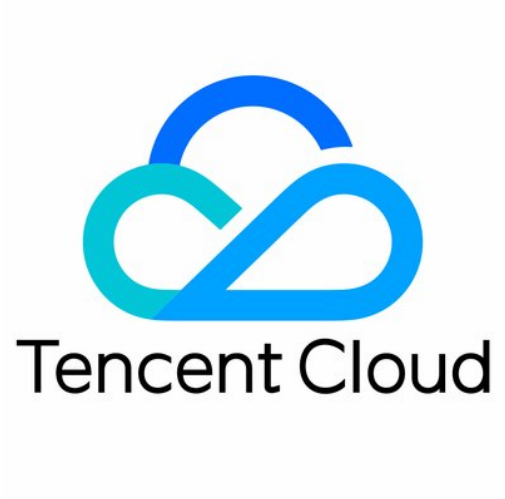
Tencent Integration Platform

User Guide

Product Documentation



Contents

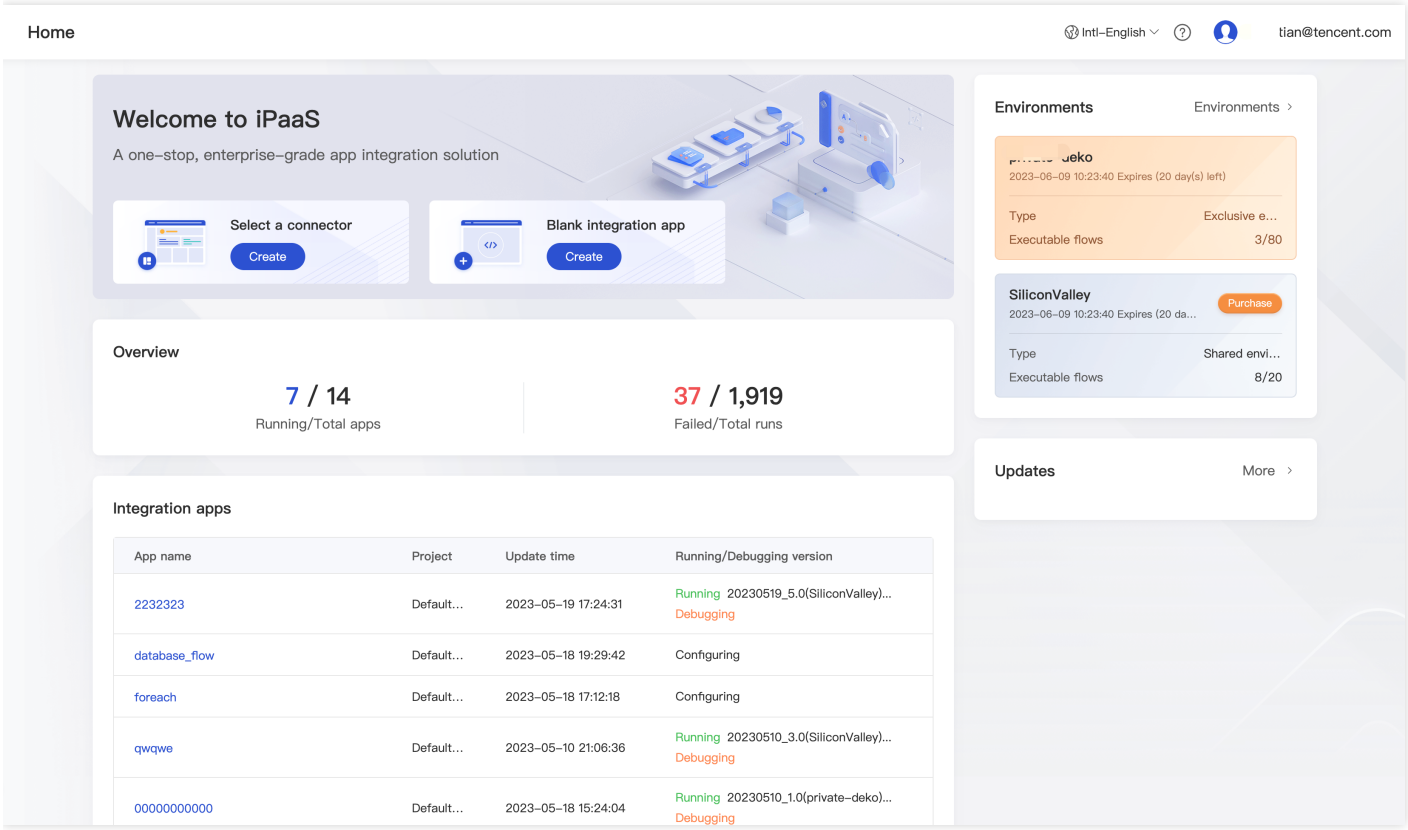
User Guide
Homepage

Last updated：2023-08-03 17:09:39

This document describes the Home page of iPaaS, which centrally displays