculate the signature must be the same as the parameter values that you specify in the request of device authentication. Example: `hmac_md5(mRPVdzSMu2nVBxzK77ERPIMxSYIv****, clientIda1NUjcV****&ff1a11e7c08d4b3db2b1500d8e0e55deviceNameff1a11e7c08d4b3db2b1500d8e0e55productKeya1NUjcV****seq10timestamp1524448722000)`
signmethod	No	The signature algorithm. Valid values: hmacmd5 and hmacsha1. Default value: hmacmd5.
clientId	Yes	The client ID. The client ID must be 1 to 64 characters in length. We recommend that you use the MAC address or SN of the device as the value of the clientId parameter.
timestamp	No	The timestamp. IoT Platform does not verify the timestamp.

Sample response:

```
{"random":"ad2b3a5eb51d6****","seqOffset":1,"token":"MZ8m37hp01w1SSqoDFzo001050****.ad2b"}
```

Response parameters

Field	Description
random	The key that is used to encrypt upstream and downstream data.
seqOffset	The initial offset of the seq parameter.
token	The token returned if the device is authenticated.

3. Submit data.

   Sample request:

```
POST /topic/${topic}
Host: ${YourEndpoint}
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(token), 2089(seq)
```

Parameters

Field	Required	Description
Method	Yes	The request method. Valid value: POST.
URL	Yes	The URL of the topic. Format: `/topic/${topic}`. Replace the *${topic}* variable with the topic to which the data is sent.
Host	Yes	The endpoint.
Port	Yes	The port number. Valid value: 5682.
Accept	Yes	The MIME type of the data that is received. Valid values: *application/json* and *application/cbor*.
Content-Format	Yes	The MIME type of the upstream data. IoT Platform does not verify the data. Valid values: *application/json* and *application/cbor*.

Field	Required	Description

payload	Yes	The AES-encrypted upstream data.
		
		Note If you use AES to encrypt data, set the Transform parameter to `AES/CBC/PKCS5Padding` and the IV parameter to `543yhjy97ae7fyfg`. A key is generated by using the SHA-256 algorithm.
		
		Example:
		
		If the request is `deviceSecret=zPwChiLh0EaifR809D5Rc6LDIC6A****`, the response is `random=8fe3c8d50e10****`.
		
		i. Combine the values of the deviceSecret and random parameters to form a string in the `${deviceSecret},${random}` format.
		
		`zPwChiLh0EaifR809D5Rc6LDIC6A****,8fe3c8d50e10****`
		
		ii. IoT Platform encodes the preceding string by using the UTF-8 format, encrypts the encoded string by using the SHA-256 algorithm, and then converts the string into a hexadecimal string.
		
		`59ea5ac1cb092e5910c405821119959e5297516d185b71e344735cf3f268****`
		
		iii. IoT Platform uses the subString(16,48) function to extract a sub-string of 32 characters from the preceding string to form a key. The extraction starts from the 17th character of the string.
		
		`10c405821119959e5297516d185b71e3`

Field	Required	Description
CustomOptions	Yes	The custom option. Valid values: ○ 2088: the token parameter. Use the value of the token parameter that is returned after the device is authenticated. Note Each time the device submits data, the token parameter is required. If the token expires, you must authenticate the device again and obtain another token. ○ 2089: the seq parameter. The value must be greater than the value of the seqOffset parameter. The value must be a random digit that is unique during the validity period of authentication. We recommend that you use a value that is incremented based on the seq parameter in each request packet and perform AES encryption on the value. Sample response: number:2090 (IoT Platform message ID) You can specify the token and seq parameters in the CustomOptions or URL parameter, for example, /topic/${topic}? token=xxxx&seq=xxxxx. If you specify the token and seq parameters for the CustomOptions and URL parameters at the same time, the CustomOptions parameter is used.

After a message is sent to IoT Platform, a status code that indicates a successful request and a message ID that is generated by IoT Platform are returned.

Connect devices to IoT Platform over DTLS

1. Connect to the CoAP server. Endpoint:
   -
   - The endpoint of the public instance in the China (Shanghai) region is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`.
     -
     - Replace the *${port}* variable with your port number. Default port number for DTLS is 5684.
2. If you use an official device SDK, DTLS uses the PSK algorithm to secure channels by default.If you use a third-party device SDK, you must download the root certificate for DTLS secure channels. Then, you can use the DTLS libraries to connect the device to IoT Platform.

```
psk_id: "${authType}" + "|" + "${signMethod}" + "|" + "${productKey}" + "&" + "${deviceName}" + "timestamp"
psk: signMethod(DeviceSecret, "${productKey}" + "&" + "${deviceName}" + "${timestamp}")
```

Fields

Field	Required	Description
authType	Yes	The authentication type. Valid value: devicename.
signMethod	Yes	The signature algorithm. Valid values: hmacmd5, hmacsha1, and hmacsha256.
productKey	Yes	The ProductKey of the product to which the device belongs.
deviceName	Yes	The DeviceName of the device.
DeviceSecret	Yes	The DeviceSecret of the device.
timestamp	Yes	The timestamp.

3. Authenticate the device. You can use the auth operation to authenticate the device and obtain a token. Each time the device submits data to IoT Platform, the token parameter is required.

   Sample request:

   ```
   POST /auth
   Host: ${YourEndpoint}
   Port: 5684
   Accept: application/json or application/cbor
   Content-Format: application/json or application/cbor
   payload: {"productKey":"ZG1EvTE****","deviceName":"NlwaSPXsCpTQuh8FxBGH","clientId":"mylight1000002","sign":"bccb3d2618afe74b3eab12b94042****"}
   ```

   For more information about the required parameters and payload parameters except for the Port parameter, see Connect devices by using symmetric encryption.

   Sample response:

   ```
   response: {"token":"f13102810756432e85dfd351eeb4****"}
   ```

Response codes

Code	Description	Payload	Remarks
2.05	Content	Token if the device passes the authentication.	The request is valid.
4.00	Bad Request	no payload	The payload in the request is invalid.
4.01	Unauthorized	no payload	The request is unauthorized.
4.03	Forbidden	no payload	The request is forbidden.

Code	Description	Payload	Remarks
4.04	Not Found	no payload	The requested URL does not exist.
4.05	Method Not Allowed	no payload	The request method is not allowed.
4.06	Not Acceptable	no payload	The Accept parameter is invalid.
4.15	Unsupported Content-Format	no payload	The requested content is invalid.
5.00	Internal Server Error	no payload	The request failed because a timeout issue or an error occurs on the authentication server.

4. Submit data.

   The device sends data to a topic. Only custom topics with the Publish permission are supported.

   For example, the topic format is `/${YourProductKey}/${YourDeviceName}/pub`. If the DeviceName is device and the ProductKey is a1GFiLP ****, you can use the `a1GFjLP****.coap.cn-shanghai.link.aliyuncs.com:5684/topic/a1GFjLP****/device/pub` topic to submit data.

