

ROMA Connect

# Best Practices

Issue 01  
Date 2023-08-28



**Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base  
Bantian, Longgang  
Shenzhen 518129  
People's Republic of China

Website: <https://www.huawei.com>

Email: [support@huawei.com](mailto:support@huawei.com)

---

# Contents

---

|  |           |
|--|-----------|
| <b>1 Digital Reconstruction of Traditional Parking Lot Management Systems.....</b> | <b>1</b>  |
| 1.1 Solution Overview.....   | 1         |
| 1.2 Registering a Device on ROMA Connect.....                                      | 3         |
| 1.3 Configuring MQS for Device Data Forwarding.....                                | 6         |
| 1.4 Creating an MQS-to-Database Integration Task.....                              | 9         |
| 1.5 Creating and Opening a Data API.....   | 14        |
| 1.6 Connecting the Device to ROMA Connect.....                                     | 17        |
| <b>2 Sharing Enterprise Data Using APIs.....</b>                                   | <b>19</b> |
| 2.1 Solution Overview.....   | 19        |
| 2.2 Opening Data Through Data APIs.....  | 20        |
| 2.3 Opening Data Through Function APIs.....  | 28        |
| <b>3 Integrating and Converting Service Data Across Systems.....</b>               | <b>35</b> |
| 3.1 Solution Overview.....   | 35        |
| 3.2 Configuring Data Integration Between Systems.....                              | 36        |
| <b>4 Building an Enterprise Service Open Platform.....</b>                         | <b>43</b> |
| 4.1 Solution Overview.....   | 43        |
| 4.2 Configuring the Service Opening Platform.....                                  | 45        |
| <b>5 Developing a Custom Authorizer with a Custom Backend.....</b>                 | <b>51</b> |
| 5.1 Solution Overview.....   | 51        |
| 5.2 Developing a Custom Authorizer.....  | 52        |
| 5.3 Using a Custom Authorizer.....   | 55        |

# 1 Digital Reconstruction of Traditional Parking Lot Management Systems

---

[Solution Overview](#)

[Registering a Device on ROMA Connect](#)

[Configuring MQS for Device Data Forwarding](#)

[Creating an MQS-to-Database Integration Task](#)

[Creating and Opening a Data API](#)

[Connecting the Device to ROMA Connect](#)

## 1.1 Solution Overview

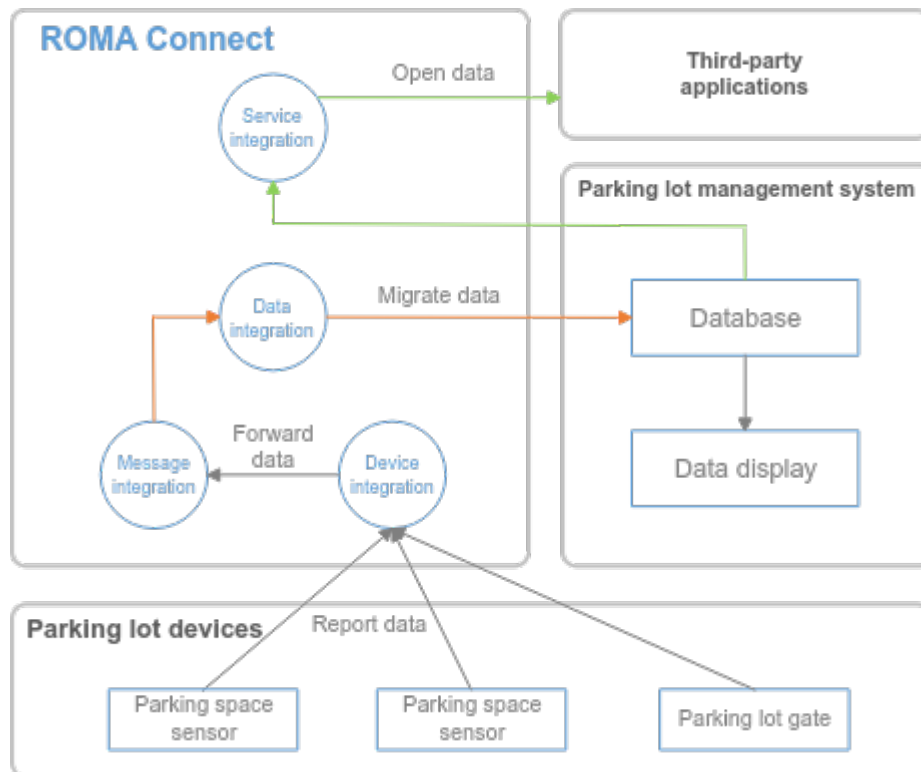
### Solution Background

Parking is becoming a prominent issue as the number of cars keeps increasing in cities.

- Traditional parking lot management is human-reliant. It is difficult to obtain an overview of the parking space in real time. Vehicle entry and exit and parking payment are time-consuming, and parking space utilization is inefficient.
- Vehicle users cannot learn about the available parking space in real time. It takes a lot of time to find parking space, which further increases the traffic pressure.
- The transportation department cannot obtain city-wide usage of parking space quickly, and therefore congestion mitigation measures for parking problems tend to be backward.

### Solution

This section describes how to use ROMA Connect to digitally reconstruct the management system of a traditional parking lot to implement intelligent management of vehicle entry and exit, charging, and parking space status.



1. Parking space sensors and access control devices are connected to the ROMA Connect through the device integration module to report the parking space status and vehicle entry and exit information in real time.
2. Data reported by devices is forwarded by the rule engine to the message integration topic for storage.
3. A data integration task writes the parking lot data in the topics into the database of the parking lot management system in real time. The data is used for parking space monitoring and charging, and can be displayed on the IOC screen.
4. The parking space status in the database is opened to third-party applications through the data API so that vehicle owners and transportation departments can learn about the parking space status.

The following shows the process of using ROMA Connect to implement digital reconstruction of the parking lot management system:

1. **Registering a Device on ROMA Connect**
2. **Configuring MQS for Device Data Forwarding**
3. **Creating an MQS-to-Database Integration Task**
4. **Creating and Opening a Data API**
5. **Connecting the Device to ROMA Connect**

## 1.2 Registering a Device on ROMA Connect

### Overview

After a device is registered on ROMA Connect, a unique ID and key are allocated to the device so that the device can access ROMA Connect.

### Prerequisites

- ROMA Connect can communicate with the parking lot devices and the parking lot management system. If the ROMA Connect instances communicate with each other through the public network, an elastic IP address (EIP) must be bound to each instance.
- Parking lot devices include parking space sensors and entrance and exit gate devices. The configuration processes are similar. The following describes how to configure parking space sensors.

### Procedure

1. Create an integration application.
  - a. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.
  - b. In the navigation pane on the left, choose **Integration Applications**. In the upper right corner of the page, click **Create Integration Application**.
  - c. In the dialog box displayed, set **Name** and click **OK**.
2. Define products on ROMA Connect based on the capability model of parking space sensors.
  - a. In the navigation pane on the left, choose **LINK > Product Management**. On the page displayed, click **Create Product** in the upper right corner.
  - b. In the **Create Product** dialog box, set product parameters and click **OK**.

**Table 1-1** Product parameters

| Parameter               | Description   |
|-------------------------|---|
| Product Type            | If the parking lot device is directly connected to ROMA Connect through the network, select <b>Common</b> .                   |
| Protocol Type           | Set this parameter to <b>MQTT</b> because the parking lot device connects to ROMA Connect through the integrated MQTT client. |
| Integration application | Select the integration application created in <b>1</b> .  |
| Product Template        | This parameter is not set in this practice.   |

| Parameter         | Description  |
|-------------------|--|
| Product Name      | Enter the name of the product. It can be user-defined.                             |
| Manufacturer Name | Enter the manufacturer name of the parking lot device.                             |
| Manufacturer ID   | Enter the manufacturer ID of the parking lot device.                               |
| Product Model     | Enter the product model of the parking lot device.                                 |
| Device Type       | Select <b>Default Type</b> as the type of the access device.                       |
| Model Version     | Enter the model version of the device. This parameter is not set in this practice. |
| Description       | Enter a brief description of the product.  |

- c. On the **Product Management** page, click the **Products** tab and click the name of the product created in [2.b](#) to access the product details page.
- d. On the **Thing Model** tab page, click **Create**.
- e. In the **Create Thing Model Service** dialog box, set thing model service parameters and click **OK**.

**Table 1-2** Thing model service configuration

| Parameter   | Description   |
|-------------|---|
| Name        | Name of the thing model service. Set this parameter as required, for example, <b>ParkingStatus</b> .          |
| Status      | This parameter specifies whether to enable the thing model service. Retain the default value <b>Enabled</b> . |
| Description | Enter a brief description of the thing model service.   |

- f. Select the thing model service created in [2.e](#), and click **Create** on the **Attributes** tab page of the thing model service on the right.
- g. In the **Create Attribute** dialog box, set attribute parameters and click **OK**.

The following uses the configuration of the parking space sensor as an example. The parking space sensor reports the following information:

- **deviceld** indicates the physical identifier of the device. The data type is **String**.
- **status** indicates the parking space attribute. The data type is **String**. The value **0** indicates vacant, and **1** indicates occupied.

**Table 1-3** Service attribute configuration

| Parameter        | deviceId Description  | status Description  |
|------------------|---|---|
| Name             | Name of the attribute parameter reported by the device. Set this parameter to <b>deviceId</b> .   | Name of the attribute parameter reported by the device. Set this parameter to <b>status</b> .   |
| Data Type        | Select <b>String</b> as the data type of the attribute parameter reported by the device.  | Select <b>String</b> as the data type of the attribute parameter reported by the device.  |
| Mandatory        | This parameter specifies whether the attribute must be reported by a device. Retain the default value, that is, the attribute must be reported by the device. | This parameter specifies whether the attribute must be reported by a device. Retain the default value, that is, the attribute must be reported by the device. |
| Description      | Enter a brief description of the attribute.   | Enter a brief description of the attribute.   |
| Max. Data Length | Enter <b>64</b> as the maximum length of the attribute value.   | Enter <b>16</b> as the maximum length of the attribute value.   |
| Enumerated Value | You do not need to set this parameter.  | Enumerated value of the attribute. Set this parameter to <b>0</b> or <b>1</b> , indicating that the parking space is vacant or occupied.                      |

3. Register a device on ROMA Connect.  
Each parking lot device must be registered on ROMA Connect.
  - a. In the navigation pane on the left, choose **LINK > Device Management**. On the page displayed, click **Create Device** in the upper right corner.
  - b. In the **Create Device** dialog box, set device parameters and click **OK**.

**Table 1-4** Device configuration

| Parameter               | Description  |
|-------------------------|--|
| Integration Application | Select the integration application created in <a href="#">1</a> .                            |
| Associated Product      | Select the product created in <a href="#">2</a> .  |
| Device ID               | Enter the physical identifier of the device, such as the IMEI and MAC address of the device. |



| Parameter        | Description   |
|------------------|---|
| Device Name      | Enter the name of a new device.   |
| Password         | Enter the access password of the device. If you do not specify this parameter, the system automatically generates a value.  |
| Confirm Password | Enter the password for confirmation. If you do not set this parameter, the system automatically generates one.  |
| Status           | This parameter specifies whether to enable the device. The device can connect to ROMA Connect only after being enabled. Retain the default value <b>Enabled</b> . |
| Device Tag       | Add tags for the device to facilitate search.   |
| Description      | Enter a brief description of the device.  |

## 1.3 Configuring MQS for Device Data Forwarding

### Overview

ROMA Connect does not directly store data reported by devices. You must configure data forwarding rules to forward device data to other services for storage. In this practice, device data is forwarded to ROMA Connect's MQS for integration.

### Procedure

1. Create a message topic for storing device data.
  - a. In the navigation pane on the left, choose **Message Queue Service > Topic Management**. On the page displayed, click **Create Topic** in the upper right corner.
  - b. In the **Create Topic** dialog box, set topic parameters and click **OK**.

**Table 1-5** Topic configuration

| Parameter               | Description  |
|-------------------------|--|
| Topic Name              | Enter the topic name, which is user-defined.   |
| Integration Application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> . |

| Parameter               | Description  |
|-------------------------|--|
| Permission              | Select the topic permission for the integration application to which the topic belongs. In this example, select <b>Publish/Subscribe</b> , which indicates that the topic can be used to produce and consume messages. |
| Partitions              | The appropriate number of partitions can improve the concurrent performance of message creation and retrieval.   |
| Replicas                | ROMA Connect automatically backs up data on each replica. If one replica is faulty, data will still be available. The reliability increases with the number of replicas of a topic.                                    |
| Aging Time (h)          | After the aging time expires, messages stored in the topic are deleted.  |
| Synchronous Replication | Whether to copy messages to all replicas before returning responses to the message client when the client produces messages to the topic. This parameter is not enabled here.  |
| Synchronous Flushing    | Whether to write each message produced by the message client to the topic to the disk immediately. This parameter is not enabled here.   |
| Tag                     | A tag can be added for a topic to facilitate topic search.   |
| Filter                  | (Optional) Add values to block the display of generated topic messages containing these values.  |
| Description             | Enter a brief description of the topic.  |

2. Configure data forwarding rules.
  - a. In the navigation pane on the left, choose **LINK > Rule Engine**. On the page displayed, click **Create Rule** in the upper right corner.
  - b. In the **Create Rule** dialog box, set rule parameters and click **OK**.