content such as integration app creation, data overview, help center, and app creation guide.

Creating an integration app

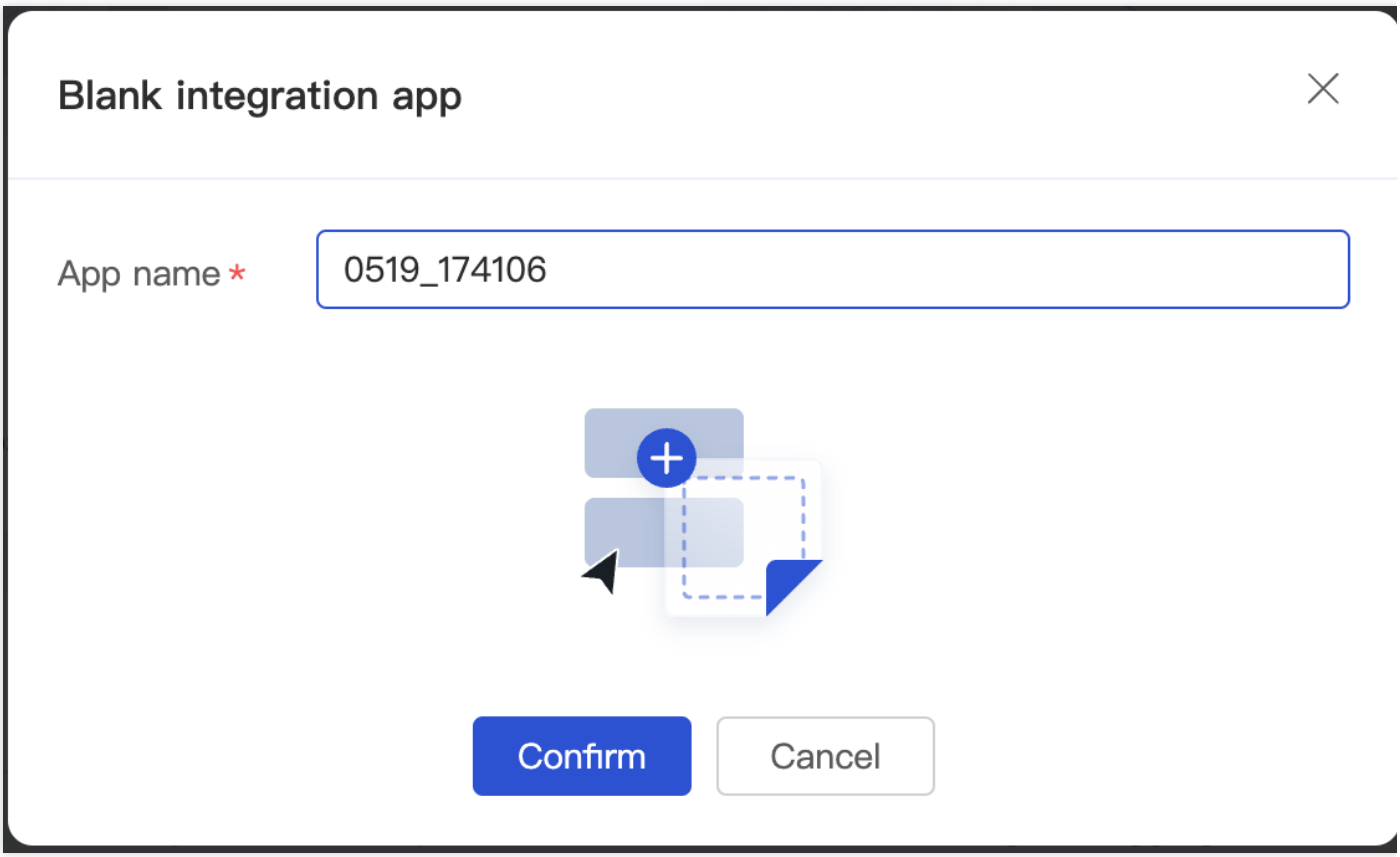
The **Home** page offers entries for quick integration app creation: **Select a connector** and **Blank application**.