   Sample request:

```
POST /topic/${topic}
Host: ${YourEndpoint}
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(token)
```

Parameters

Parameter	Required	Description
Method	Yes	The request method. Valid value: POST.
URL	Yes	`/topic/${topic}`. Replace the `${topic}` variable with the topic to which the data is sent.
Host	Yes	The endpoint.
Port	Yes	The port number. Valid value: 5684.
Accept	Yes	The MIME type of the data that is received. Valid values: *application/json* and *application/cbor*.
Content-Format	Yes	The MIME type of the upstream data. IoT Platform does not verify the data. Valid values: *application/json* and *application/cbor*.

Parameter	Required	Description
CustomOptions	Yes	◦ number: 2088. ◦ token: the token that is returned from the authentication service. Note Each time the device submits data, the token parameter is required. If the token expires, you must re-authenticate the device and obtain another token.

7.3. Use HTTP protocol

7.3.1. HTTP standard

IoT Platform supports HTTPS. This article describes the HTTP standard that is supported by IoT Platform.

HTTP versions

- IoT Platform supports HTTP/1.0. For more information, see RFC 1945
- IoT Platform supports HTTP/1.1. For more information, see RFC 2616

Channel security

IoT Platform uses HTTPS to ensure channel security.

Note
- IoT Platform supports TLS 1.0, TLS 1.1, and TLS 1.2. IoT Platform will soon not support TLS 1.0 due to security risks. We recommend that you use TLS 1.2 to authenticate your device.
- TLS 1.2 is integrated with Link SDKs by default. You do not need to configure the TLS feature.

Limits

- Only HTTPS is supported.
- Request parameters cannot be specified after a question mark (?).
- The resource discovery feature is not supported.

Description

- The HTTP standard is consistent with the MQTT standard. For more information, see MQTT standard.
- If IoT Platform identifies that a device reports data once or more over CoAP in the last 10 minutes, the device is in the Online state. The status appears in the IoT Platform.

7.3.2. Establish connections over HTTP

You can connect a device to IoT Platform over HTTP. Only HTTPS is supported. This article describes how to connect a device to IoT Platform over HTTP.

Limits

- You can establish connections over HTTP only in the China (Shanghai) region.
- Only HTTPS is supported.
- Connections over HTTP are used for devices to submit data. A device can submit a maximum of 128 KB of data at a time.
- The standards of HTTP topics and MQTT topics are the same. If you connect a device to IoT Platform over HTTP, you can use MQTT topics for message communication between the device and IoT Platform. When you connect a device to IoT Platform over HTTP, the device submits data to a topic in the `${endpoint}/topic/${topic}` format. You cannot specify a parameter by using the `?query_String=xxx` format.
- Only the POST request method is supported.
- The token that is returned during device authentication expires after a specified period. The current validity period is seven days. Make sure that you specify the logic to process expired tokens.

Procedure

You must authenticate a device to obtain the device token, and then use the token to submit data.

1. Authenticate the device to obtain the device token. Endpoint:
   -
   - The endpoint of the public instance that resides in the China (Shanghai) region is `https://iot-as-http.cn-shanghai.aliyuncs.com`.

   Authentication request:

```
POST /auth HTTP/1.1
Host: ${YourEndpoint}
Content-Type: application/json
body: {"version":"default","clientId":"mylight1000002","signmethod":"hmacsha1","sign":"4870141D4067227128CBB4377906C3731CAC221C","productKey":"ZG1EvTE****","deviceName":"NlwaSPXsCpTQuh8FxBGH","timestamp":"1501668289957"}
```

Parameters

Parameter	Description
Method	The request method. Valid value: POST.
URL	The URL. Only HTTPS is supported. Valid value: /auth.
Host	The endpoint.
Content-Type	The MIME type of upstream data that the device submits to IoT Platform. Valid value: application/json. If another MIME type is specified, an error occurs.
body	The device information for authentication, in JSON format. For more information, see body parameters.

body parameters

Parameter	Required	Description
productKey	Yes	The ProductKey of the product to which the device belongs. You can log on to the IoT Platform consoleand view the ProductKey on the **Device Details** page.
deviceName	Yes	The DeviceName of the device. You can log on to the IoT Platform consoleand view the DeviceName on the **Device Details** page.
clientId	Yes	The ID of the client. The client ID must be 1 to 64 characters in length. We recommend that you use the MAC address or SN of the device as the value of the clientId parameter.
timestamp	No	The timestamp. A request is valid within 15 minutes after the timestamp is created. The timestamp is in the numeric format. This value is a UNIX timestamp representing the number of milliseconds that have elapsed since the epoch time January 1, 1970, 00:00:00 UTC.

Parameter	Required	Description
sign	Yes	The signature value. The signature is calculated by using the `hmacmd5(deviceSecret,content)` function. The value of content is a string that contains all the parameters to be submitted to IoT Platform, except for the version, sign, and signmethod parameters. These parameters are sorted in alphabetical order and spliced without a splicing symbol. Example: If the clientId parameter is set to 127.0.0.1, the deviceName parameter is set to http_test, the productKey parameter is set to a1FHTWxQ****, the timestamp parameter is set to 1567003778853, the signmethod parameter is set to hmacmd5, and the deviceSecret parameter is set to 89VTJylyMRFuy2T3sywQGbm5Hmk1****, use the following function to calculate the signature: `hmacmd5("89VTJylyMRFuy2T3sywQGbm5Hmk1****","clientId127.0.0.1deviceNamehttp_testproductKeya1FHTWxQ****timestamp1567003778853").toHexString();` The `toHexString()` method is used to convert a set of four-bit binary digits that form the binary result into a hexadecimal string. The hexadecimal string is case-insensitive. For example, an array that includes the result in the decimal format is [60 68 -67 -7 -17 99 30 69 117 -54 -58 -58 103 -23 113 71]. After the array is converted into a hexadecimal string, it is 3C44BDF9EF631E4575CAC6C667E97147.
signmethod	No	The signature algorithm. Valid values: hmacmd5 and hmacsha1. Default value: hmacmd5.
version	No	The version number. Default value: default.

Sample response:

```
body:
{
 "code": 0,
 "message": "success",
 "info": {
  "token": "6944e5bfb92e4d4ea3918d1eda39****"
 }
}
```

> Note
>
> - Cache the returned token on the device.
> - The token is required each time the device submits data to IoT Platform. If the token expires, you must re-authenticate the device to obtain another token.

Error codes

code	message	Remarks
10000	common error	The error message returned because an unknown error occurred.
10001	param error	The error message returned because one or more parameters are invalid.
20000	auth check error	The error message returned because the device failed to be authenticated.
20004	update session error	The error message returned because the device failed to be updated.
40000	request too many	The error message returned because IoT Platform cannot process this number of requests. The throttling policy is applied.

2. Submit data.

   The device sends data to a topic. Only custom topics with the Publish permission are supported.

   For example, the topic is `/${YourProductKey}/${YourDeviceName}/pub`. Assume that the DeviceName is device123 and the ProductKey is a1GFiLP ****. You can use the `https://iot-as-http.cn-shanghai.aliyuncs.com/topic/a1GFjLP****/device123/pub` URL to submit data.