**Table 1-6** Rule configuration

| Parameter               | Description  |
|-------------------------|--|
| Integration application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> . |
| Rule Name               | Enter the rule name, which is user-defined.  |
| Description             | Enter a brief description of the rule.   |

| Parameter | Description  |
|-----------|--|
| Status    | This parameter specifies whether to enable the rule. Retain the default value <b>Enabled</b> . |

- c. In the rule list, click the name of the rule created in [2.b](#) to access the rule details page.
- d. In the **Source** area, click **Create Source**, set related parameters, and click **Save**.

**Table 1-7** Source configuration

| Parameter           | Description   |
|---------------------|---|
| Product Name        | Select the product created in <a href="#">Registering a Device on ROMA Connect</a> .  |
| Device Name         | Select <b>All devices</b> .   |
| Topic Name          | Select the topic used by the device to send messages. The format is <code>{Product ID}/out/+</code> .   |
| Topic Level         | Select a topic level. The value is automatically adapted based on the value of <b>Device Name</b> . If you do not specify <b>Device Name</b> , <b>Topic Level</b> is <b>Product-level</b> . If you select a specific device from the <b>Device Name</b> drop-down list box, <b>Topic Level</b> is <b>Device-level</b> . |
| Base64 Encoding     | This parameter specifies whether Base64 encoding is performed on the forwarded device data. This function is disabled here.   |
| Include Device Data | Specifies whether the forwarded data includes device data. To facilitate subsequent identification of the device to which the forwarded data belongs, enable this option.   |

- e. In the **Destination** area, click **Create Destination**, set related parameters and click **Save**.

**Table 1-8** Destination configuration

| Parameter          | Description   |
|--------------------|---|
| Destination        | In this practice, data is forwarded to the MQS of ROMA Connect. Therefore, select <b>ROMA MQS</b> .   |
| Connection Address | Select the MQS intranet address of the current ROMA Connect instance. You can view the address on the <b>Instance Information</b> page of the ROMA Connect console. |

| Parameter  | Description   |
|------------|---|
| Topic Name | Select the message topic created in <a href="#">1</a> .   |
| Username   | <b>This parameter is mandatory only if MQS SASL_SSL is enabled for the ROMA Connect instance.</b><br>Enter the key of the integration application to which the topic defined in <b>Topic Name</b> belongs.    |
| Password   | <b>This parameter is mandatory only if MQS SASL_SSL is enabled for the ROMA Connect instance.</b><br>Enter the secret of the integration application to which the topic defined in <b>Topic Name</b> belongs. |

## 1.4 Creating an MQS-to-Database Integration Task

### Overview

A data integration task is performed to convert the structure of device data in MQS and write the converted data into the database of the parking lot management system. The parking lot management system database uses MySQL as an example.

### Procedure

1. Connect to the MQS data source.
  - a. In the navigation pane on the left, choose **Data Sources**. In the upper right corner of the page, click **Access Data Source**.
  - b. On the **Default** tab page, select **MQS** and click **Next**.
  - c. Enter the configuration information about the MQS data source.

**Table 1-9** MQS data source configuration

| Parameter               | Description  |
|-------------------------|--|
| Name                    | Enter the name of the MQS data source, which is user-defined.  |
| Integration Application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> .   |
| Description             | Enter a brief description of the data source.  |
| Connection Address      | Select the MQS intranet address of the current ROMA Connect instance. You can view the MQS intranet address on the <b>Instance Information</b> page of the ROMA Connect console. |

| Parameter                           | Description  |
|-------------------------------------|--|
| Enable SSL                          | Set this parameter to <b>Yes</b> if <b>MQS SASL_SSL</b> is enabled and <b>Intra-VPC Plaintext Access</b> is disabled for the ROMA Connect instance. In other cases, set this parameter to <b>No</b> .                    |
| SSL Username/<br>Application Key    | Mandatory for <b>Enable SSL</b> set to <b>Yes</b> .<br>Set this parameter to the key of the integration application to which the topic created in <a href="#">Configuring MQS for Device Data Forwarding</a> belongs.    |
| SSL Password/<br>Application Secret | Mandatory for <b>Enable SSL</b> set to <b>Yes</b> .<br>Set this parameter to the secret of the integration application to which the topic created in <a href="#">Configuring MQS for Device Data Forwarding</a> belongs. |

- d. After setting the parameters for the MQS data source, click **Check Connectivity** to test the data source connection.
    - If the test result is **Data source connected successfully**, click **Create**.
    - If the test result is **Failed to connect to the data source**, check and modify the connection parameters, and click **Recheck** until the connection is successful.
  - e. Click **Create**.
2. Connect to the MySQL data source.
    - a. On the **Data Sources** page, click **Access Data Source** in the upper right corner.
    - b. On the **Default** tab page, select **MySQL** and click **Next**.
    - c. Enter the configuration information about the MySQL data source.

**Table 1-10** MySQL data source configuration

| Parameter               | Description  |
|-------------------------|--|
| Name                    | Enter the name of the data source, which is user-defined.  |
| Integration Application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> . |
| Description             | Enter a brief description of the data source.  |
| Connection Mode         | Select <b>Default</b> .  |
| Connection Address      | Enter the IP address of the MySQL database.  |

| Parameter            | Description   |
|----------------------|---|
| Port                 | Enter the port number for accessing the MySQL database. The default port number is 3306. Set this parameter based on the actual database configuration. |
| Database Name        | Enter the name of the database where the table resides.   |
| Encoding Format      | Enter the encoding format used by the database.   |
| Timeout Interval (s) | Enter the timeout interval for connecting to the database, in seconds.  |
| Username             | Enter the username used for accessing the database. The user must have the read and write permissions on the database.                                  |
| Password             | Enter the password used for accessing the database.   |

- d. After setting the parameters for the MySQL data source, click **Check Connectivity** to test the data source connection.
    - If the test result is **Data source connected successfully**, go to the next step.
    - If the test result is **Failed to connect to the data source**, check and modify the connection parameters, and click **Recheck** until the connection is successful.
  - e. Click **Create**.
3. Create a data integration task from MQS to MySQL.
    - a. In the navigation pane on the left, choose **Fast Data Integration > Task Management** and click **Create Common Task**.
    - b. On the **Create Task** page, set the parameters of the data integration task.
      - i. Configure basic information about the task.

**Table 1-11** Basic task configuration

| Parameter        | Description  |
|------------------|--|
| Task Name        | Enter the task name, which is user-defined.  |
| Description      | Enter a brief description of the task.   |
| Integration Mode | Select the mode of data integration. When MQS is used as the source data source, only real-time tasks are supported. In this case, select <b>Real-Time</b> . |

| Parameter          | Description   |
|--------------------|---|
| Tag                | Add a tag to classify tasks for quick search. In this practice, you do not need to set this parameter.              |
| Enterprise Project | Select the enterprise project to which the task belongs. In this example, retain the default value <b>default</b> . |

- ii. Configure the MQS data source information at the source.

**Table 1-12** Source information configuration

| Parameter               | Description  |
|-------------------------|--|
| Instance                | Select the ROMA Connect instance that is being used.   |
| Integration Application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> .   |
| Data Source Type        | Select <b>MQS</b> .  |
| Data Source Name        | Select the MQS data source that you configured in <a href="#">1</a> .  |
| Topic Name              | Select the topic created in <a href="#">Configuring MQS for Device Data Forwarding</a> .   |
| Parse                   | The data structure of the source MQS is different from that of the destination MySQL database. You need to parse and convert the data. In this example, select <b>Yes</b> .  |
| Data Root Field         | This parameter specifies the path of the upper-layer common fields among all metadata in the data obtained from the source in JSON format. This parameter is not required in this practice.                                    |
| Data Type               | Select the format of data obtained from the MQS data source. The data format must be the same as the actual data format stored in MQS. The device data stored in MQS is in JSON format. In this practice, select <b>JSON</b> . |
| Offset                  | If you select <b>Latest</b> , the latest message data is integrated.   |

| Parameter | Description  |
|-----------|--|
| Metadata  | <p>This parameter specifies the JSON data elements that are obtained from the source and need to be integrated into the destination.</p> <p>Take the data of parking slot sensors as an example. Integrate sensor and parking slot information into the destination database.</p> <ul style="list-style-type: none"> <li>• <b>Alias: deviceId. Type: String. Parsing Path: data.deviceId.</b></li> <li>• <b>Alias: status. Type: String. Parsing Path: data.status.</b></li> </ul> |
| Time Zone | Select the time zone used by the MQS data source so that ROMA Connect can identify the data timestamps.  |

- iii. Configure the MySQL data source at the destination.

**Table 1-13** Destination information configuration

| Parameter               | Description  |
|-------------------------|--|
| Instance                | Select the ROMA Connect instance that is being used.   |
| Integration Application | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> .   |
| Data Source Type        | Select <b>MySQL</b> .  |
| Data Source Name        | Select the MySQL data source that you configured in <a href="#">2</a> .  |
| Table                   | <p>Select the data table to which the integrated data in the MySQL database will be written.</p> <p>After selecting a data table, click <b>Select Table Field</b> to select the column fields for writing the parking space sensor information and parking status information into the database.</p> |
| Batch Number Field      | This parameter is not required in this practice.   |

- iv. Configure the data mapping rule from MQS to MySQL.
- Click **Automatic Mapping**. The mapping rules between the source and destination data fields are automatically created. If the fields in the data tables at the two ends are inconsistent, you need to select the corresponding source fields for the destination fields.
- c. Click **Create**.



- d. On the **Task Management** page, click **Start** on the right of the created data integration task to start the task.

## 1.5 Creating and Opening a Data API

### Overview

Through the data API, the parking space status data in the parking lot management system is opened to third-party applications so that vehicle owners and transportation departments can learn about the parking space status.

### Procedure

1. Create a backend.
  - a. In the navigation pane on the left, choose **API Connect > Custom Backends**. On the **Backends** tab, click **Create Backend**.
  - b. Set backend parameters and click **Create**.

**Table 1-14** Backend configuration

| Parameter                       | Description  |
|---------------------------------|--|
| Name                            | Backend name.  |
| Integration Application         | Select the integration application created in <a href="#">Registering a Device on ROMA Connect</a> .   |
| Backend Request Method          | Request method of the backend. Set this parameter to <b>GET</b> here.  |
| Backend Request Path            | Enter the request path of the backend in the <i>/serviceName/interfaceName</i> format.   |
| Backend Security Authentication | Security authentication mode of the backend. Set this parameter to <b>None</b> here because the API to open and the backend service are in the same instance in this practice. |
| Description                     | Enter a brief description of the backend.  |
| Advanced Settings               | Retain the default settings in the <b>Advanced Settings</b> area.  |

After the backend is created, the online IDE of the backend is automatically displayed. The backend type is data backend by default.

2. Configure and deploy the data backend.
  - a. In the upper left corner, choose **New Data Backend > Add Data Source**.
  - b. On the **Add Data Source** page displayed, configure data source information and click **Add**.

**Table 1-15** Data source configuration

| Parameter             | Description   |
|-----------------------|---|
| Select Data Source    | Select the MySQL data source connected in <b>Creating an MQS-to-Database Integration Task</b> . |
| Select Statement Type | Select the type of the statement to be executed. In this example, select <b>SQL</b> .           |
| Advanced Settings     | Retain the default settings in the <b>Advanced Settings</b> area.                               |

- c. After adding a data source, select the data source on the left of the online IDE and add the following statement to the statement editing box on the right to obtain the number of vacant parking spaces from the database.

The following statement assumes that the name of the data table that stores parking space information and parking space status information is **ParkingData**, and the parking space status field in the table is **Status** in the MySQL database.

```
SELECT COUNT(Status) AS FreeNo FROM ParkingData WHERE Status="0";
```

- d. Click **Save** in the upper right corner of the page to save the backend configuration.
- e. Click **Test** in the upper right corner of the page, and then click **Test** in the **Test Parameters** pane below.