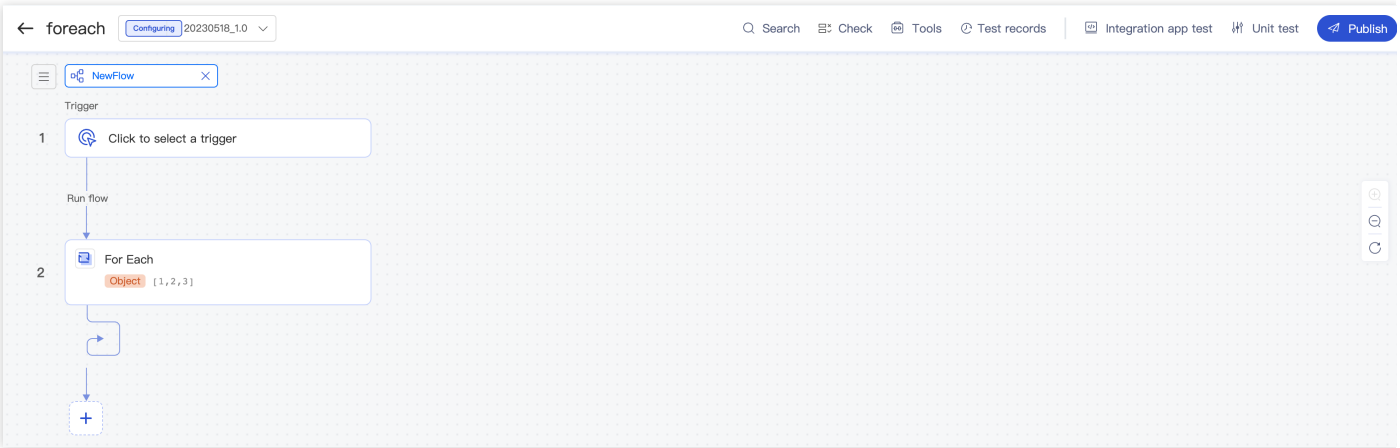
In the **Blank application** module, you can create an app from scratch and configure it as needed.