   Sample request:

```
POST /topic/${topic} HTTP/1.1
Host: ${YourEndpoint}
password:${token}
Content-Type: application/octet-stream
body: ${your_data}
```

Parameters

Parameter	Description
Method	The request method. Valid value: POST.
URL	`/topic/${topic}`. Replace the `${topic}` variable with the topic to which data is sent. Only HTTPS is supported.
Host	The endpoint.
password	This parameter is included in the request header. Set this parameter to the token that is returned after you call the auth operation to authenticate the device.
Content-Type	The MIME type of upstream data that the device submits to IoT Platform. Valid value: application/octet-stream. If another MIME type is specified, an error occurs.
body	The data that is sent to the specified topic.

Sample response:

```
body:
{
 "code": 0,
 "message": "success",
 "info": {
  "messageId": 892687627916247040,
 }
}
```

Error codes

code	message	Remarks
10000	common error	The error message returned because an unknown error occurred.
10001	param error	The error message returned because one or more parameters are invalid.
20001	token is expired	The error message returned because the token has expired. You must call the auth operation to re-authenticate the device and obtain another token.
20002	token is null	The error message returned because no token is specified in the request header.
20003	check token error	The error message returned because IoT Platform failed to obtain the device identity information based on the token. You must call the auth operation to re-authenticate the device and obtain another token.

code	message	Remarks
30001	publish message error	The error message returned because the device failed to submit data.
40000	request too many	The error message returned because IoT Platform cannot process this number of requests. The throttling policy is applied.

8.Generic protocol SDK

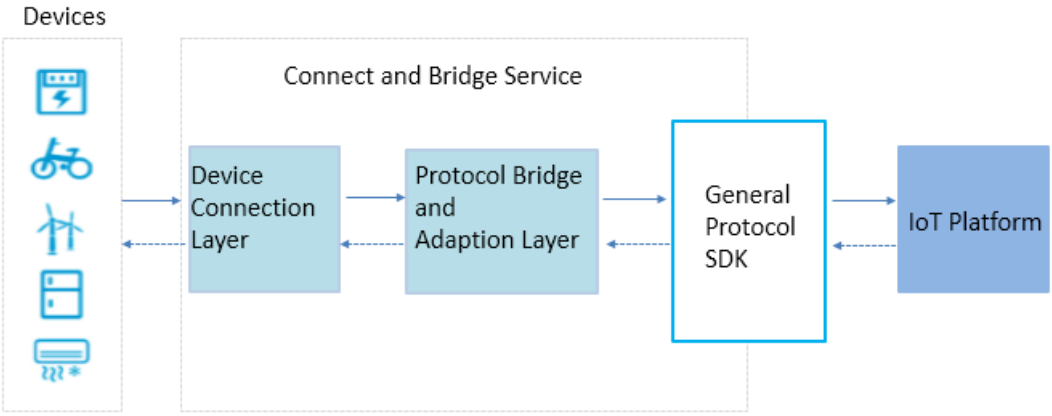
8.1. What is the IoT as Bridge SDK?

Alibaba Cloud IoT Platform supports communication over MQTT, CoAP, or HTTP. Other types of protocols, such as the fire protection agreement GB/T 26875.3-2011, Modbus, and JT808, are not supported. In some scenarios where devices cannot be directly connected to IoT Platform, you can use the IoT as Bridge SDK to deploy a bridging service and establish connections between the devices and IoT Platform.

> **Note** The IoT as Bridge SDK is supported only in the following regions: China (Shanghai), Germany (Frankfurt), and US (Virginia).

Architecture

The IoT as Bridge SDK is a self-adaptive protocol framework. This SDK is used to deploy a bridging service and achieve communication between IoT Platform and your devices.



Scenarios

- Your device cannot be directly connected to IoT Platform due to network or hardware limits.
- Your device uses a protocol that is unsupported by IoT Platform.
- A connection is already established between your device and a bridge server. You want to connect the device to IoT Platform without modifying the device and protocol.
- Your device is connected to a server and needs to be updated.
  - The device needs to use IoT Platform capabilities, such as the OTA update feature.
  - The device needs to be integrated into an IoT Platform-based solution.
  - The device needs additional processing logic to meet business requirements.

Features

The IoT as Bridge SDK enables a bridge server to communicate with IoT Platform.

Basic features:

- Allows you to manage configurations by using a configuration file.

- Allows you to manage device connections.
- Provides upstream communication capabilities.
- Provides downstream communication capabilities.

Advanced features:

- Allows you to manage configurations by using API operations.
- Provides API operations to submit one or more properties and events, update tags, set properties, and call services.

Terms

Term	Description
device	The device in a real IoT scenario that cannot directly communicate with IoT Platform by using a supported protocol.
bridge server	The server to which the device is connected. This server uses a specific protocol to communicate with the device and uses the IoT as Bridge SDK to communicate with IoT Platform.
original protocol	The specific protocol that is used between the device and the bridge server. The IoT as Bridge SDK does not involve the definition and implementation of the original protocol.
original device identifier	The unique identifier that is used by the device to communicate with the bridge server over the original protocol. The IoT as Bridge SDK provides the originalIdentity parameter to specify the identifier of the device.
device certificate	The device certificate that is obtained after you register the device in IoT Platform. The certificate information includes ProductKey, DeviceName, and DeviceSecret. When you use the IoT as Bridge SDK, you do not need to burn the device certificate on the device. Instead, you must configure the *devices.conf* file. The bridge maps the originalIdentity parameter of the device to the device certificate.
bridge certificate	The device certificate that is returned after you register the bridge device in IoT Platform. The certificate information includes ProductKey, DeviceName, and DeviceSecret. The bridge certificate uniquely identifies the bridge in IoT Platform.

Develop and deploy a bridging service

1. Log on to the IoT Platform console, create a product and device, and then obtain the certificate of the bridge device.

   For more information, see Create a product, Create a device, and Create multiple devices at a time.

   You must specify the certificate of the bridge device when you configure the IoT as Bridge SDK.

   > **Note** The bridge is a virtual device. You can use any device certificate as the certificate of the bridge.

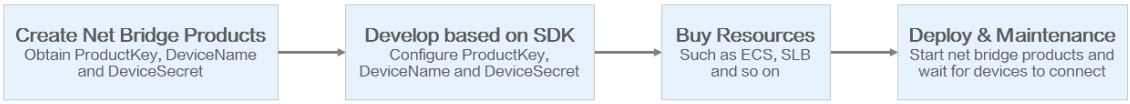
2. Configure the IoT as Bridge SDK.

   For more information, see Use the basic features and Use the advanced features.

> **Note** The IoT as Bridge SDK supports only the Java programming language. Only JDK 1.8 and later versions are supported.

3. Deploy the developed bridging service.
   - To ensure high scalability, you can use Alibaba Cloud services such as Elastic Compute Service (ECS) and Server Load Balancer (SLB) to deploy the bridging service in Alibaba Cloud.
   - You can also deploy the service on premises to ensure trusted communication.

   The following figure shows the procedure of using ECS to deploy the bridging service.