In the **Execution Result** column, check whether the following JSON data is returned. *xxx* indicates the number of vacant parking spaces.

```
{"default":[{"FreeNo":xxx}]}
```

- f. After the backend is tested, click **Deploy** in the upper right corner of the page. In the dialog box displayed, click **Yes** to deploy the data backend.
3. Publish the data API.
- a. After the backend deployment is complete, click **Publish** in the upper right corner of the page.
- b. On the page displayed, set the parameters and click **Publish** to create a frontend API for the backend and publish the API in an environment. After the API is published, the API details page is displayed.

**Table 1-16** Parameters for publishing a backend

| Parameter   | Description   |
|-------------|---|
| Group       | Select an API group for the frontend API. If no API group is available, click <b>Create API Group</b> on the right to create one. |
| Environment | Select the default environment <b>RELEASE</b> .   |

| Parameter                        | Description   |
|----------------------------------|---|
| Frontend Security Authentication | Select the authentication mode of the API. The <b>App</b> authentication mode is recommended. <ul style="list-style-type: none"> <li>• <b>App</b>: ROMA Connect authenticates API requests. When calling an API, a user gets authenticated using the key and secret of an integration application.</li> <li>• <b>IAM</b>: IAM authenticates API requests. When calling an API, a user gets authenticated using a token or an AK/SK pair.</li> <li>• <b>Custom</b>: The custom function API is used for authenticating API requests.</li> <li>• <b>None</b>: Authentication is not required for API requests.</li> </ul> |
| Custom Authorizer                | Mandatory for <b>Frontend Security Authentication</b> set to <b>Custom</b> .<br>Select a frontend custom authorizer you have created.   |
| Frontend Request Protocol        | Select the request protocol used by the frontend API. The value can be <b>HTTPS</b> , <b>HTTP</b> , or <b>HTTP&amp;HTTPS</b> . <b>HTTPS</b> is recommended for transmitting important or sensitive data.  |
| Timeout (ms)                     | Enter the timeout interval of a backend service request. The default value is <b>60000</b> .  |
| Retries                          | Number of retries after ROMA Connect fails to call the backend service. <ul style="list-style-type: none"> <li>• If the value is <b>-1</b>, this parameter is disabled.</li> <li>• If the value ranges from 0 to 10, this parameter is enabled, and retries are performed based on the configured value.</li> </ul>   |
| Advanced Settings                |   |
| Frontend Request Method          | Select the request method of the frontend API. <b>ANY</b> indicates that the API can be accessed using any request method.  |
| Frontend Request Path            | Enter the request path of the frontend API, for example, <b>/getUserInfo/userId</b> .<br>The request path is case sensitive.  |
| CORS                             | Whether the API supports cross-origin resource sharing (CORS). In this example, disable this parameter.   |

#### 4. Bind an independent domain name to the data API.

- a. On the API details page displayed after you publish the API, choose **Group Information** and click **Bind Independent Domain Name** in the **Independent Domain Names** area.
- b. In the displayed dialog box, enter the domain name and click **OK**.

**Table 1-17** Independent domain name configuration

| Parameter                      | Description  |
|--------------------------------|--|
| Domain Name                    | Enter the domain name to be bound.   |
| Minimum TLS Version            | The minimum TLS version that can be used to access the domain name.  |
| HTTP-to-HTTPS Auto Redirection | Whether to support HTTP-to-HTTPS redirection. Redirection takes effect only when the API request protocol is HTTPS or HTTP&HTTPS and an SSL certificate has been bound to the independent domain name.<br><b>NOTE</b><br>Redirection is only suitable for GET and HEAD requests. Redirecting other requests may cause data loss due to browser restrictions. |

- c. (Optional) For HTTPS-compatible data APIs, bind an SSL certificate to their independent domain name.
  - i. Click **Select SSL Certificate** on the right of the independent domain name.
  - ii. Select an SSL certificate and click **OK**.  
If none is available, click **Create SSL Certificate** to create one.
5. Switch to the **APIs** tab and obtain the API request method and URL for external applications to call the API.

## 1.6 Connecting the Device to ROMA Connect

### Overview

The parking space sensors in the parking lot connect to ROMA Connect through an integrated open-source MQTT client to report the parking space status.

### Procedure

1. Obtain the MQTT client.  
Obtain **Eclipse Paho MQTT Client** 3.1 or 3.1.1 based on the programming language you use.
2. Obtain device access information.
  - a. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.

- b. In the navigation pane on the left, choose **LINK > Device Management**. On the page displayed, obtain the device connection information.
  - MQTT/MQTTS connection address: Obtain the access address of the device from the upper part of the **Device Management** page and then download the SSL certificate. If MQTT is used for access, obtain the MQTT connection address. If MQTTS is used for access, obtain the MQTTS connection address.
  - Client ID/Username/Password: Locate the corresponding device on the **Devices** tab page and obtain **Client ID**, **Username**, and **Password** of the device.
  - Topic: Click the device name to access the device details page. On the **Topics** tab page, obtain the topic information about the messages reported (published) by the device. The topic name is in the following format: `{Product ID}/out/{Device ID}`.
3. Integrate the device with the MQTT client.

Develop the parking space sensor and integrate it with the **Eclipse Paho MQTT Client**. During the integration development, write the device access information and configure the parking space sensor to report the device information and parking space status when the parking space status changes. The data reported by the device is in JSON format. The format is as follows.

**deviceld** indicates the physical identifier of a device. The data type is **String**. **status** indicates the parking space status. The data type is **String**. The value can be **0** (vacant) or **1** (occupied).

```
{
  "deviceld": "xxxxxx",
  "status": "0/1"
}
```

For details, see [Configuring Device Integration](#).
4. Connect the device to ROMA Connect.

After the integration development is complete, power on the device to take it online and connect it to ROMA Connect. On the ROMA Connect console, choose **LINK > Device Management**. On the page displayed, **Status** of the device will be displayed as **Online**.
5. Configure the device to report parking space status data.

When the parking space status changes, the parking space sensor reports the parking space status data.

  - On the ROMA Connect instance console, choose **Message Queue Service > Message Query** and select the topic created in [Configuring MQS for Device Data Forwarding](#) to view the device data records forwarded to MQS.
  - On the ROMA Connect instance console, choose **Fast Data Integration > Task Management** and click the name of the task created in [Creating an MQS-to-Database Integration Task](#). On the **Log > Run Log** tab page of the task information, you can view the task execution success record.
  - In the database of the parking lot management system, you can view the parking space status data written into the data table.

# 2 Sharing Enterprise Data Using APIs

---

[Solution Overview](#)

[Opening Data Through Data APIs](#)

[Opening Data Through Function APIs](#)

## 2.1 Solution Overview

### Solution Background

With the continuous advancements of IT, enterprises have more and more service systems. Data needs to flow smoothly between these systems to improve operational efficiency. Enterprises are paying more and more attention to secure exchange of data between legacy and new service systems within an enterprise and even between the service systems of different enterprises.

- As an enterprise expands across regions, its service systems are deployed in its branches in different regions. Service systems need to be interconnected between the headquarters and the branches. If different service systems directly access the database of each other, the operation mode is too complex and database information leakage may occur.
- In their daily operation, enterprises produce and accumulate huge volumes of data assets. However, in most enterprises, data assets are confined only to internal use, resulting in low reuse of the assets. If the database is directly accessed by partners or third parties, it is difficult to ensure access security.

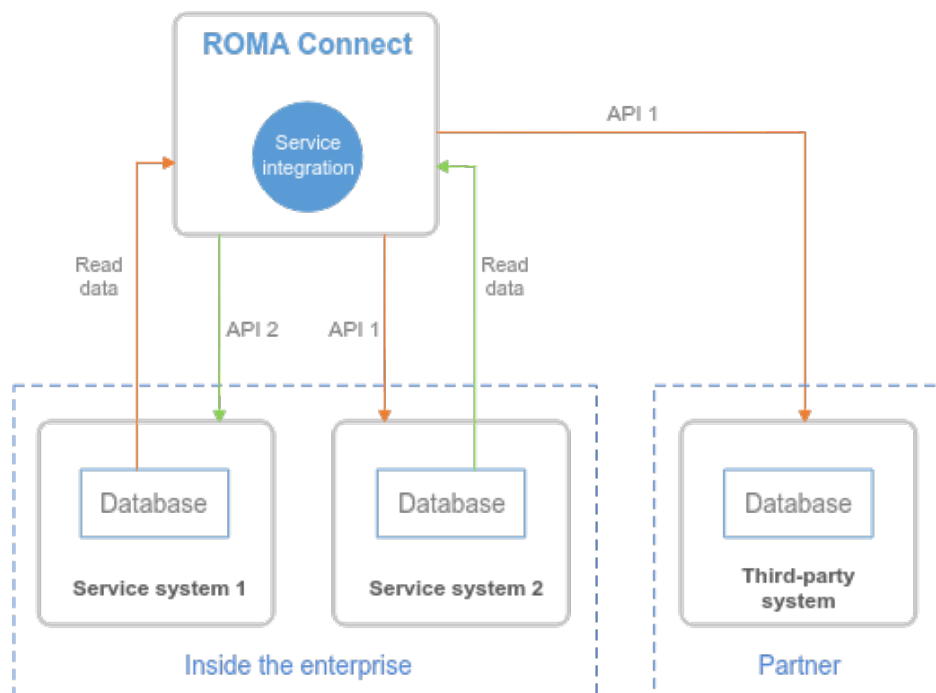
### Solution

API Connect (APIC) is a component of ROMA Connect for API integration. APIC can encapsulate existing backend services, data sources, and user-defined functions into standard RESTful APIs and expose them to external systems. This simplifies data sharing or service provisioning and reduces the cost of interconnection between enterprises.

This section describes how to use the APIC of ROMA Connect to quickly open up a database as RESTful APIs for data access between different service systems. In

addition, multiple security authentication modes can be selected to implement secure access across networks and regions.

- Data providers use RESTful APIs to open data in their databases to external systems. Through simple operations, they can quickly open data or services with low costs and risks to provide value-added services.
- Data users do not need to customize connection clients. To obtain data from various databases, they only need a client that can send simple RESTful requests.



The APIC of ROMA Connect directly reads data from the database, encapsulates the data into a RESTful API, and exposes the API to other service systems inside or outside the enterprise. Other service systems can call this API to obtain data from the database.

ROMA Connect APIC encapsulates database data into RESTful APIs in either of the following ways:

- Data API: Reads data from a database by SQL scripts. The operation is simple and convenient, but the flexibility is low.
- Function API: Reads data from a database by JavaScript. ROMA Connect provides a Java class `DataSourceClient` for reading database data. After reading data, you can use JavaScript to orchestrate and adapt data. The operation is complex, but the flexibility is higher.

## 2.2 Opening Data Through Data APIs

### Prerequisites

- The network where the service system is located can communicate with the network of ROMA Connect.

If the ROMA Connect instances communicate with each other through the public network, an elastic IP address (EIP) must be bound to each instance.

- The data source type of the service system database is supported by ROMA Connect.

For details about the data sources supported by data APIs, see [Data Sources Supported by Data APIs](#).

- Prepare an available independent domain name as the access domain name of the API.

You have configured domain name resolution from the independent domain name to the APIC connection address. For details, see [Adding an A Record Set](#).

If you do not have an independent domain name, you can apply for a domain name from the domain name registrar.

- If the data API uses the HTTPS request protocol, you need to apply for an SSL certificate for the independent domain name and obtain the SSL certificate content and key in PEM format.

## Exposing the Data API of the Service System

1. Create an integration application.

All resources in the ROMA Connect instance must belong to an integration application. Before creating other resources, make sure an integration application is available. If an integration application is available, skip this step.

- a. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.
- b. In the navigation pane on the left, choose **Integration Applications**. In the upper right corner of the page, click **Create Integration Application**.
- c. In the dialog box displayed, set **Name** and click **OK**.

2. Connect to a data source.

Configure the connection between ROMA Connect and service system databases to read data from the databases. The access configuration varies depending on the data source type. The MySQL database is used as an example. For details about how to access other types of databases, see [Connecting to Data Sources](#).

- a. In the navigation pane on the left, choose **Data Sources**. In the upper right corner of the page, click **Access Data Source**.
- b. On the **Default** tab page, select **MySQL** and click **Next**.
- c. Configure the data source connection information.