1. In the **Blank application** module, click **Create**.

2. Enter an app name and click **Done**.



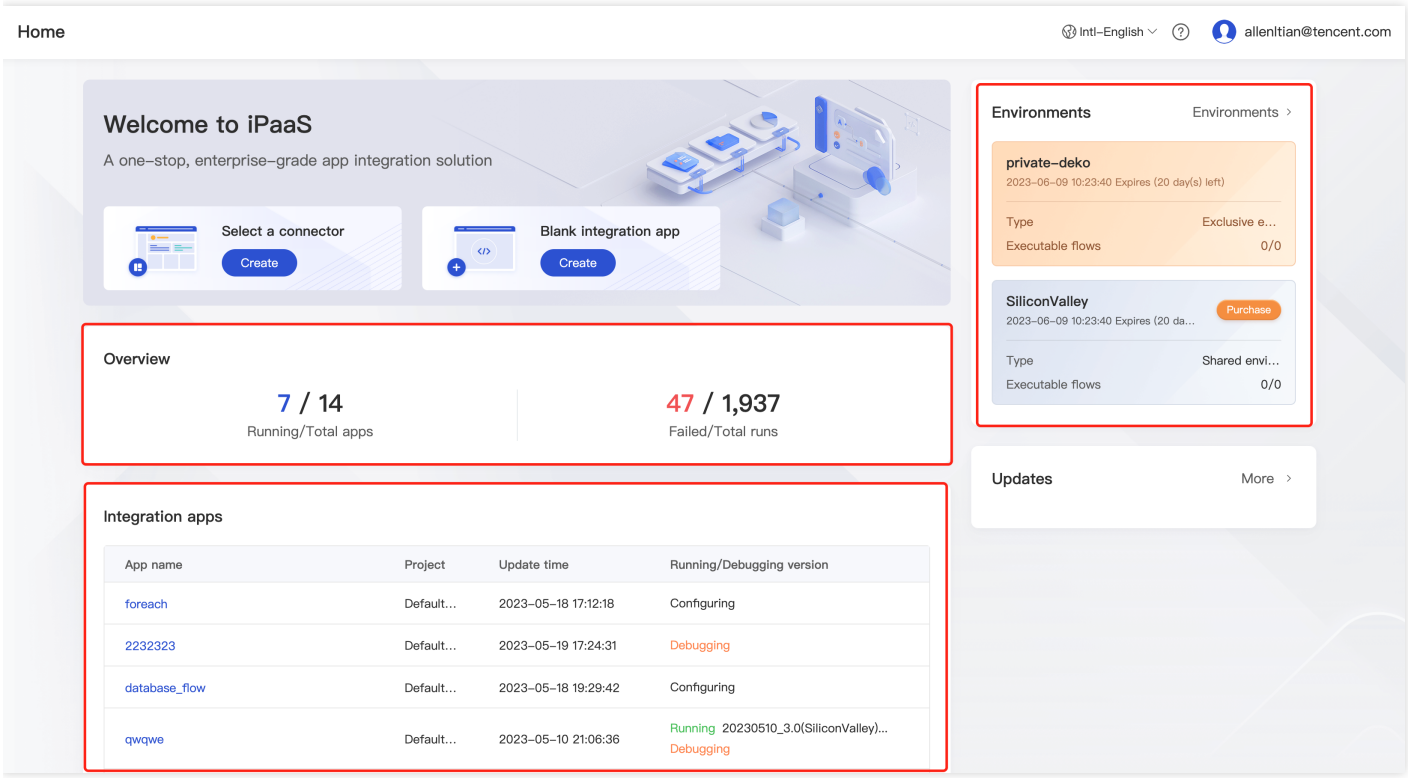
3. Go to the flow development page to configure a flow as instructed in App Configuration.



Overview

The **Data overview** module on the **Home** page displays the numbers of integration apps, running integration apps, executions, and execution failures under the current account for you to directly view the basic status of integration

apps.



App list

The **App list** module on the **Home** page displays the basic information of some apps under the current account. It displays the information of the latest five apps by update time. To query the information of all apps under the account, click **More**.