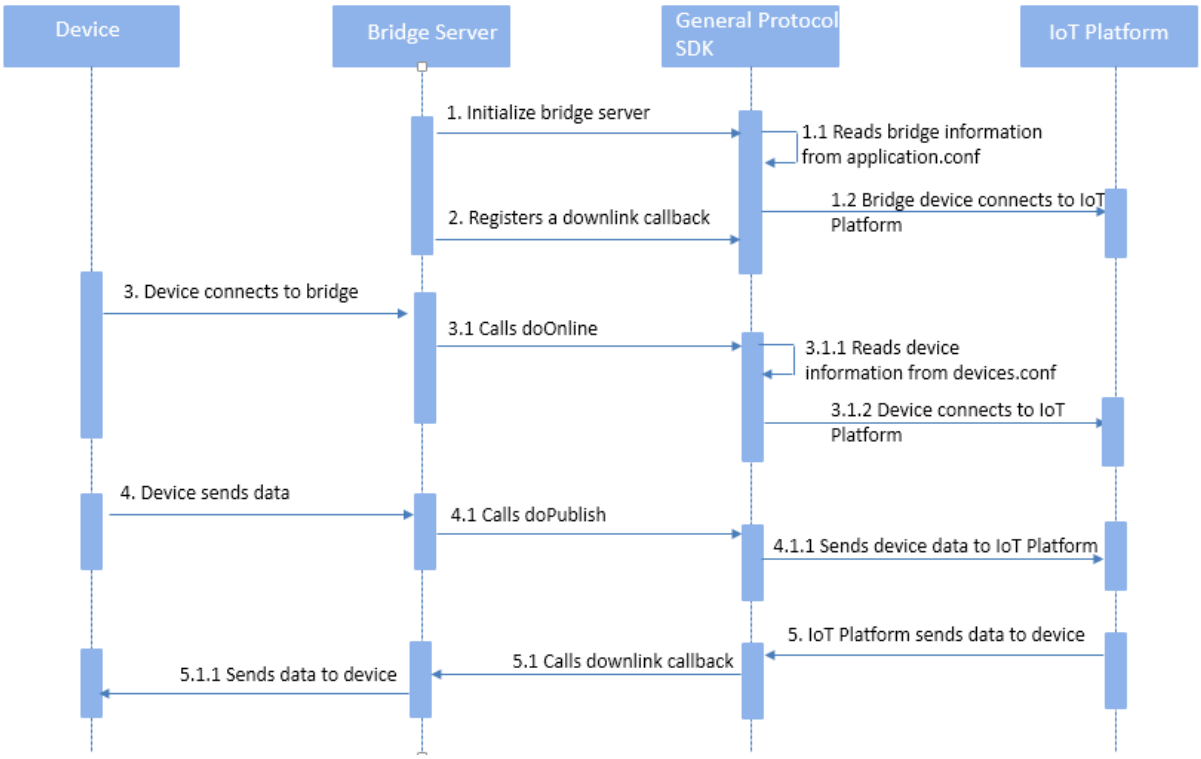
8.2. Use the basic features

Your device can connect to and communicate with Alibaba Cloud IoT Platform by using the bridging service that is supported by the IoT as Bridge SDK. This article describes how to configure the IoT as Bridge SDK to use the basic features. These features include device connection and disconnection, and upstream and downstream message transmission.

For more information, see IoT as Bridge SDK.

Process

The following figure shows the process of connecting a device to IoT Platform by using the IoT as Bridge SDK.



Deploy a development environment

Deploy a development environment to use the SDK for Java and add the following Maven dependency to your project to import the IoT as Bridge SDK.

```
<dependency>
 <groupId>com.aliyun.openservices</groupId>
 <artifactId>iot-as-bridge-sdk-core</artifactId>
 <version>2.3.5</version>
</dependency>
```

Initialize the SDK

- Initialize the SDK.

  Create a BridgeBootstrap object and call the bootstrap() method. When you call this method, the IoT as Bridge SDK registers the DownlinkChannelHandler callback to receive downstream messages from IoT Platform.

  After the IoT as Bridge SDK is initialized, the SDK reads the bridge information and sends a connection request to IoT Platform for the bridge.

  Sample code:

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

- Specify the bridge information.

  By default, a bridge is configured by using a configuration file. The *application.conf* configuration file is in the *src/main/resources/* directory of the Java project. The file format is HOCON (a JSON superset).

  The IoT as Bridge SDK uses the typesafe.config file to parse the configuration file.

  You must dynamically register a bridge device and specify the parameters of the bridge device.

  For more information about how to dynamically register bridge devices, see Dynamically register bridge devices.

  The following table describes the parameters of the bridge device.

Parameter	Required	Description
productKey	Yes	The ProductKey of the product to which the bridge device belongs.
deviceName	Yes	The DeviceName of the bridge device.

Parameter	Required	Description
deviceSecret	Yes	The DeviceSecret of the bridge device.
subDeviceConnectMode	No	The mode that devices use to connect with the bridge. ◦ If this parameter is set to 3, a large-size bridge is created. A maximum of 500,000 devices can connect with the bridge. ◦ If this parameter is not specified, a small-size bridge is created. A maximum of 15,00 devices can connect with the bridge. Large-size bridges and small-size bridges use different policies to disconnect devices. For more information, see Disconnect a device from IoT Platform.
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway. The bridge and IoT Platform establish a persistent connection over the HTTP/2 protocol. Endpoint format: ◦ Public instance: `https://${productKey}.iot-as-http2. ${RegionId}.aliyuncs.com:443`. Replace *${productKey}* with the ProductKey of the product to which your bridge device belongs. Replace *${RegionId}* with the ID of the region where you purchased the IoT Platform service. For information about region IDs, see Regions and zones. For example, if the ProductKey of a bridge device is a1abcab**** and the IoT Platform service is purchased in the China (Shanghai) region, the endpoint is `https://a1abcab****.iot-as-http2.cn-shanghai.aliyuncs.com:443`. ◦ Enterprise Edition instance: `https://${IotInstanceId}.http2.iothub.aliyuncs.com:443`. Replace *${IotInstanceId}* with the ID of the purchased instance. For example, if the instance ID is iot-cn-g06kwb****, the endpoint is `https://iot-cn-g06kwb****.http2.iothub.aliyuncs.com:443`.

Parameter	Required	Description
authEndpoint	Yes	The endpoint of the device authentication server. Endpoint format: ◦ Public instance: `https://iot-auth. .${RegionId}.aliyuncs.com/auth/bridge`. Replace *${RegionId}* with the ID of the region where you purchased the IoT Platform service. For information about region IDs, see Regions and zones. For example, if the IoT Platform service is purchased in the China (Shanghai) region, the endpoint is `https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`. ◦ Enterprise Edition instance: `https://${IotInstanceId}.auth.iothub.aliyuncs.com/auth/bridge`. Replace *${IotInstanceId}* with the ID of the purchased instance. For example, if the instance ID is iot-cn-g06kwb****, the endpoint is `https://iot-cn-g06kwb****.auth.iothub.aliyuncs.com/auth/bridge`.