**Table 2-1** Data source connection information

| Parameter               | Description  |
|-------------------------|--|
| Name                    | Enter a custom data source name. Using naming rules facilitates future search. |
| Integration Application | Select the integration application to which the data source belongs.           |



| Parameter           | Description  |
|---------------------|--|
| Description         | Click <a href="#">🔗</a> to edit the description of the data source.  |
| Connection Mode     | Select a database connection mode. <ul style="list-style-type: none"><li>• <b>Default:</b> The system automatically concatenates data source connection character strings based on your configured data.</li><li>• <b>Professional:</b> You need to enter a data source connection string in JDBC format.</li></ul>  |
| Connection Address  | Mandatory for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the IP address and port number for connecting to the database.   |
| Database Name       | Mandatory for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the name of the database to be accessed.   |
| Encoding Format     | Available for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the encoding format used by the database.  |
| Timeout Interval(s) | Available for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the timeout interval for connecting to the database, in seconds.   |
| Connection String   | Mandatory for <b>Connection Mode</b> set to <b>Professional</b> .<br>Enter the JDBC connection string of the MySQL database, for example, <i>jdbc:mysql://{hostname}:{port}/{dbname}</i> . <ul style="list-style-type: none"><li>• <i>{hostname}</i> indicates the connection address of the database.</li><li>• <i>{port}</i> indicates the port number for connecting to the database.</li><li>• <i>{dbname}</i> indicates the name of the database to be connected.</li></ul> |
| Username            | Enter the username for connecting to the database.   |
| Password            | Enter the password for connecting to the database.   |

- d. After setting the parameters for the MySQL data source, click **Check Connectivity** to test the data source connectivity.
  - If the test result is **Data source connected successfully**, go to the next step.
  - If the test result is **Failed to connect to the data source**, check and modify the connection parameters, and click **Recheck** until the connection is successful.
- e. Click **Create**.

## 3. Create a data backend.

You can customize a backend to convert the data of a service system into a backend service.

- a. In the navigation pane on the left, choose **API Connect > Custom Backends**. On the **Backends** tab, click **Create Backend**.
- b. On the **Create Backend** page, set backend parameters and click **Create**.

**Table 2-2** Backend request information

| Parameter                       | Description   |
|---------------------------------|---|
| Name                            | Enter a backend name. Using naming rules facilitates future search.   |
| Integration Application         | Select the integration application to which the backend belongs.  |
| Backend Request Method          | Select the request method of the backend. The value can be <b>GET</b> , <b>POST</b> , <b>PUT</b> , or <b>DELETE</b> .   |
| Backend Request Path            | Enter the request path of the backend, for example, <b>/getUserInfo/userId</b> .<br>The request path is case sensitive.   |
| Backend Security Authentication | Select the security authentication mode of the backend. <ul style="list-style-type: none"><li>● <b>Signature key</b>: A signature key is used to authenticate the request sender. If a signature key is used for authentication, the same signature key must be bound to the API that calls the backend service.</li><li>● <b>None</b>: No authentication is required for calling requests.</li></ul> |
| Description                     | Enter a brief description of the backend.   |
| Advanced Settings               |   |
| Version                         | Enter the backend version, for example, <b>V1.0</b> . Customized backend version numbers differentiate backend services.  |

| Parameter        | Description   |
|------------------|---|
| Input Parameters | <p>Define request parameters of the backend service based on site requirements.</p> <p>In the <b>Input Parameters</b> area, click <b>Add Input Parameter</b> to add request parameters of the custom backend.</p> <ul style="list-style-type: none"> <li>● <b>Name:</b> name of a request parameter, which is user-defined.</li> <li>● <b>Parameter Location:</b> location of the request parameter in the backend request. The value can be <b>Headers</b> or <b>Parameters</b>.</li> <li>● <b>Default Value:</b> It is used as the default test value of parameters only in the subsequent custom backend test procedure. This parameter does not take effect during custom backend deployment.</li> <li>● <b>Mandatory:</b> specifies whether a request parameter is mandatory in a backend request.</li> <li>● <b>Description:</b> Enter the description of the parameter.</li> </ul> |
| Returned Type    | Select the response data format of the backend. The value can be <b>JSON</b> , <b>XML</b> , or <b>STREAM</b> .  |
| Formatting       | This parameter specifies whether to format the response message body based on the selected returned type.   |

After the backend is created, the online IDE of the backend is automatically displayed. The backend type is data backend by default.

- c. In the upper left corner, choose **New Data Backend > Add Data Source**.
- d. On the **Add Data Source** page displayed, configure data source information and click **Add**.

**Table 2-3** Data source configuration

| Parameter             | Description   |
|-----------------------|---|
| Select Data Source    | Select a data source that has been created in <b>Access Data Source</b> .   |
| Select Statement Type | <p>Select the type of the statement to be executed. The value can be <b>SQL</b> or <b>SP</b>.</p> <p>If the Redis or MongoDB data source is used, select <b>SQL</b>. The actual statement is NoSQL.</p> |
| Advanced Settings     |   |

| Parameter       | Description   |
|-----------------|---|
| Returned Object | Enter the name of the returned object. The execution result of the statement is encapsulated in the object and returned.  |
| Paging          | This parameter indicates whether the statement execution results are returned on multiple pages. If multiple data sources are added to the same data backend, the <b>Paging</b> parameter is not available. |
| Precompiling    | This parameter specifies whether to precompile execution statements to prevent SQL injection risks.   |

- e. After adding a data source, you can select the data source on the left of the IDE and add an execution statement in the statement editing box on the right to obtain the data to be returned.
  - Data backends support standard SQL syntax. For details about the syntax differences, see [Developing Custom Data Backends](#).
  - For the Redis or MongoDB data source, the data processing command of Redis or MongoDB is used.

 **NOTE**

If an SQL statement references backend request parameters of multiple data types, ROMA Connect converts the input parameters to the String type by default. Therefore, when the SQL statement is executed, the corresponding function needs to be called to convert non-String parameters.

- f. Click **Save** in the upper right corner of the page to save the backend configuration.
  - g. Test the backend functions.

In the upper right corner of the page, click **Test**. In the **Test Parameters** area, add request parameters based on the definition of the backend service and click **Test** to send the request. On the **Execution Result** tab page on the right, check whether the expected data is returned.
  - h. After the backend is tested, click **Deploy** in the upper right corner of the page. In the dialog box displayed, click **Yes** to deploy the backend.
4. Publish the data API.

Publish the data backend as a data API and deploy the API in the environment so that other service systems can call this API.

    - a. Click **Publish** in the upper right corner of the page.
    - b. On the page displayed, configure API information and click **Publish** to create a frontend API for the backend and publish the API in an environment.

After the API is published, the API details page is displayed.

**Table 2-4** Parameters for publishing an API

| Parameter                        | Description   |
|----------------------------------|---|
| Group                            | Select an API group for the frontend API. If no API group is available, click <b>Create API Group</b> on the right to create one.   |
| Environment                      | Select the environment in which the frontend API will be published. Select the default environment <b>RELEASE</b> when exposing an API.   |
| Frontend Security Authentication | Select the authentication mode of the API. The <b>App</b> authentication mode is recommended. <ul style="list-style-type: none"> <li>• <b>App</b>: ROMA Connect authenticates API requests. When calling an API, a user gets authenticated using the key and secret of an integration application.</li> <li>• <b>IAM</b>: IAM authenticates API requests. When calling an API, a user gets authenticated using a token or an AK/SK pair.</li> <li>• <b>Custom</b>: The custom function API is used for authenticating API requests.</li> <li>• <b>None</b>: Authentication is not required for API requests.</li> </ul> |
| Custom Authorizer                | Mandatory for <b>Frontend Security Authentication</b> set to <b>Custom</b> .<br>Select a frontend custom authorizer you have created.   |
| Frontend Request Protocol        | Select the request protocol used by the frontend API. The value can be <b>HTTPS</b> , <b>HTTP</b> , or <b>HTTP&amp;HTTPS</b> . <b>HTTPS</b> is recommended for transmitting important or sensitive data.  |
| Timeout (ms)                     | Enter the timeout interval of a backend service request. The default value is <b>60000</b> .  |
| Retries                          | Number of retries after ROMA Connect fails to call the backend service. <ul style="list-style-type: none"> <li>• If the value is <b>-1</b>, this parameter is disabled.</li> <li>• If the value ranges from <b>0</b> to <b>10</b>, this parameter is enabled, and retries are performed based on the configured value.</li> </ul>   |
| <b>Advanced Settings</b>         |   |
| Frontend Request Method          | Select the request method of the frontend API. <b>ANY</b> indicates that the API supports any request method. Generally, the API used to obtain database data uses the <b>GET</b> method.   |

| Parameter             | Description   |
|-----------------------|---|
| Frontend Request Path | Enter the request path of the frontend API, for example, <b>/getUserInfo/userId</b> .<br>The request path is case sensitive.  |
| CORS                  | Whether to enable CORS for the API.<br>For security purposes, a browser restricts cross-domain requests initiated from scripts. That is, only resources from the same domain can be requested. CORS allows the browser to send XMLHttpRequest requests to the server in a different domain. For details about how to call APIs for cross-domain access, see <a href="#">Configuring CORS for APIs</a> . |

5. Bind an independent domain name to the data API.
  - a. On the API details page displayed after you publish the API, choose **Group Information** and click **Bind Independent Domain Name** in the **Independent Domain Names** area.
  - b. In the displayed dialog box, enter the domain name and click **OK**.

**Table 2-5** Independent domain name configuration

| Parameter                      | Description  |
|--------------------------------|--|
| Domain Name                    | Enter the domain name to be bound.   |
| Minimum TLS Version            | The minimum TLS version that can be used to access the domain name.  |
| HTTP-to-HTTPS Auto Redirection | Whether to support HTTP-to-HTTPS redirection. Redirection takes effect only when the API request protocol is HTTPS or HTTP&HTTPS and an SSL certificate has been bound to the independent domain name.<br><b>NOTE</b><br>Redirection is only suitable for GET and HEAD requests. Redirecting other requests may cause data loss due to browser restrictions. |

- c. (Optional) For HTTPS-compatible data APIs, bind an SSL certificate to their independent domain name.
  - i. Click **Select SSL Certificate** on the right of the independent domain name.
  - ii. Select an SSL certificate and click **OK**.  
If none is available, click **Create SSL Certificate** to create one.

## Calling an API from Other Service Systems

1. Obtain the API calling information from the API provider.

- Obtain the request information of the API.  
On the ROMA Connect console, choose **API Connect > APIs**. In the API list, click the API name to go to the details page and obtain the API request method and access address. Click **Frontend Configuration** in the lower part of the page and obtain the API request parameters.
  - Obtain the authentication information of the API.  
The authentication information to be obtained varies with the API authentication mode.
    - App authentication:
      - Signature authentication: Obtain the key and secret of the credential authorized by the API provider and the SDK for calling the API.
      - Simple authentication: Obtain the AppCode of the credential authorized by the API from the API provider.
      - Others: Obtain the key and secret of the credential authorized by the API from the API provider.
    - Huawei IAM authentication: The account credential (password or AK/SK) of the cloud service platform is used for authentication. If the AK/SK is used for authentication, you also need to obtain the SDK from the API provider.
    - Custom authentication: Obtain the custom authentication information to be carried in the request parameters from the API provider.
    - None: No authentication information is required.
2. Assemble and send API requests based on the obtained API calling information to obtain required data.

## 2.3 Opening Data Through Function APIs

### Prerequisites

- The network where the service system is located can communicate with the network of ROMA Connect.  
If the ROMA Connect instances communicate with each other through the public network, an elastic IP address (EIP) must be bound to each instance.
- The data source type of the service system database is supported by ROMA Connect.  
For details about the data sources supported by function APIs, see [Data Sources Supported by Service Integration](#).
- Prepare an available independent domain name as the access domain name of the API.  
You have configured domain name resolution from the independent domain name to the APIC connection address. For details, see [Adding an A Record Set](#).

If you do not have an independent domain name, you can apply for a domain name from the domain name registrar.

- If the function API uses the HTTPS request protocol, you need to apply for an SSL certificate for the independent domain name and obtain the SSL certificate content and key in PEM format.

## Exposing the Function API of the Service System

1. Create an integration application.

All resources in the ROMA Connect instance must belong to an integration application. Before creating other resources, make sure an integration application is available. If an integration application is available, skip this step.

- a. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.
- b. In the navigation pane on the left, choose **Integration Applications**. In the upper right corner of the page, click **Create Integration Application**.
- c. In the dialog box displayed, set **Name** and click **OK**.

2. Create a function backend.

You can customize a backend to convert the data of a service system into a backend service.

- a. In the navigation pane on the left, choose **API Connect > Custom Backends**. On the **Backends** tab, click **Create Backend**.
- b. On the **Create Backend** page, set backend parameters and click **Create**.