Environment management

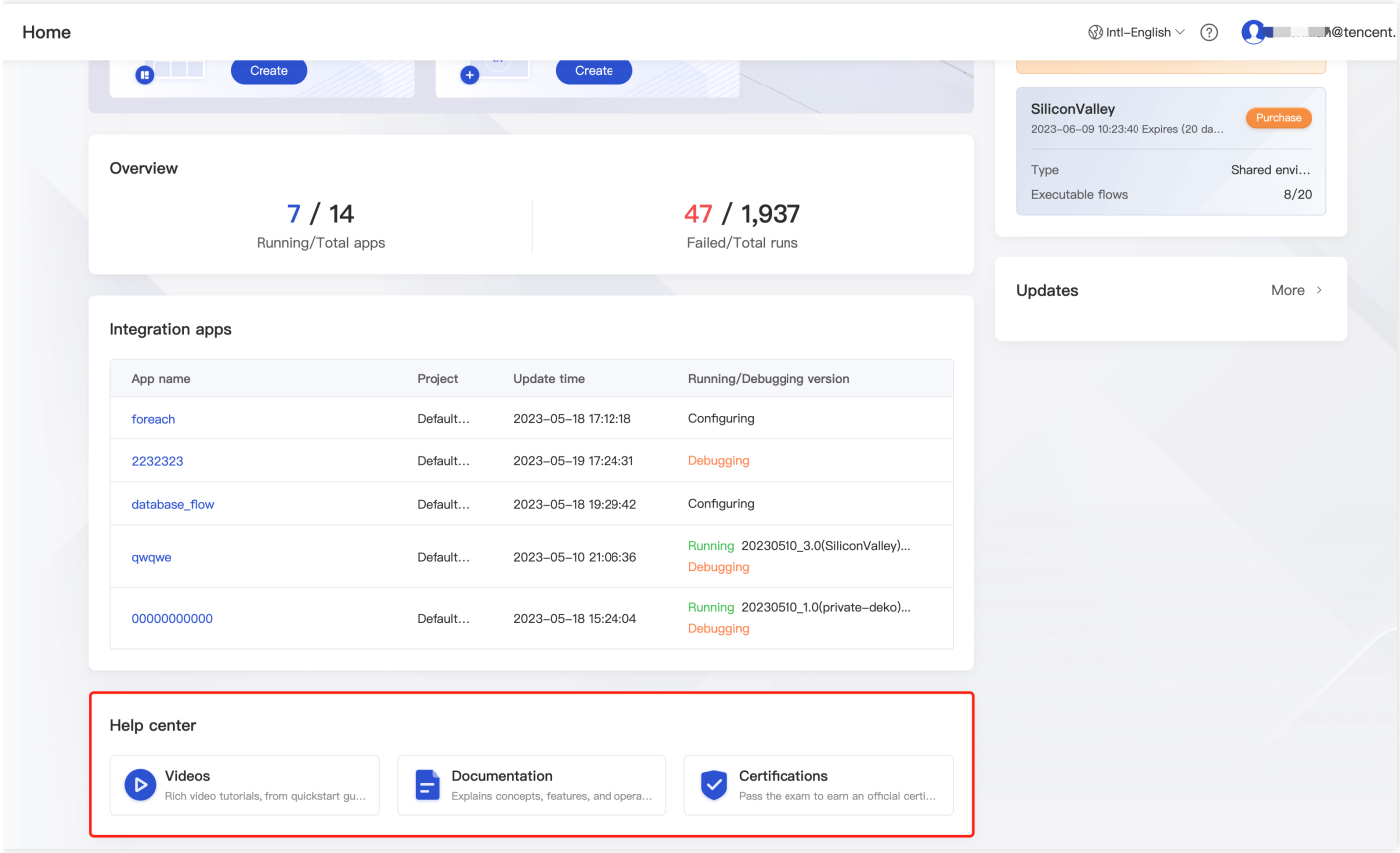
The **Environment management** module on the **Home** page displays the basic information of some environments under the current account, including the environment name and type and the number of flows running in the environment. To query the information of all environments under the account, click **Environment management**.

Updates

The **Updates** module on the **Home** page displays information such as new or updated documents or features of the product for you to stay up to date with the new features of the product. To view more historical updates.

Help center

The **Help center** module on the **Home** page provides tutorial videos and documentation for you to further understand iPaaS. If you are familiar with iPaaS usage and concepts, you can get an official certification of iPaaS after passing the exam in the certification center.



Integration Development
Integration App
App Management

Last updated：2023-08-03 17:12:12

Overview

You can create integration apps to meet your integration needs. In a complex project, you can create multiple integration apps for different integration scenarios. On the **Integration apps** page, you can create, configure, rename, export, import, and delete apps.