The following example shows how to configure a small-size bridge device. The public instance is used in the example.

```
# The endpoint of the server.
http2Endpoint = "https://a1tN7***.iot-as-http2.cn-shanghai.aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge"
# The information of the bridge device.
productKey = ${bridge-ProductKey-in-Iot-Plaform}
deviceName = ${bridge-DeviceName-in-Iot-Plaform}
deviceSecret = ${bridge-DeviceSecret-in-Iot-Plaform}
```

Authenticate a device and connect the device to IoT Platform

- Connect a device to the bridge.

  The following example shows how to connect a device to the bridge by using the IoT as Bridge SDK:

```
/**
 * Authenticate the device.
 * @param newSession: the device session information that is returned in a downstream callback.
 * @param originalIdentity: the original identifier of the device.
 * @return
 */
public boolean doOnline(Session newSession, String originalIdentity);
```

  When the device is connected to the bridge, the device must pass the Session parameter. The Session parameter is returned to the bridge by using a callback function of a downstream message. The bridge determines the device that sends the message by using the originalIdentity field in the Session parameter.

The Session parameter includes the optional channel field that carries the information about the device connection. For example, your bridge server is built based on Netty. You can use the channel field to carry the channel object that corresponds to the persistent connection of the device. When a downstream message is sent, the bridge can obtain the channel object from the Session parameter.

The IoT as Bridge SDK does not process the data of the channel field. You can also use the channel field to carry the device-related information.

Sample code:

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

- You must configure the mappings between device certificates and original device identifiers.

  By default, you can configure the settings by using a configuration file. The *application.conf* configuration file is in the *src/main/resources/* directory of the Java project. The file format is HOCON (a JSON superset).

  The IoT as Bridge SDK uses the typesafe.config file to parse the configuration file.

  You must specify the following parameters in the file:

```
${device-originalIdentity} {
 productKey : ${device-ProductKey-in-Iot-Plaform}
 deviceName : ${device-DeviceName-in-Iot-Platform}
 deviceSecret : ${device-DeviceSceret-in-Iot-Platform}
}
```

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

Send data from a device to IoT Platform

The following example shows how to send data from a device to IoT Platform by using the IoT as Bridge SDK:

```
/**
 * Send a message from the device by using a synchronous call.
 * @param originalIdentity: the original identifier of the device.
 * @param protocolMsg: the message to be sent, including the topic, payload, and quality of service (QoS) in
formation.
 * @param timeout: the timeout period. Unit: seconds.
 * @return: indicates whether the message is sent within the timeout period.
 */
boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg, int timeout);
/**
 * Send a message from the device by using an asynchronous call.
 * @param originalIdentity: the original identifier of the device.
 * @param protocolMsg: the message to be sent, including the topic, payload, and QoS information.
 * @return: After this method is called, the CompletableFuture parameter is immediately returned and avail
able for subsequent use.
 */
CompletableFuture<ProtocolMessage> doPublishAsync(String originalIdentity,
                                  ProtocolMessage protocolMsg);
```

Sample code:

```
DeviceIdentity deviceIdentity =
  ConfigFactory.getDeviceConfigManager().getDeviceIdentity(originalIdentity);
ProtocolMessage protocolMessage = new ProtocolMessage();
protocolMessage.setPayload("Hello world".getBytes());
protocolMessage.setQos(0);
protocolMessage.setTopic(String.format("/%s/%s/update",
    deviceIdentity.getProductKey(), deviceIdentity.getDeviceName()));
// Synchronous sending.
int timeoutSeconds = 3;
boolean success = upLinkHandler.doPublish(originalIdentity, protocolMessage, timeoutSeconds);
// Asynchronous sending.
upLinkHandler.doPublishAsync(originalIdentity, protocolMessage);
```

Push data from IoT Platform to a device

When the bridge calls the bootstrap() method, the IoT as Bridge SDK registers the DownlinkChannelHandler callback. When the IoT as Bridge SDK receives a message from IoT Platform, it calls the pushToDevice() method in DownlinkChannelHandler.

You can modify the pushToDevice() method to enable the bridge to process the downstream message.

> **Note**
>
> - After a device is connected to IoT Platform by using the IoT as Bridge SDK, the device can receive downstream messages without subscribing to topics.
> - Do not implement a time-consuming logic in the pushToDevice() method. Otherwise, the thread that receives downstream messages are blocked. If a time-consuming logic or I/O logic is required, implement the logic in an asynchronous manner. For example, if a bridge uses a persistent connection to forward downstream messages to a device, you can implement the logic in an asynchronous manner.

Sample code:

```
private static ExecutorService executorService  = new ThreadPoolExecutor(
  Runtime.getRuntime().availableProcessors(),
  Runtime.getRuntime().availableProcessors() * 2,
  60, TimeUnit.SECONDS,
  new LinkedBlockingQueue<>(1000),
  new ThreadFactoryBuilder().setDaemon(true).setNameFormat("bridge-downlink-handle-%d").build(),
  new ThreadPoolExecutor.AbortPolicy());
public static void main(String args[]) {
  // By default, application.conf and devices.conf are used.
  bridgeBootstrap = new BridgeBootstrap();
  bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
    @Override
    public boolean pushToDevice(Session session, String topic, byte[] payload) {
      // Receive messages from IoT Platform.
      executorService.submit(() -> handleDownLinkMessage(session, topic, payload));
      return true;
    }
    @Override
    public boolean broadcast(String s, byte[] bytes) {
      return false;
    }
  });
}
private static void handleDownLinkMessage(Session session, String topic, byte[] payload) {
  String content = new String(payload);
  log.info("Get DownLink message, session:{}, topic:{}, content:{}", session, topic, content);
  Object channel = session.getChannel();
  String originalIdentity = session.getOriginalIdentity();
}
```

Parameter	Description
session	When you call the doOnline operation, pass the session parameter. This parameter is used to determine the device to which the downstream message is sent.
topic	The topic of the downstream message.
payload	The message body of the downstream message. The message body is in the binary format.

Disconnect a device from IoT Platform

The following points describe the scenarios of device disconnection:

- If a small-size bridge is disconnected from IoT Platform, all devices that are connected to the bridge are automatically disconnected from IoT Platform.
- If a large-size bridge is disconnected from IoT Platform, the devices that are connected to the bridge are not disconnected from IoT Platform. After the bridge is reconnected to IoT Platform, you can update the status of a sub-device by calling the doOffline operation.

The status of a sub-device indicates whether the sub-device is connected to a gateway. The gateway reports the status of sub-devices to IoT Platform. If the gateway cannot report the status of the sub-device to IoT Platform, the status that is displayed in the IoT Platform console remains unchanged.

Assume that a sub-device is connected to IoT Platform by using a gateway and the status of the sub-device is online. If the gateway is disconnected from IoT Platform, the gateway cannot report the status of the sub-device to IoT Platform. Therefore, the status of the sub-device remains online.

- If small-size and large-size bridges are connected to IoT Platform, they can send a request to disconnect a device from IoT Platform.