**Table 2-6** Backend request information

| Parameter               | Description   |
|-------------------------|---|
| Name                    | Enter a custom backend name. Using naming rules facilitates future search.  |
| Integration Application | Select the integration application to which the backend belongs.  |
| Backend Request Method  | Select the request method of the backend. The value can be <b>GET</b> , <b>POST</b> , <b>PUT</b> , or <b>DELETE</b> .   |
| Backend Request Path    | Enter the request path of the backend, for example, <b>/getUserInfo/userId</b> .<br>The request path is case sensitive. |



| Parameter                       | Description   |
|---------------------------------|---|
| Backend Security Authentication | <p>Select the security authentication mode of the backend.</p> <ul style="list-style-type: none"> <li>• <b>Signature key:</b> A signature key is used to authenticate the request sender. If a signature key is used for authentication, the same signature key must be bound to the API that calls the backend service.</li> <li>• <b>None:</b> No authentication is required for calling requests.</li> </ul>   |
| Description                     | Enter a brief description of the backend.   |
| Advanced Settings               |   |
| Version                         | Enter the backend version, for example, <b>V1.0</b> . Customized backend version numbers differentiate backend services.  |
| Input Parameters                | <p>Define request parameters of the backend service based on site requirements.</p> <p>In the <b>Input Parameters</b> area, click <b>Add Input Parameter</b> to add request parameters of the custom backend.</p> <ul style="list-style-type: none"> <li>• <b>Name:</b> name of a request parameter, which is user-defined.</li> <li>• <b>Parameter Location:</b> location of the request parameter in the backend request. The value can be <b>Headers</b> or <b>Parameters</b>.</li> <li>• <b>Default Value:</b> It is used as the default test value of parameters only in the subsequent custom backend test procedure. This parameter does not take effect during custom backend deployment.</li> <li>• <b>Mandatory:</b> specifies whether a request parameter is mandatory in a backend request.</li> <li>• <b>Description:</b> Enter the description of the parameter.</li> </ul> |
| Returned Type                   | Select the response data format of the backend. The value can be <b>JSON</b> , <b>XML</b> , or <b>STREAM</b> .  |
| Formatting                      | This parameter specifies whether to format the response message body based on the selected returned type.   |

After the backend is created, the online IDE of the backend is automatically displayed. The backend type is data backend by default.

- c. In the upper left corner of the online IDE, choose **File > New Function Backend > MySQL Calling Example**. In the dialog box displayed, click **Yes** to switch the backend to a function backend.

If the Redis or MongoDB data source is used, select **Redis Calling Example**.

- d. In the statement editing box on the right, modify the data source access configuration based on the function script example provided by the system.

```
importClass(com.roma.apic.livedata.client.v1.DataSourceClient);
importClass(com.roma.apic.livedata.config.v1.DataSourceConfig);
function execute(data){
var config = new DataSourceConfig()
config.setType("mysql")
config.setUrl("jdbc:mysql://127.0.0.1:3306/db?allowPublicKeyRetrieval=true")
config.setUser("xxx")
config.setPassword("xxx")
var ds = new DataSourceClient(config)
return ds.execute("SELECT * FROM person where name = ? and age = ?","Tom",20);
}
```

- Configure the data source connection information in **setType**, **setUrl**, **setUser**, and **setPassword**. For details about the parameter configuration, see [DataSourceConfig](#).
  - Set the SQL statement in **ds.execute** to obtain the data to be returned.
- e. Click **Save** in the upper right corner of the page to save the backend configuration.
  - f. Test the backend functions.

In the upper right corner of the page, click **Test**. In the **Test Parameters** area, add request parameters based on the definition of the backend service and click **Test** to send the request. On the **Execution Result** tab page on the right, check whether the expected data is returned.

- g. After the backend is tested, click **Deploy** in the upper right corner of the page. In the dialog box displayed, click **Yes** to deploy the backend.
3. Publish the function API.

Publish the backend as a function API and deploy it in the environment so that other service systems can call it.

- a. Click **Publish** in the upper right corner of the page.
- b. On the page displayed, configure API information and click **Publish** to create a frontend API for the backend and publish the API in an environment.

After the API is published, the API details page is displayed.

**Table 2-7** Parameters for publishing an API

| Parameter   | Description   |
|-------------|---|
| Group       | Select an API group for the frontend API. If no API group is available, click <b>Create API Group</b> on the right to create one.       |
| Environment | Select the environment in which the frontend API will be published. Select the default environment <b>RELEASE</b> when exposing an API. |

| Parameter                        | Description  |
|----------------------------------|--|
| Frontend Security Authentication | <p>Select the authentication mode of the API. The <b>App</b> authentication mode is recommended.</p> <ul style="list-style-type: none"> <li>● <b>App</b>: ROMA Connect authenticates API requests. When calling an API, a user gets authenticated using the key and secret of an integration application.</li> <li>● <b>IAM</b>: IAM authenticates API requests. When calling an API, a user gets authenticated using a token or an AK/SK pair.</li> <li>● <b>Custom</b>: The custom function API is used for authenticating API requests.</li> <li>● <b>None</b>: Authentication is not required for API requests.</li> </ul> |
| Custom Authorizer                | <p>Mandatory for <b>Frontend Security Authentication</b> set to <b>Custom</b>.</p> <p>Select a frontend custom authorizer you have created.</p>  |
| Frontend Request Protocol        | <p>Select the request protocol used by the frontend API. The value can be <b>HTTPS</b>, <b>HTTP</b>, or <b>HTTP&amp;HTTPS</b>. <b>HTTPS</b> is recommended for transmitting important or sensitive data.</p>   |
| Timeout (ms)                     | <p>Enter the timeout interval of a backend service request. The default value is <b>60000</b>.</p>   |
| Retries                          | <p>Number of retries after ROMA Connect fails to call the backend service.</p> <ul style="list-style-type: none"> <li>● If the value is <b>-1</b>, this parameter is disabled.</li> <li>● If the value ranges from <b>0</b> to <b>10</b>, this parameter is enabled, and retries are performed based on the configured value.</li> </ul>   |
| Advanced Settings                |  |
| Frontend Request Method          | <p>Select the request method of the frontend API. <b>ANY</b> indicates that the API supports any request method. Generally, the API used to obtain database data uses the <b>GET</b> method.</p>   |
| Frontend Request Path            | <p>Enter the request path of the frontend API, for example, <b>/getUserInfo/userId</b>.<br/>The request path is case sensitive.</p>  |

| Parameter | Description   |
|-----------|---|
| CORS      | Whether to enable CORS for the API.<br>For security purposes, a browser restricts cross-domain requests initiated from scripts. That is, only resources from the same domain can be requested. CORS allows the browser to send XMLHttpRequest requests to the server in a different domain. For details about how to call APIs for cross-domain access, see <a href="#">Configuring CORS for APIs</a> . |

4. Bind an independent domain name to the function API.
  - a. On the API details page displayed after you publish the API, choose **Group Information** and click **Bind Independent Domain Name** in the **Independent Domain Names** area.
  - b. In the displayed dialog box, enter the domain name and click **OK**.

**Table 2-8** Independent domain name configuration

| Parameter                      | Description  |
|--------------------------------|--|
| Domain Name                    | Enter the domain name to be bound.   |
| Minimum TLS Version            | The minimum TLS version that can be used to access the domain name.  |
| HTTP-to-HTTPS Auto Redirection | Whether to support HTTP-to-HTTPS redirection. Redirection takes effect only when the API request protocol is HTTPS or HTTP&HTTPS and an SSL certificate has been bound to the independent domain name.<br><br><b>NOTE</b><br>Redirection is only suitable for GET and HEAD requests. Redirecting other requests may cause data loss due to browser restrictions. |

- c. (Optional) For HTTPS-compatible function APIs, bind an SSL certificate to their independent domain name. Otherwise, skip this step.
  - i. Click **Select SSL Certificate** on the right of the independent domain name.
  - ii. Select an SSL certificate and click **OK**.  
If none is available, click **Create SSL Certificate** to create one.

## Calling an API from Other Service Systems

1. Obtain the API calling information from the API provider.
  - Obtain the request information of the API.

On the ROMA Connect console, choose **API Connect > APIs**. In the API list, click the API name to go to the details page and obtain the API

request method and access address. Click **Frontend Configuration** in the lower part of the page and obtain the API request parameters.

- Obtain the authentication information of the API.

The authentication information to be obtained varies with the API authentication mode.

- App authentication:
    - Signature authentication: Obtain the key and secret of the credential authorized by the API provider and the SDK for calling the API.
    - Simple authentication: Obtain the AppCode of the credential authorized by the API from the API provider.
    - Others: Obtain the key and secret of the credential authorized by the API from the API provider.
  - Huawei IAM authentication: The account credential (password or AK/SK) of the cloud service platform is used for authentication. If the AK/SK is used for authentication, you also need to obtain the SDK from the API provider.
  - Custom authentication: Obtain the custom authentication information to be carried in the request parameters from the API provider.
  - None: No authentication information is required.
2. Assemble and send API requests based on the obtained API calling information to obtain required data.

# 3 Integrating and Converting Service Data Across Systems

---

[Solution Overview](#)

[Configuring Data Integration Between Systems](#)

## 3.1 Solution Overview

### Background

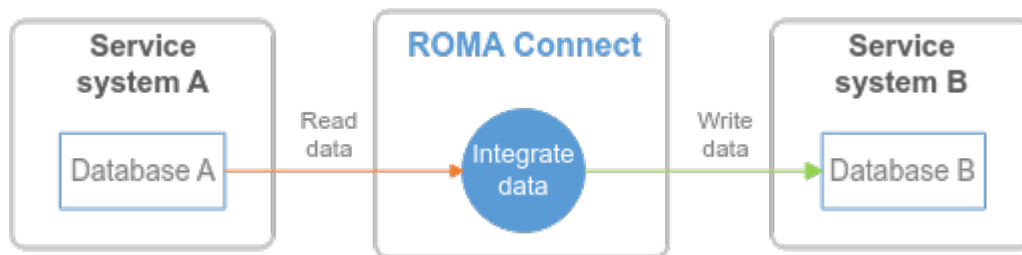
During digital transformation, service systems of some large enterprises have diversified data formats, making it difficult to transfer information effectively between service systems. Enterprises have to address pressing issues such as how to flexibly and quickly integrate and convert data of multiple data types, formats, and locations.

When a service system needs to obtain some data from another service system, the obtained data content does not meet the requirements and cannot be directly used with its own data. This makes data sharing difficult among service systems, leading to low efficiency.

### Solution

Fast Data Integration (FDI) is a data integration component of ROMA Connect. It supports flexible, fast, and non-intrusive data integration and conversion between multiple data types to establish effective data connections between service systems.

This section describes how to use ROMA Connect to convert the format of data in service system A and integrate the data into the database of service system B. The new data in service system A will be periodically synchronized to service system B as planned, ensuring data consistency between the two systems.



The FDI component directly reads data from the database of service system A, filters out specified data based on the integration task configuration, converts the data, and writes the converted data to the database of service system B.

## 3.2 Configuring Data Integration Between Systems

### Prerequisites

- The network where the service system is located can communicate with the network of ROMA Connect.  
If the ROMA Connect instances communicate with each other through the public network, an elastic IP address (EIP) must be bound to each instance.
- The data source types of source and destination service system databases are supported by ROMA Connect.  
For details about the data sources used for integration, see [Data Sources Supported by ROMA Connect](#).
- ROMA Connect has the permission to write data to the destination database.

### Configuring a Data Integration Task

1. Create an integration application.  
All resources in the ROMA Connect instance must belong to an integration application. Before creating other resources, make sure an integration application is available. If an integration application is available, skip this step.
  - a. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.
  - b. In the navigation pane on the left, choose **Integration Applications**. In the upper right corner of the page, click **Create Integration Application**.
  - c. In the dialog box displayed, set **Name** and click **OK**.
2. Connect to a data source.  
Configure the connection between ROMA Connect and service system databases to read and write data from the databases.  
The access configuration varies depending on the data source type. The following procedure uses Kafka as the source database and MySQL as the destination database. For details about other databases, see [Connecting to Data Sources](#).  
**Connecting to the Kafka data source at the source:**
  - a. In the navigation pane on the left, choose **Data Sources**. In the upper right corner of the page, click **Access Data Source**.

- b. On the **Default** tab page, select **Kafka** and click **Next**.
- c. Configure the data source connection information.

**Table 3-1** Data source connection information

| Parameter                           | Description   |
|-------------------------------------|---|
| Name                                | Enter a custom data source name. Using naming rules facilitates future search.  |
| Integration Application             | Select the integration application to which the data source belongs.  |
| Description                         | Edit the description of the data source.  |
| Connection Address                  | Enter the IP address and port number for connecting to Kafka.<br>If Kafka has multiple brokers, click <b>Add Address</b> to enter the connection addresses. |
| Enable SSL                          | Determine whether to use SSL authentication for the connection between ROMA Connect and Kafka.  |
| SSL Username/<br>Application Key    | Mandatory for <b>Enable SSL</b> set to <b>Yes</b> .<br>User name used for SSL authentication.   |
| SSL Password/<br>Application Secret | Mandatory for <b>Enable SSL</b> set to <b>Yes</b> .<br>User password used for SSL authentication.   |

- d. After setting the parameters for the data source, click **Check Connectivity** to test the data source connectivity.
  - If the test result is **Data source connected successfully**, go to the next step.
  - If the test result is **Failed to connect to the data source**, check the data source status and connection parameters, and click **Recheck** until the connection is successful.
- e. Click **Create**.