The options displayed in the **Operation** column vary by app status as follows:

```
<blockquote class="d-mod-notice">
<div class="d-mod-title d-notice-title">
<i class="d-icon-notice"></i>注意：
</div>
<p> The debugging feature in the standalone console manipulates a certain version
of the current integration app. You can still configure the current app after min
imizing the test window.</p>

</blockquote>
<table>
```

Status Available operations Running View, rename, copy, configure, export, and stop the app. Configuring View, rename, export, configure, publish, and delete the app. Stopped View, configure, rename, publish, and delete the app. Debugging Vew, configure, rename, and export the app, perform an app test and a unit test, and exit the debug mode.

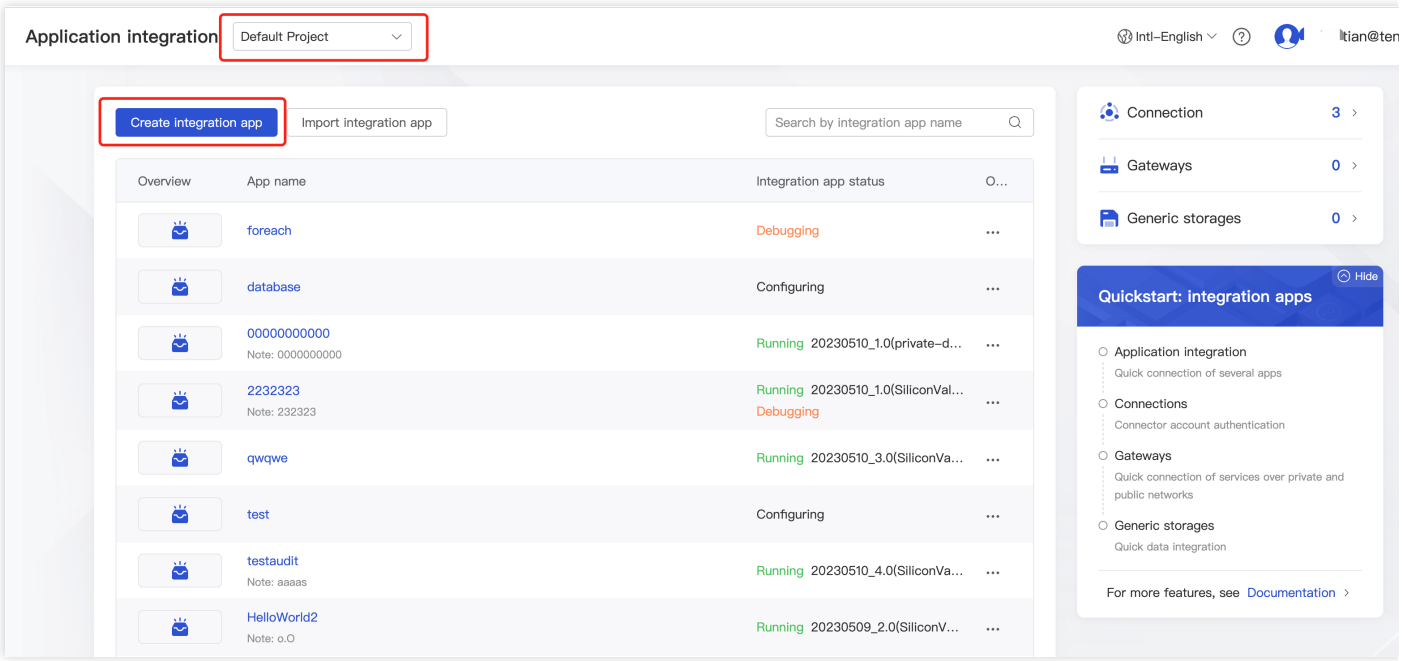
Directions

Creating an integration app

Create an integration app as follows:

1. Log in to the iPaaS console and click **Integration apps** on the left sidebar.

2. On the **Integration apps** page, select the target project name and click **Create integration app**.

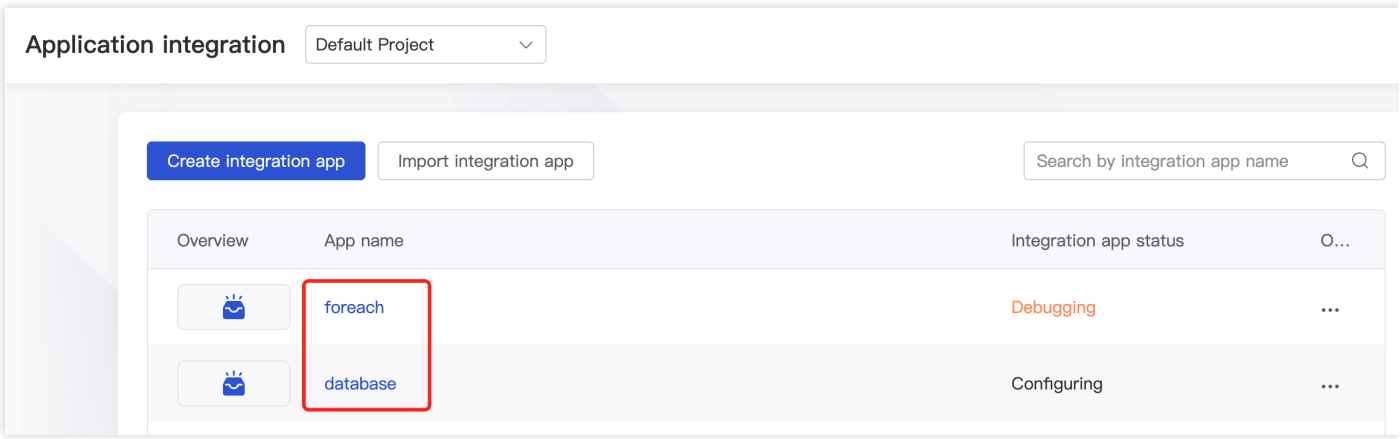


3. In the pop-up windown, enter the app name, select the creation method, and click **Confirm** to enter the app configuration page.

Configuring an integration app

View the details or modify the configuration of an app as follows:

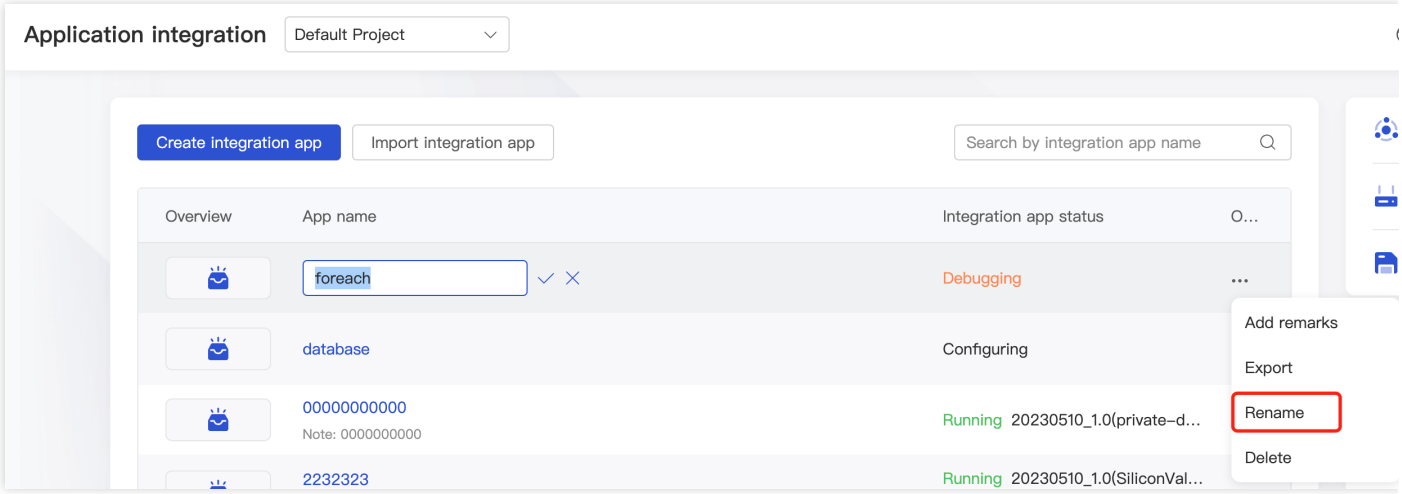
1. Log in to the iPaaS console and click **Integration apps** on the left sidebar.
2. On the **Integration apps** page, find the target app and click the **App name** to enter the app details page for configuration.