  The following example shows how to send a disconnection request:

```
/**
 * Send a request to disconnect a device from IoT Platform.
 * @param originalIdentity: the original identifier of the device.
 * @return: indicates whether the disconnection request is sent.
 */
boolean doOffline(String originalIdentity);
```

  Sample code:

```
upLinkHandler.doOffline(originalIdentity);
```

A bridge ends and re-establishes a connection with IoT Platform

A bridge can use the BridgeBootstrap object and call the disconnectBridge and reconnectBridge operations to end and re-establish a connection with IoT Platform.

> **Note** The reconnectBridge operation is used for reconnection and cannot be used for first connection.

Sample code:

```
// End the connection with IoT Platform.
bridgeBootstrap.disconnectBridge();
Thread.sleep(1000);
// Check whether the bridge is connected to IoT Platform.
boolean isConnected = bridgeBootstrap.isBridgeConnected();
// Re-establish a connection with IoT Platform.
bridgeBootstrap.reconnectBridge();
Thread.sleep(1000);
isConnected = bridgeBootstrap.isBridgeConnected();
```

8.3. Use the advanced features

This article describes how to use the advanced features of the IoT as Bridge SDK. The advanced features allow you to specify the configuration file paths, dynamically register bridge devices, and call the specified operations that are encapsulated in the IoT as Bridge SDK to submit properties, events, and tags.

Specify configuration file paths

By default, the configuration file of a bridge device is *application.conf*, and the configuration file of the device certificate mapping is *devices.conf*.

The IoT as Bridge SDK allows you to specify paths. Before you call the bootstrap() method, you must call the *ConfigFactory.init()* method to specify the path of a configuration file. You can also create an instance and configure the API operations as needed.

Sample code:

```
ConfigFactory.init(
  ConfigFactory.getBridgeConfigManager("application-self-define.conf"),
  selfDefineDeviceConfigManager);
bridgeBootstrap.bootstrap();
private static DeviceConfigManager selfDefineDeviceConfigManager = new DeviceConfigManager() {
  @Override
  public DeviceIdentity getDeviceIdentity(String originalIdentity) {
    return devicesMap.get(originalIdentity);
  }
  @Override
  public String getOriginalIdentity(String productKey, String deviceName) {
    return null;
  }
};
```

Dynamically register bridge devices

When you deploy bridges on a large number of servers, the deployment process is complex if you specify different bridge devices for different bridge servers. You can edit the bridge configuration file *application.conf* to dynamically register the bridge devices.

You must specify the productKey and popClientProfile parameters of each bridge device in the configuration file. Then, the IoT as Bridge SDK calls an IoT Platform operation to register the bridge devices and uses the MAC addresses of the servers as the device names.

> **Note**
>
> - To dynamically register the bridge devices, you only need to edit the bridge configuration file. For information about the sample code, see Use the basic features.
> - All parameters in the popClientprofile file must be specified. If a MAC address is used by an existing device, the device is used as a bridge device.
> - The deviceName parameter and deviceSecret parameters must be left empty. If you have specified the information about a bridge device, the device cannot be dynamically registered.
> - We recommend that you use dedicated test devices for debugging. Do not debug programs on local machines to prevent possible impacts on production environment.
>
>   If you debug the programs on the local machines, the MAC addresses of the machines are registered as bridge names. The bridges are associated with all devices in the *devices.conf* file.

Parameters

Parameter	Required	Description
productKey	Yes	The ProductKey of the product to which the bridge device belongs.
subDeviceConnectMode	No	The mode that devices use to connect with the bridge. • If this parameter is set to 3, a large-size bridge is created. A maximum of 500,000 devices can connect with the bridge. • If this parameter is not specified, a small-size bridge is created. A maximum of 15,00 devices can connect with the bridge. Large-size bridges and small-size bridges use different policies to disconnect devices. For more information, see Disconnect a device from IoT Platform.
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway. The bridge and IoT Platform establish a persistent connection over the HTTP/2 protocol. Endpoint format: • Public instance: `https://${productKey}.iot-as-http2. ${RegionId}.aliyuncs.com:443`. Replace *${productKey}* with the ProductKey of the product to which your bridge device belongs. Replace *${RegionId}* with the ID of the region where you purchased the IoT Platform service. For information about region IDs, see Regions and zones. For example, if the ProductKey of a bridge device is a1abcab**** and the IoT Platform service is purchased in the China (Shanghai) region, the endpoint is `https://a1abcab****.iot-as-http2.cn-shanghai.aliyuncs.com:443`. • Enterprise Edition instance: `https://${IotInstanceId}.http2.iothub.aliyuncs.com:443`. Replace *${IotInstanceId}* with the ID of the purchased instance. For example, if the instance ID is iot-cn-g06kwb****, the endpoint is `https://iot-cn-g06kwb****.http2.iothub.aliyuncs.com:443`.

Parameter	Required	Description
authEndpoint	Yes	The endpoint of the device authentication server. Endpoint format: • Public instance: `https://iot-auth. .${RegionId}.aliyuncs.com/auth/bridge`. Replace *${RegionId}* with the ID of the region where you purchased the IoT Platform service. For information about region IDs, see Regions and zones. For example, if the IoT Platform service is purchased in the China (Shanghai) region, the endpoint is `https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`. • Enterprise Edition instance: `https://${IotInstanceId}.auth.iothub.aliyuncs.com/auth/bridge`. Replace *${IotInstanceId}* with the ID of the purchased instance. For example, if the instance ID is iot-cn-g06kwb****, the endpoint is `https://iot-cn-g06kwb****.auth.iothub.aliyuncs.com/auth/bridge`.
popClientProfile	Yes	If you specify this parameter, the IoT as Bridge SDK calls the related IoT Platform API operation to create a bridge device. The following table describes the parameters of popClientProfile.

popClientProfile

Parameter	Required	Description
accessKey	Yes	The AccessKey ID of your Alibaba Cloud account. To create or view an AccessKey pair, log on to the IoT Platform console, move the pointer over your profile picture, and then click **AccessKey Management**. On the Security Management page, you can create or view an AccessKey pair.
accessSecret	Yes	The AccessKey secret of you Alibaba Cloud account.
name	Yes	The ID of the region to which the bridge device belongs. For more information about region IDs, see Regions and zones.
region	Yes	
product	Yes	The name of the product. Set the value to Iot.

Parameter	Required	Description
endpoint	Yes	The endpoint of the APIs in the specified region. Format: `iot.${RegionId}.aliyuncs.com`. Replace *${RegionId}* with the ID of the region where you purchased the IoT Platform service. For information about region IDs, see Regions and zones. For example, if the region is China (Shanghai), the endpoint is `iot.cn-shanghai.aliyuncs.com`.

In this example, a public instance is used to dynamically register a small-size bridge device.

```
# The endpoint of the server.
http2Endpoint = "https://${YourProductKey}.iot-as-http2.cn-shanghai.aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge"
# The information of the bridge device.
productKey = ${YourProductKey}
popClientProfile = {
  accessKey = ${YourAliyunAccessKey}
  accessSecret = ${YourAliyunAccessSecret}
  name = cn-shanghai
  region = cn-shanghai
  product = Iot
  endpoint = iot.cn-shanghai.aliyuncs.com
}
```

Call the operations to submit Thing Specification Language (TSL) data

The IoT as Bridge SDK encapsulates the operations to submit data. You can call the reportProperty operation to submit properties, call the fireEvent operation to submit events, and call the updateDeviceTag operation to update tags.