**Connecting to the MySQL data source at the destination:**

- a. On the **Data Sources** page, click **Access Data Source** in the upper right corner.
- b. On the **Default** tab page, select **MySQL** and click **Next**.
- c. Configure the data source connection information.

**Table 3-2** Data source connection information

| Parameter | Description  |
|-----------|--|
| Name      | Enter a custom data source name. Using naming rules facilitates future search. |



| Parameter               | Description  |
|-------------------------|--|
| Integration Application | Select the integration application to which the data source belongs.   |
| Description             | Edit the description of the data source.   |
| Connection Mode         | Select a database connection mode. <ul style="list-style-type: none"><li>● <b>Default:</b> The system automatically concatenates data source connection character strings based on your configured data.</li><li>● <b>Professional:</b> You need to enter a data source connection string in JDBC format.</li></ul>  |
| Connection Address      | Mandatory for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the IP address and port number for connecting to the database.   |
| Database Name           | Mandatory for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the name of the database to be accessed.   |
| Encoding Format         | Available for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the encoding format used by the database.  |
| Timeout Interval(s)     | Available for <b>Connection Mode</b> set to <b>Default</b> .<br>Enter the timeout interval for connecting to the database, in seconds.   |
| Connection String       | Mandatory for <b>Connection Mode</b> set to <b>Professional</b> .<br>Enter the JDBC connection string of the MySQL database, for example, <i>jdbc:mysql://{hostname}:{port}/{dbname}</i> . <ul style="list-style-type: none"><li>● <i>{hostname}</i> indicates the connection address of the database.</li><li>● <i>{port}</i> indicates the port number for connecting to the database.</li><li>● <i>{dbname}</i> indicates the name of the database to be connected.</li></ul> |
| Username                | Enter the username for connecting to the database.   |
| Password                | Enter the password for connecting to the database.   |

- d. After setting the parameters for the data source, click **Check Connectivity** to test the data source connection.
- If the test result is **Data source connected successfully**, go to the next step.

- If the test result is **Failed to connect to the data source**, check the data source status and connection parameters, and click **Recheck** until the connection is successful.
  - e. Click **Create**.
3. Create a data integration task.

ROMA Connect uses a data integration task to read data from the source database, convert the data structure, and write the converted data to the destination database.

- a. In the navigation pane on the left, choose **Fast Data Integration > Task Management**. On the displayed page, click **Create Common Task**.
- b. On the **Create Task** page, configure basic task information.

**Table 3-3** Basic task configuration

| Parameter          | Description   |
|--------------------|---|
| Task Name          | Enter the task name as planned. Using naming rules facilitates future search.   |
| Description        | Enter a brief description of the task.  |
| Integration Mode   | Select the mode of data integration. <ul style="list-style-type: none"><li>• <b>Scheduled:</b> A data integration task is executed based on the schedule to integrate data on the source to the destination.</li><li>• <b>Real-Time:</b> The data integration task continuously detects updates to the data at the source and integrates updates to the destination in real time.</li></ul> When Kafka is used as the source data source, only real-time tasks are supported. In this case, select <b>Real-Time</b> . |
| Tag                | Add a tag to classify tasks for quick search.   |
| Enterprise Project | Select the enterprise project to which the task belongs. In this example, retain the default value <b>default</b> .   |

- c. Configure source information.

**Table 3-4** Source information

| Parameter | Description  |
|-----------|--|
| Instance  | Select the ROMA Connect instance that is being used. |

| Parameter               | Description   |
|-------------------------|---|
| Integration Application | Integration application to which the Kafka data source at the source belongs. The integration application has been configured on the <a href="#">Access Data Source</a> page.   |
| Data Source Type        | Select <b>Kafka</b> .   |
| Data Source Name        | Select the Kafka data source that has been configured on the <a href="#">Access Data Source</a> page.   |
| Topic Name              | Select the name of the topic whose data is to be obtained.  |
| Parse                   | <p>This parameter specifies whether ROMA Connect parses the obtained source data.</p> <ul style="list-style-type: none"><li>• If you select <b>Yes</b>, ROMA Connect parses the obtained source data based on the configured parsing rules and then integrates the data to the destination.</li><li>• If you select <b>No</b>, ROMA Connect transparently transmits the obtained source data and integrates the data to the destination.</li></ul> <p>In this practice, you need to convert the data structure of the source database and then write the data to the destination database. Select <b>Yes</b>.</p> |
| Data Root Field         | This parameter specifies the path of the upper-layer common fields among all metadata in the data obtained from the source in JSON or XML format. This parameter is not required in this practice.  |
| Data Type               | Select the format of data obtained from the Kafka data source. The data format must be the same as the actual data format stored in Kafka.  |
| Offset                  | Select whether to integrate the earliest message data or the latest message data.   |

| Parameter | Description   |
|-----------|---|
| Metadata  | <p>This parameter specifies each underlying key-value data element that is obtained from the source in JSON or XML format and needs to be integrated to the destination.</p> <ul style="list-style-type: none"> <li>• <b>Alias:</b> user-defined metadata name.</li> <li>• <b>Type:</b> data type of metadata. The value must be the same as the data type of the corresponding parameter in the source data.</li> <li>• <b>Parsing Path:</b> Set this parameter to the complete path of the metadata because the data root field is not set.</li> </ul> <p>For example, in the JSON data <code>{"a": {"b": "xx", "c": "xx"}}</code>, parameters <b>b</b> and <b>c</b> are bottom-layer data elements, and their parsing paths are <b>a.b</b> and <b>a.c</b>, respectively.</p> |
| Time Zone | <p>Select the time zone used by the Kafka data source so that ROMA Connect can identify the data timestamp. The default time zone is <b>GMT+8:00</b>.</p>   |

d. Configure destination information.

**Table 3-5** Destination information

| Parameter               | Description   |
|-------------------------|---|
| Instance                | <p>Set this parameter to the ROMA Connect instance that is being used. After the source instance is configured, the destination instance is automatically associated and does not need to be configured.</p>  |
| Integration Application | <p>Integration application to which the MySQL data source at the destination belongs. The integration application has been configured on the <a href="#">Access Data Source</a> page.</p>                     |
| Data Source Type        | <p>Select <b>MySQL</b>.</p>   |
| Data Source Name        | <p>Select the MySQL data source that has been configured on the <a href="#">Access Data Source</a> page.</p>  |
| Table                   | <p>Select the data table to be written to the MySQL database.</p> <p>After selecting a data table, click <b>Select Table Field</b> and select the column fields in which you want the data to be written.</p> |

| Parameter                  | Description   |
|----------------------------|---|
| Batch Number Field         | Select a field whose type is <b>String</b> and length is greater than 14 characters in the destination table. In addition, the batch number field cannot be the same as the destination field in mapping information.<br><br>The value of this field is a random number, which is used to identify the data in the same batch. The data inserted in the same batch uses the same batch number, indicating that the data is inserted in the same batch and can be used for location or rollback. |
| Update Changed Fields Only | If this option is enabled, only the table fields whose values change are updated. If this option is disabled, all table fields are updated.   |

- e. Configure the data mapping rule from the source to the destination. Click **Automatic Mapping**. The mapping rules between the source and destination data fields are automatically created. If the fields in the data tables at the two ends are inconsistent, you need to select the corresponding source fields for the destination fields.
- f. Click **Create**.

## Starting a Data Integration Task

After a data integration task is created, **Task Status** is displayed as **Stopped** by default. You need to manually start the task by clicking **Start**.

- After a real-time task is started, ROMA Connect continuously detects data changes at the source. During the first execution, all source data that meets the conditions is integrated to the destination. Subsequently, only new data will be integrated to the destination each time.
  - After a scheduled task is started, ROMA Connect integrates data on a scheduled basis. During the first execution, all source data that meets the conditions is integrated to the destination. Then, full data that meets the conditions or only incremental data will be integrated based on the task configuration.
1. Start the data integration task.  
Select the task to be started and click **Start** above the task list. After the task is started, **Task Status** changes to **Started**.
  2. After the task is started, the task status is **Executing**. If the task status is successful, the data integration task is complete.
  3. After the execution is complete, you can view the integrated and synchronized data in the data table of the destination database.

# 4 Building an Enterprise Service Open Platform

---

[Solution Overview](#)

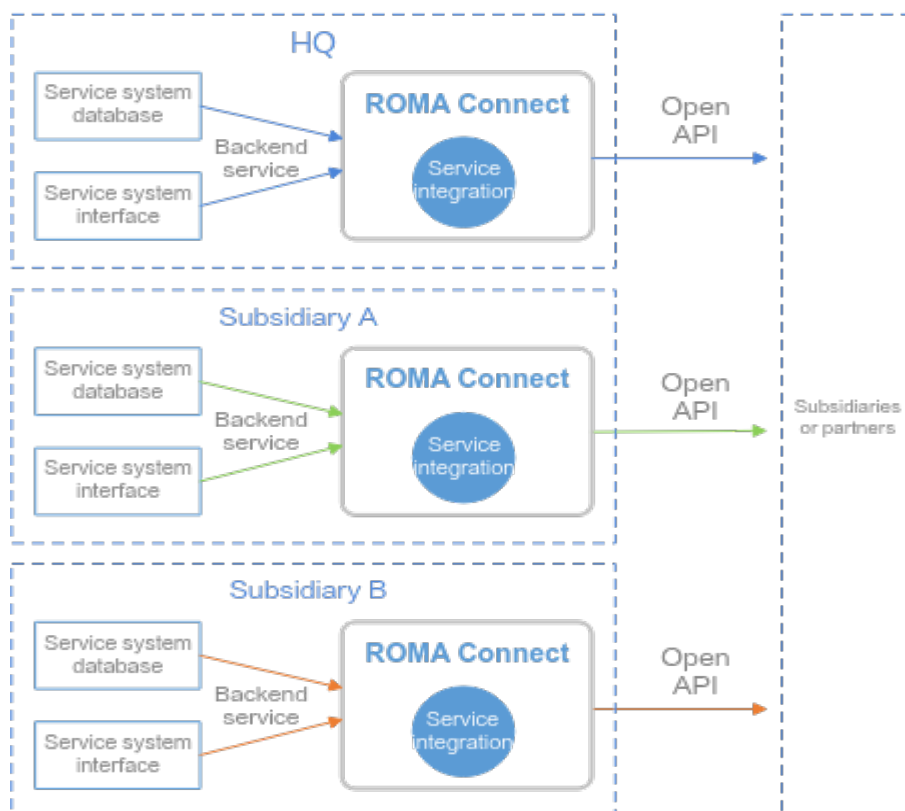
[Configuring the Service Opening Platform](#)

## 4.1 Solution Overview

### Solution Background

In some large enterprises that use ROMA Connect to digitalize their service systems, different subsidiaries or departments have their own ROMA Connect instances based on independent maintenance requirements. The service systems of each subsidiary or department are connected to their own ROMA Connect instances and share digital assets through APIs.

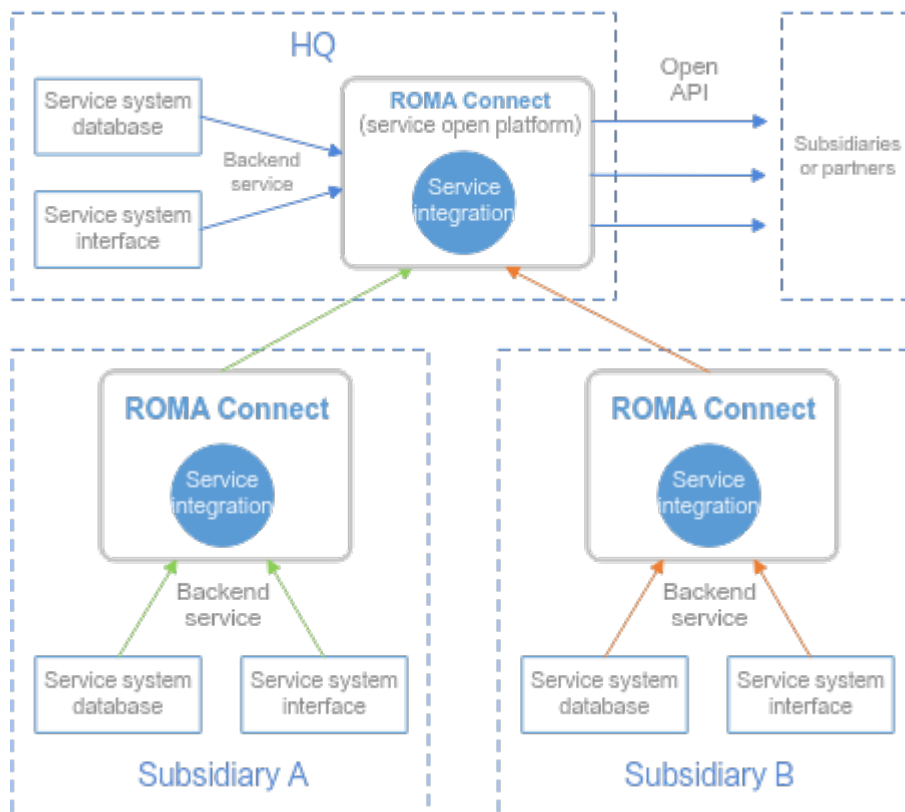
- Because subsidiaries or departments are independent from each other, ROMA Connect instances may belong to different VPCs, and each ROMA Connect instance has its own access address. To access the open APIs of other subsidiaries or departments through the intranet, a subsidiary or department needs to connect to multiple VPCs and access multiple addresses. The operation configuration is complex.
- Different subsidiaries or departments define different API authentication modes and formats when opening APIs. When a subsidiary or department needs to access the open APIs of other subsidiaries or departments, different authentication information is required, which increases the difficulty in sharing digital assets between subsidiaries or departments.