> **Note**
>
> - Before you call the reportProperty() and fireEvent() operations, you must define properties and events in the IoT Platform console. Log on to the IoT Platform console and go to the **Product Details** page. Then, you can define properties and events on the **Define Feature** tab. For more information, see Add a TSL feature.
> - If a tag already exists when you call the updateDeviceTag operation, the tag value is updated. To view existing tags, go to the **Device Details** page of the IoT Platform console. If a tag does not exist, IoT Platform automatically creates the tag.

Sample code:

```
TslUplinkHandler tslUplinkHandler = new TslUplinkHandler();
// Submit a property.
// The testProp property is defined.
String requestId = String.valueOf(random.nextInt(1000));
// The timestamp is not included in the submitted property data.
tslUplinkHandler.reportProperty(requestId, originalIdentity, "testProp", random.nextInt(100));
// The timestamp is included in the submitted property data.
//tslUplinkHandler.reportProperty(requestId, originalIdentity, "testProp", random.nextInt(100), System.currentTimeMillis());
// Submit an event.
// The testEvent event is defined.
requestId = String.valueOf(random.nextInt(1000));
HashMap<String, Object> params = new HashMap<String, Object>();
params.put("testEventParam", 123);
// The timestamp is not included in the submitted event data.
tslUplinkHandler.fireEvent(originalIdentity, "testEvent", ThingEventTypes.INFO, params);
// The timestamp is included in the submitted event data.
//tslUplinkHandler.fireEvent(originalIdentity, "testEvent", ThingEventTypes.INFO, params, System.currentTimeMillis());
// Update a device tag.
// The key of the tag is set to testDeviceTag.
requestId = String.valueOf(random.nextInt(1000));
tslUplinkHandler.updateDeviceTag(requestId, originalIdentity, "testDeviceTag", String.valueOf(random.nextInt(1000)));
```

The following table describes the parameters.

Parameter	Description
requestId	The ID of the request.
originalIdentity	The original identifier of the device.
testProp	The identifier of the property. In this example, the identifier of the defined property is testProp. The testProp property is submitted to IoT Platform.
random.nextInt(100)	The value of the property. When you define a property, you can set a value range. In this example, `random.nextInt(100)` indicates a random integer that is less than 100.
testEvent	The identifier of the event. In this example, the identifier of a defined event is testEvent. In this example, the testEvent event is submitted to IoT Platform.
ThingEventTypes.INFO	The type of the event. ThingEventTypes specifies the event type. A value of *INFO* indicates that the event type is Info. In this example, the event type is set to INFO when the testEvent event is defined in the IoT Platform console. If the event type is ERROR, set this parameter to `ThingEventTypes.ERROR`.

Parameter	Description
params	The output parameters of the event. The identifiers, data types, and value ranges of the output parameters are defined in the IoT Platform console. In this example, the **identifier** of the output parameter is testEventParam, and the value is 123.
testDeviceTag	The key of the tag. The data type is string. In this example, the key is testDeviceTag. Set the key of the tag as instructed based on your business requirements. For more information, see Device tags.
String.valueOf(random.nextInt(1000))	The value of the tag. The data type is string. In this example, `String.valueOf(random.nextInt(1000))` indicates a random integer that is less than 1000. Set the value of the tag as instructed based on your business requirements. For more information, see Device tags.
System.currentTimeMillis()	The current system time, in milliseconds.

Calls the operations to submit multiple properties and events at the same time

The IoT as Bridge SDK encapsulates the operations to submit multiple properties and events at the same time. You must create the BatchPostEventPropertyMessage object and call the addProperty() and addEvent() methods to add property and event data. Then, create the TslUplinkHandler object and call the BatchPostEventPropertyMessage() method to submit data.

> **Note**
>
> Before you call the reportProperty() and fireEvent() operations, you must define properties and events in the IoT Platform console. Log on to the IoT Platform console and go to the **Product Details** page. Then, you can define properties and events on the **Define Feature** tab. For more information, see Add a TSL feature.

Sample code:

```
TslUplinkHandler tslUplinkHandler = new TslUplinkHandler();
// Submit multiple properties and events at the same time.
String requestId = String.valueOf(random.nextInt(1000));
long startTime = System.currentTimeMillis() - 3000;
// Create a message to submit data.
BatchPostEventPropertyMessage batchPostEventPropertyMessage = new BatchPostEventPropertyMessage
();
Map<String, Object> aiEventParams = new HashMap<>();
aiEventParams.put("EventContent", "hello world");
batchPostEventPropertyMessage
  .addProperty("PowerConsumption", 1000, startTime)
  .addProperty("PowerConsumption", 123, startTime + 1000)
  .addProperty("LightAdjustLevel", 23, startTime)
  .addProperty("LightAdjustLevel", 44, startTime + 1000)
  .addProperty("LightAdjustLevel", 47, startTime + 2000)
  .addEvent("AIEvent", aiEventParams, startTime);
batchPostEventPropertyMessage.setId(requestId);
// Submit the data.
tslUplinkHandler.batchPostEventPropertyMessage(originalIdentity, batchPostEventPropertyMessage);
```

The following table describes the parameters.

Parameter	Description
requestId	The ID of the request.
startTime	The timestamps of the properties and events to be submitted. Unit: milliseconds. The time is displayed in UTC. You can customize this parameter based on your business requirements.
aiEventParams	The information about the event.
PowerConsumption	The identifiers of the properties. In this example, the identifiers of the defined properties include PowerConsumption and LightAdjustLevel. The values of the two properties at different time points are submitted.
LightAdjustLevel	
AIEvent	The identifier of the event. In this example, the identifier of the defined event is AIEvent. The AIEvent event is submitted.
originalIdentity	The original identifier of the device.

Call the operations to set properties and call services

The IoT as Bridge SDK encapsulates the PropertySetHandler operation to set properties and the ServiceInvokeHandler operation to call services. Devices can receive commands from IoT Platform and update data.

Sample code:

```
BridgeBootstrap bridgeBootstrap = new BridgeBootstrap();
// Set properties.
bridgeBootstrap.setPropertySetHandler(new PropertySetHandler() {
  @Override
  public void onPropertySet(PropertySetMessage msg) {
    log.info("on property set, {}", msg.getParams());
    // If you call the replySuccess() method, the SDK sends the /property/set_reply message to IoT Platform. The response code is 200.
    msg.replySuccess();
    // If you call the replyFail() method, the SDK sends the /property/set_reply message to IoT Platform. You can specify the response code as needed.
    //msg.replyFail(400);
  }
});
// Call services.
bridgeBootstrap.setServiceInvokeHandler(new ServiceInvokeHandler() {
  @Override
  public void onServiceInvoke(ServiceInvokeMessage message) {
    log.info("on service invoke, {}", message.getParams());
    // If you call the replySuccess() method, the SDK sends the /service/{service.identifier}_reply message to IoT Platform. The response code is 200.
    message.replySuccess();
    // If you call the replyFail() method, the SDK sends the /service/{service.identifier}_reply message to IoT Platform. You can specify the response code as needed.
    //msg.replyFail(400);
  }
});
```

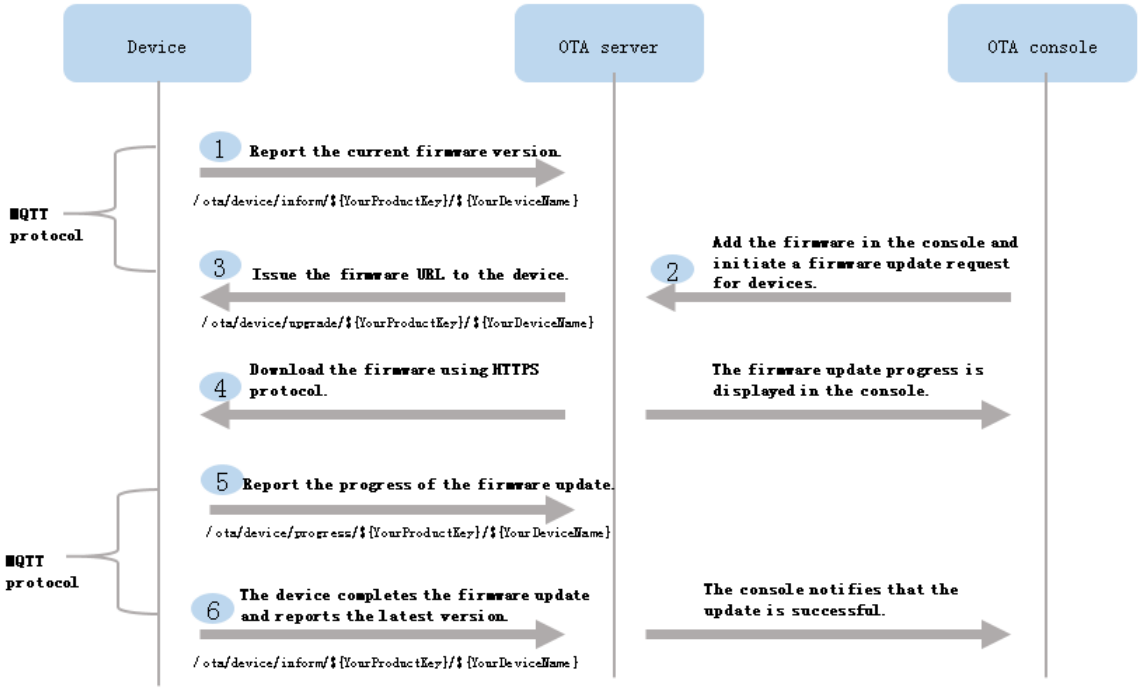
8.4. OTA updates

The IoT as Bridge SDK 2.1.3 and later support over-the-air (OTA) updates for firmware. This article describes how to use the IoT as Bridge SDK to update the firmware of a device over the air.

Background information

> **Note** Only the IoT as Bridge SDK 2.1.3 and later support OTA updates.

The following figure shows the process of an OTA update.



For more information about how to push an update package from IoT Platform to devices, see Overview.

For more information about the update process and how to specify topics and data formats for firmware updates, see Update devices by using OTA.

Call operations for OTA updates

The IoT as Bridge SDK encapsulates the operations that are related to OTA updates. To achieve OTA updates, you must use the SDK to call the following three operations:

- The operation that is used to submit versions of device firmware

  After a device is enabled and updated, the device must submit the current version of the firmware to the following topic: `/ota/device/inform/${YourProductKey}/${YourDeviceName}`.

  To enable a device to submit a firmware version, you must call the TslUplinkHandler.reportFirmwareVersion operation. Syntax:

  ```
  /**
   * The device submits the firmware version to IoT Platform.
   * @param requestId: the ID of the request
   * @param originalIdentity: the original identifier of the device
   * @param version: the firmware version to be submitted
   * @ return: The value true is returned if the firmware version is submitted.
  */
  boolean reportOtaVersion(String requestId, String originalIdentity, String version)
  ```

  Example

  ```
  TslUplinkHandler tslUplinkHandler = new TslUplinkHandler();
  tslUplinkHandler.doOnline(session, originalIdentity);
  tslUplinkHandler.reportOtaVersion("12345", originalIdentity, "1.0.1");
  ```

- The operation that is used to push update notifications from IoT Platform to devices

After you add update packages to IoT Platform and initiate OTA updates for multiple devices, IoT Platform pushes update notifications to the devices. The devices must subscribe to the following topic to retrieve update notifications: `/ota/device/upgrade/${YourProductKey}/${YourDeviceName}`.

To enable a device to receive the OTA update notification that is pushed from IoT Platform, you can call the BridgeBootStrap.setOtaUpgradeHandler() operation and configure a callback. Syntax:

```
/**
 * Configure a callback to receive the OTA update notifications that are pushed from IoT Platform.
 * @ param otaUpgradeHandler: the callback that is set.
*/
public void setOtaUpgradeHandler(OtaUpgradeHandler otaUpgradeHandler) {
  this.callback.setOtaUpgradeHandler(otaUpgradeHandler);
}
public interface OtaUpgradeHandler {
  /**
   * Push OTA update notifications from IoT Platform.
   * @param requestId: the ID of the request
   * @param firmwareInfo: the information about OTA updates
   * @param session: the current session
   * @return
   */
  boolean onUpgrade(String requestId, OtaFirmwareInfo firmwareInfo, Session session);
}
public class OtaFirmwareInfo {
  /**
   * The size of the update package.
   */
  private long size;
  /**
   * Valid signature methods: MD5 and SHA256.
   */
  private String signMethod;
  /**
   * The signature of the update package.
   */
  private String sign;
  /**
   * The version of the update package.
   */
  private String version;
  /**
   * The download URL of the update package.
   */
  private String url;
}
```

Example

```
bridgeBootstrap.setOtaUpgradeHandler(new OtaUpgradeHandler() {
  @Override
  public boolean onUpgrade(String requestId, OtaFirmwareInfo firmwareInfo, Session session) {
    log.info("ota onUpgrade, requestId:{}, firmware:{}, identity:{}",
      requestId, firmwareInfo, session.getOriginalIdentity());
    // Implement the OTA update.
    return true;
  }
});
```

- The operation that is used to submit the update progress

  After a firmware update starts, a device must submit the update progress to the following topic: `/ota/device/progress/${YourProductKey}/${YourDeviceName}`.

  To enable a device to submit the update progress, you can call the TslUplinkHandler.reportOtaProgress operation. Syntax:

  ```
  /**
   * Submit the OTA update progress.
   * @param requestId: the ID of the request
   * @param originalIdentity: the original identifier of the device
   * @param step: the current progress
   * @param desc: the description
   * @return
  */
  boolean reportOtaProgress(String requestId, String originalIdentity, String step, String desc)
  ```

  Example

  ```
  tslUplinkHandler.reportOtaProgress("7979", session.getOriginalIdentity(), "100", "ota success");
  ```