## Solutions

APIC provides the API cascading capability to cascade the APIs to be opened by all subsidiaries or departments of an enterprise to a ROMA Connect instance. The ROMA Connect instance functions as the service openness platform of the enterprise. All subsidiaries or departments call the APIs of the service openness platform to obtain shared digital assets, implementing unified call entrance and unified authentication information, improving the sharing efficiency of digital assets, and reducing the sharing difficulty.

This section describes how to use the API cascading capability integrated by the service to cascade the APIs of the ROMA Connect instances of subsidiaries A and B to the ROMA Connect instance of the HQ. The ROMA Connect instance of the HQ is used as the service openness platform. All digital assets are shared externally through the service openness platform.



## 4.2 Configuring the Service Opening Platform

### Prerequisites

- An instance (platform) and its cascaded instance (subsidiary) can communicate with each other.  
If the two instances are in different VPCs, create a VPC peering connection. For details, see [VPC Peering Connection](#).
- If the two instances are located in different networks and communicate with each other through an air wall, their IP address and port number must be configured on the air wall. In addition, the air wall must use the TCP protocol for secure access. A dedicated VPN or tunnel can also be used to implement cross-network interworking.

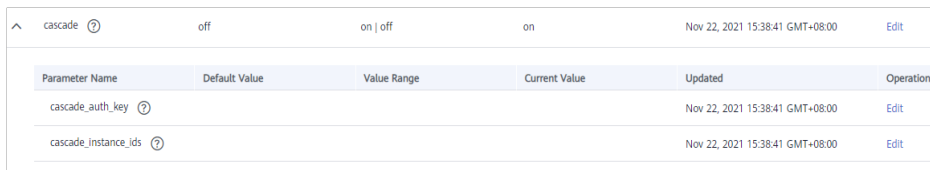
### Configuring Instance Cascading

Cascade the platform instance and its subsidiary instances so that the platform instance can open each subsidiary instance as backend service APIs.

1. Enable cascading for subsidiary instances.
  - a. Log in to the ROMA Connect console that displays subsidiary instance A. On the **Instances** page, click **View Console**.
  - b. Choose **Instance Information > Configuration Parameters** and locate the **cascade** parameter.
  - c. Click **Edit** on the right of the parameter, set **Current Value** to **on**, and click **Save**.




- d. Click  to set more cascading parameters.

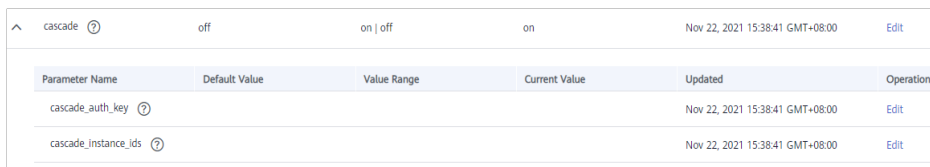


| Parameter Name       | Default Value | Value Range | Current Value | Updated                         | Operation |
|----------------------|---------------|-------------|---------------|---------------------------------|-----------|
| cascade_auth_key     |               |             |               | Nov 22, 2021 15:38:41 GMT+08:00 | Edit      |
| cascade_instance_ids |               |             |               | Nov 22, 2021 15:38:41 GMT+08:00 | Edit      |

**Table 4-1** Subsidiary instance configuration

| Parameter            | Description   |
|----------------------|---|
| cascade_auth_key     | Encryption key used for authentication between APIs in the cascading relationship. The cascade_auth_key values of the subsidiary instance and platform instance must be the same. |
| cascade_instance_ids | Enter the platform instance ID.   |

- e. Repeat **1.a** to **1.d** to enable cascading for subsidiary instance B.
2. Enable cascading for the platform instance.
- Log in to the ROMA Connect console that displays the platform instance. On the **Instances** page, click **View Console**.
  - Choose **Instance Information** > **Configuration Parameters** and locate the **cascade** parameter.
  - Click **Edit** on the right of the parameter, set **Current Value** to **on**, and click **Save**.
  - Click  to set more cascading parameters.



| Parameter Name       | Default Value | Value Range | Current Value | Updated                         | Operation |
|----------------------|---------------|-------------|---------------|---------------------------------|-----------|
| cascade_auth_key     |               |             |               | Nov 22, 2021 15:38:41 GMT+08:00 | Edit      |
| cascade_instance_ids |               |             |               | Nov 22, 2021 15:38:41 GMT+08:00 | Edit      |

**Table 4-2** Platform instance configuration

| Parameter            | Description   |
|----------------------|---|
| cascade_auth_key     | Encryption key used for authentication between cascading APIs. The subsidiary and platform instances must use the same <b>cascade_auth_key</b> value. |
| cascade_instance_ids | You do not need to set this parameter in the platform instance.   |

## Configuring a Load Balance Channel Between Instances

Configure the load balance channel from the platform instance to each subsidiary instance. When the platform instance calls the API of the subsidiary instance as

the backend service, the dedicated authentication channel can be used to avoid authentication conflicts.

1. In the navigation pane of the platform instance console, choose **API Connect > API Policies**. On the **Load Balance Channels** tab, click **Create Load Balance Channel**.
2. On the page displayed, configure load balance channel information.

**Table 4-3** Basic information

| Parameter         | Description   |
|-------------------|---|
| Name              | Enter a load balance channel name. Using naming rules facilitates future search.  |
| Port              | Enter the access port number in the load balance channel. You can determine the port number based on the protocol used by the API in the subsidiary instance. For HTTP, set this parameter to <b>80</b> . For HTTPS, set this parameter to <b>443</b> . |
| Routing Algorithm | Select an algorithm for routing backend service requests. The load balance channel determines which servers the algorithm will send the requests to.  |

3. Configure servers.
  - a. Specify the way to add servers. Set **Mode** to **Specify IP addresses** if you want to access subsidiary instance APIs.
  - b. Click **Create Server Group**. In the dialog box displayed, configure group information and click **OK**.  
Servers can be added to different groups.

**Table 4-4** Server group configuration

| Parameter   | Description  |
|-------------|--|
| Group Name  | Enter a server group name. Using naming rules facilitates future search.   |
| Weight      | Enter the weight of the server group. The larger the weight, the more requests can be forwarded to the servers in the group. |
| Description | Enter a brief description of the server group.   |

- c. Click **Add Backend Server Address**.
- d. Configure backend server information.

**Table 4-5** Backend server information

| Parameter              | Description   |
|------------------------|---|
| Backend Server Address | Enter the API access address of the subsidiary instance. <ul style="list-style-type: none"><li>• IP address format:<ul style="list-style-type: none"><li>– Set this parameter to the EIP of the subsidiary instance if the two instances communicate with each other over a public network.</li><li>– Set this parameter to the APIC connection address of the subsidiary instance if the two instances communicate with each other over a VPC intranet.</li></ul></li><li>• Domain name format: Enter the access domain name of the API.</li></ul> |
| Standby Node           | Enabled: The backend server serves as a standby node. It works only when all non-standby nodes are faulty.  |
| Port                   | Access port number of the backend server. If the port number is 0, the port of the load balance channel is used.  |
| Server Status          | Specify whether to enable the server. Requests are distributed to the server only after it is enabled.  |

## 4. Configure the health check.

The health check function is enabled by default. If you do not need the health check, disable this function.

**Table 4-6** Health check configurations

| Parameter              | Description  |
|------------------------|--|
| Protocol               | Select the protocol used for the health check. Options: <b>TCP, HTTP, or HTTPS</b>   |
| Two-Way Authentication | Available for <b>Protocol</b> set to <b>HTTPS</b> .<br>Specify whether to enable two-way authentication between ROMA Connect and backend servers.                            |
| Path                   | Mandatory for <b>Protocol</b> set to <b>HTTP</b> or <b>HTTPS</b> .<br>Enter the health check URL.  |
| Method                 | Mandatory for <b>Protocol</b> set to <b>HTTP</b> or <b>HTTPS</b> .<br>Select the HTTP request method used for the health check. The value can be <b>GET</b> or <b>HEAD</b> . |

| Parameter           | Description   |
|---------------------|---|
| Check Port          | Enter the destination port of the health check. By default, the port number configured for the load balance channel is used.  |
| Healthy Threshold   | Number of consecutive successful checks required for a server to be considered healthy. Example: If set to <b>2</b> , ROMA Connect declares the ECS status to be healthy when the check is successful twice in a row. |
| Unhealthy Threshold | Number of consecutive failed checks required for a server to be considered unhealthy. Example: If set to <b>5</b> , ROMA Connect declares the ECS status to be abnormal when the check fails five times in a row.     |
| Timeout Interval(s) | Response timeout of a health check, in seconds. If no response is received within this time, the health check fails.  |
| Interval (s)        | Interval between consecutive checks, in seconds.  |
| Status Codes        | Mandatory for <b>Protocol</b> set to <b>HTTP</b> or <b>HTTPS</b> .<br>If specified HTTP status codes are returned, the response is successful.  |

5. Click **Finish**.
6. Repeat **1** to **5** to configure the load balance channel from the platform instance to other subsidiary instances.

## Creating an API on the Platform Instance

Create and open APIs in the platform instance, and use the subsidiary instance APIs as the backend service. Each subsidiary or partner calls other subsidiary instances' APIs opened by the platform instance.

For details about how to create an API, see [Creating an API](#). Only the backend configuration is different, as shown as follows.

**Table 4-7** API backend service parameters

| Parameter            | Description   |
|----------------------|---|
| Backend Type         | Select a backend service type. When the API of the subsidiary instance is used as the backend service, select <b>HTTP/HTTPS</b> .                                 |
| Load Balance Channel | Whether the backend service is accessed through a load balance channel. Configure the channel when the subsidiary instance APIs are used as the backend services. |

| Parameter              | Description  |
|------------------------|--|
| URL                    | <p>Configure the URL of the backend service.</p> <ul style="list-style-type: none"><li>● <b>Method:</b> Select the request method of the backend service based on the API request method of the subsidiary instance.</li><li>● <b>Protocol:</b> Select the request protocol used by the backend service based on the API request protocol of the subsidiary instance.</li><li>● <b>Load Balance Channel:</b> Select the load balance channel created in <a href="#">Configuring a Load Balance Channel Between Instances</a>.</li><li>● <b>Path:</b> Enter the request path of the backend service, for example, <code>/getUserInfo/{userId}</code>. The request path can contain the path parameter.<br/>If the path needs to contain an environment variable, enclose the environment variable in number signs (#), for example, <code>/#path#</code>. Environment variable names are case sensitive. Multiple environment variables can be added, for example, <code>/#path##request#</code>.</li></ul> |
| Cascading Flag         | Determine whether to use the cascading mode to access backend services. Enable this option.  |
| Host Header            | Define the host header field carried in the backend service request. If you have specified <b>Backend Server Address</b> with an IP address when creating a load balance channel in <a href="#">Configuring a Load Balance Channel Between Instances</a> , set <b>Host Header</b> to the domain name of the API of the subsidiary instance.  |
| Timeout (ms)           | Enter the timeout interval of a backend service request. The default value is <b>5000</b> .  |
| Retries                | <p>Number of retries after ROMA Connect fails to call the backend service.</p> <ul style="list-style-type: none"><li>● If the value is <b>-1</b>, this parameter is disabled.</li><li>● If the value ranges from <b>0</b> to <b>10</b>, this parameter is enabled, and retries are performed based on the configured value. The number of retries cannot be greater than the number of backend servers enabled in the load balance channel.</li></ul>  |
| Two-Way Authentication | <p>Available for <b>Protocol</b> set to <b>HTTPS</b>.</p> <p>Specify whether to enable two-way authentication between ROMA Connect and backend services. Disable two-way authentication for subsidiary instance APIs set as the backend services.</p>  |
| Backend Authentication | Determine whether to enable backend authentication. When the API of the subsidiary instance is used as the backend service, do not enable backend authentication.  |

# 5 Developing a Custom Authorizer with a Custom Backend

---

[Solution Overview](#)

[Developing a Custom Authorizer](#)

[Using a Custom Authorizer](#)

## 5.1 Solution Overview

### Application Scenario

ROMA Connect APIC quickly encapsulates existing backend services, data sources, and user-defined functions into RESTful APIs, and then exposes the APIs to external systems, streamlining the interconnection between service systems and lowering the costs.

APIC supports App and IAM authentication for secure API access. APIC also supports custom authorizers with your own authentication mechanism to better adapt to your business capabilities.

### Solution Advantages

The custom authorizer enables your service system to inherit the existing authentication mechanism to better adapt to your business capabilities and lower development costs.

### Constraints

APIC supports frontend and backend custom authorizers. This practice takes a frontend custom authorizer as an example. For details about how to use a backend custom authorizer, see [Creating a Backend Custom Authorizer](#).

## 5.2 Developing a Custom Authorizer

### Creating a Function Backend for Frontend Authentication

Develop a custom authentication function with a custom backend for access authentication.

1. Log in to the ROMA Connect console. On the **Instances** page, click **View Console** next to a specific instance.
2. Create an integration application.
  - a. In the navigation pane on the left, choose **Integration Applications**. In the upper right corner of the page, click **Create Integration Application**.
  - b. In the dialog box displayed, set **Name** and click **OK**.
3. In the navigation pane on the left, choose **API Connect > Custom Backends**. On the **Backends** tab, click **Create Backend**.
4. On the **Create Backend** page, set backend parameters and click **Create**.

**Figure 5-1** Custom backend configuration

The screenshot shows the 'Create Backend' configuration form. It contains the following fields and options:

- Name:** A text input field containing 'Auth'. Below it is a note: 'Enter a string of 3 to 100 characters starting with a letter. Only letters, digits, hyphens (-), underscores (\_), periods (.)'.
- Integration Application:** A dropdown menu showing 'app-test' and a 'Create Integration Application' button.
- Backend Request Method:** A dropdown menu showing 'POST'.
- Backend Request Path:** A text input field containing '/auth'. Below it is a note: 'Start with a slash (/), and use letters, digits, hyphens (-), underscores (\_), and periods (.). (2 to 256 characters)'.
- Backend Security Authentication:** Two radio buttons: 'Signature key' (selected) and 'None'. Below them is a warning: 'No security authentication. Access will be granted to all users. (Not recommended)'.
- Description:** A large text area with the placeholder 'Enter a backend description.' and a character count '0/1,000' at the bottom right.

**Table 5-1** Backend configuration

| Parameter               | Description   |
|-------------------------|---|
| Name                    | Backend name.   |
| Integration Application | Select the integration application created in <a href="#">2</a> . |

| Parameter                       | Description  |
|---------------------------------|--|
| Backend Request Method          | The backend request method must be <b>POST</b> .                                 |
| Backend Request Path            | Enter the request path of the backend, for example, <i>/getUserInfo/userId</i> . |
| Backend Security Authentication | Backend authentication mode. Select <b>None</b> .                                |
| Description                     | Enter a description of the backend.  |
| Advanced Settings               | Retain the default settings in the <b>Advanced Settings</b> area.                |

After the backend is created, the online IDE of the backend is automatically displayed. The backend type is data backend by default.

5. Develop a custom authentication function.

In the upper left corner of the online IDE, choose **File > New Function Backend > Frontend Custom Authentication Example**. In the dialog box displayed, click **OK**, and then compile a function script for security authentication.

Modify the authentication parameter name and value in the example. In this practice, the authentication parameter is header **x-auth** of the API request. When the value of **x-auth** is **user1:xxxx**, the authentication is successful.

```
function execute(data) {
  data = JSON.parse(data)
  body = data.body
  if (body["headers"]["x-auth"] === 'user1:xxxx') {
    result = {
      "status": "allow",
      "context": {
        "user_name": "user1"
      }
    }
  }
  else {
    result = {
      "status": "deny",
      "context": {
        "error_code": "1001",
        "error_message": "incorrect token input"
      }
    }
  }
  return result
}
```

6. After editing the function, click **Save** in the upper right corner of the page to save the backend configuration.
7. Click **Deploy** in the upper right corner of the page. In the dialog box displayed, click **Yes** to deploy the function backend.



## Creating a Frontend Custom Authorizer

1. In the navigation pane on the left, choose **API Connect > API Policies**. On the **Custom Authorizers** tab, click **Create Custom Authorizer**.
2. On the page displayed, configure custom authorization.

**Figure 5-2** Custom authorizer configuration

Create Custom Authorizer

---

\* Name

\* Integration Application

\* Type  Frontend  Backend

\* Function URN

\* Max. Cache Age (s)

Identity Sources 

| Parameter Location                                   | Parameter Name | Operation |
|--|----------------|-----------|
| <input type="button" value="+"/> Add Identity Source |                |           |

Send Request Body

User Data

0/2,048

**Table 5-2** Parameters for creating a frontend custom authorizer

| Parameter               | Description  |
|-------------------------|--|
| Name                    | Custom authorizer name.  |
| Integration Application | Select the integration application created in <a href="#">2</a> .  |
| Type                    | Select <b>Frontend</b> .   |
| Function URN            | Select the Function backend created in <a href="#">Creating a Function Backend for Frontend Authentication</a> . |
| Max. Cache Age (s)      | Retain the default settings.   |
| Identity Sources        | Retain the default settings.   |

| Parameter         | Description                  |
|-------------------|------------------------------|
| Send Request Body | Retain the default settings. |
| User Data         | Retain the default settings. |

3. Click **OK**.

## 5.3 Using a Custom Authorizer

### Creating a Custom Authentication API

Create an API that uses a custom authorizer in ROMA Connect.

1. In the left navigation pane on the instance console, choose **API Connect > APIs**. In the upper right corner, click **Create API**.
2. On the page displayed, configure the frontend definition of the API.

**Figure 5-3** Frontend configuration

Frontend Definition

\* API Name   
Enter a string of 3 to 255 characters starting with a letter. Only letters, digits, hyphens (-), underscores (\_), periods (.), slash (/), colons (:), and parentheses (()) are allowed.

\* Integration Application  [Create Integration Application](#)

\* Group  [Create API Group](#)

\* URL

| Method                           | Protocol                           | Subdomain Name                        | Path                           |
|----------------------------------|------------------------------------|---------------------------------------|--------------------------------|
| <input type="text" value="GET"/> | <input type="text" value="HTTPS"/> | <input type="text" value="c890c..."/> | <input type="text" value="/"/> |

Matching  Exact match  Prefix match  
API requests will be forwarded to the specified path.

Tags

Description

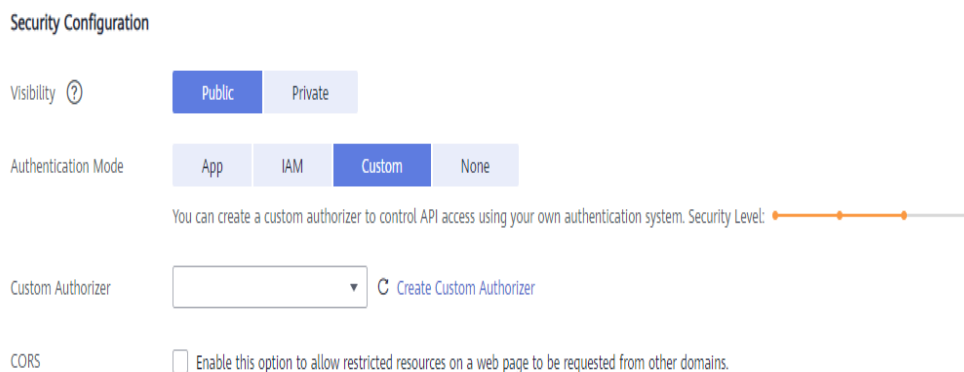
**Table 5-3** Frontend configuration

| Parameter               | Description   |
|-------------------------|---|
| API Name                | Enter an API name.  |
| Integration Application | Select the integration application created in <b>Creating a Function Backend for Frontend Authentication</b> .                    |
| Group                   | Select an API group for the frontend API. If no API group is available, click <b>Create API Group</b> on the right to create one. |

| Parameter   | Description  |
|-------------|--|
| URL         | Configure the API access address. <ul style="list-style-type: none"> <li>• <b>Method:</b> request method of the API. Set this parameter to <b>GET</b> here.</li> <li>• <b>Protocol:</b> request protocol of the API. Set this parameter to <b>HTTPS</b> here.</li> <li>• <b>Path:</b> request path of the API, in <code>/getUserInfo/{userId}</code> format. Set this parameter to <code>/test</code> here.</li> </ul> |
| Matching    | Matching mode of the API request path. Set this parameter to <b>Exact match</b> here.  |
| Tags        | Retain the default settings.   |
| Description | Retain the default settings.   |

3. Configure API security information.

**Figure 5-4** Security configuration



**Table 5-4** Security configuration

| Parameter           | Description   |
|---------------------|---|
| Visibility          | Whether the API can be released to the marketplace. Retain the default value <b>Public</b> , which indicates that the API can be released to the marketplace. |
| Authentication Mode | Select <b>Custom</b> .  |
| Custom Authorizer   | Select the custom authorizer created in <a href="#">Creating a Frontend Custom Authorizer</a> .   |
| CORS                | Retain the default setting to disable CORS.   |

4. Configure the request parameters of the API. Or skip this step and click **Next**.
5. Set the backend type. In this example, select **Mock** to return specified response.

6. Configure API backend information.

**Figure 5-5** Backend configuration

**Basic Information**

Status Code

Response

10/2,048

Mock enables the system to return a response without sending the request to the backend. This is useful for testing APIs when the backend is not available.

Backend Authentication  Use custom authorizer for authentication

---

| Constant Parameter Name | Parameter Value |
|-------------------------|-----------------|
|-------------------------|-----------------|

**Table 5-5** Backend configuration

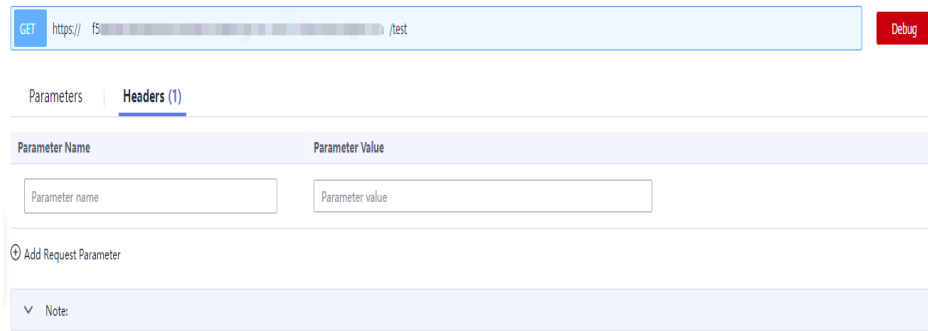
| Parameter              | Description  |
|------------------------|--|
| Status Code            | Select the HTTP status code returned by the API. Use <b>200</b> here.  |
| Response               | Enter the response result of the API. In this example, set this parameter to <b>ROMA TEST!</b> . Once you call the API, <b>ROMA TEST!</b> is returned. |
| Backend Authentication | Retain the default setting to disable backend authentication.  |
| Add Header             | Retain the default settings.   |

7. Configure the response example. Retain the default settings.
  8. Click **Finish**.
- After the API is created, the API details page is displayed.

## Debugging an API

1. On the details page of the API created in [Creating a Custom Authentication API](#), click **Debug** in the upper right corner.
2. On the page displayed, click the **Headers** tab, and click **Add Request Parameter**.
3. Set **Parameter name** to **x-auth** and **Parameter value** to **user1:xxxx**.

**Figure 5-6** API debugging



- Click **Debug** on the right of the URL. The request sent when you call the API and the response received are displayed in the lower part of the page. The message **ROMA TEST!** is displayed, indicating that the API is successfully called.

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Connection: keep-alive
Content-Type: application/json
...
ROMA TEST!
```

- Delete the request parameter **x-auth** added in 3 and click **Debug** again. The response indicates that the API fails to be called, and the error message indicates that the authentication information is incorrect.

```
HTTP/1.1 401 Unauthorized
Transfer-Encoding: chunked
Connection: keep-alive
Content-Type: application/json
...
{"error_msg":"Incorrect authentication information: frontend authorizer","error_code":"APIC.0305","request_id":"56b2c6ae4a49f44b88670201eade9e05"}
```