Renaming an integration app

Rename a modified app as follows for easier identification:

1. Log in to the iPaaS console and click **Integration apps** on the left sidebar.
2. On the **Integration apps** page, find the target app, click **Rename** in the **Operation** column, and enter a new name as prompted.

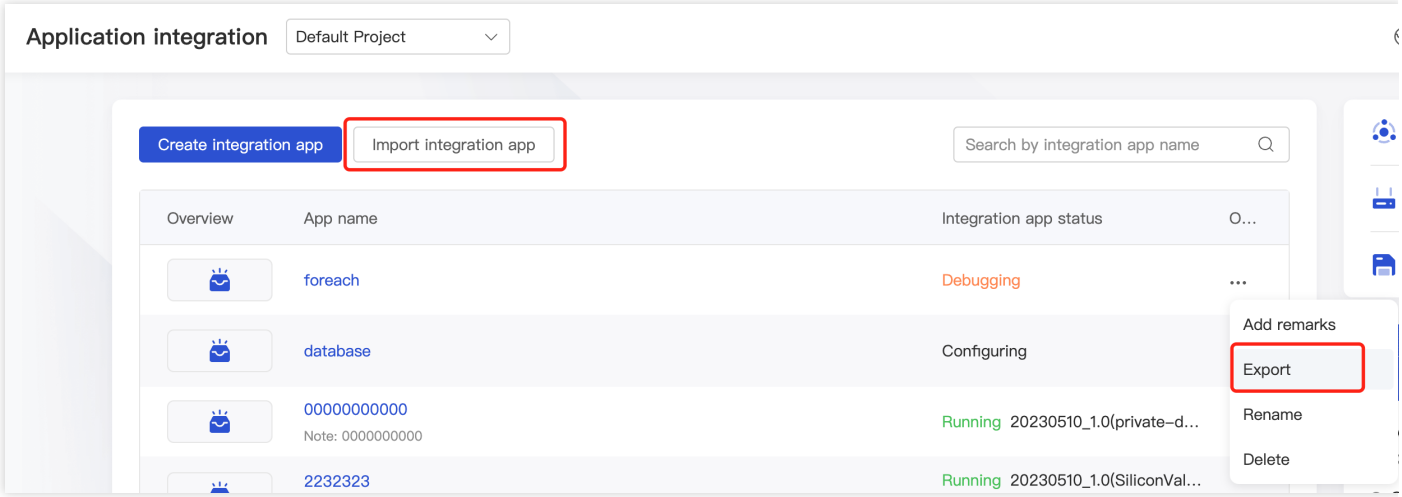


Importing/Exporting an integration app

To share an integration app with another account or change the project of an app (currently, you cannot directly perform such operations), you can use the import and export features.

1. Log in to the iPaaS console and click **Integration apps** on the left sidebar.
2. On the **Integration apps** page, click **Import integration app** in the top-left corner to import an app, or find the target app and click **Export** in the **Operation** column to export the app.

- When exporting an app, you can select the target version and choose to export all or the specified flows and connections. The exported app will be automatically downloaded as an .ipaas file.
- To import an app, you only need to upload its .ipaas file.



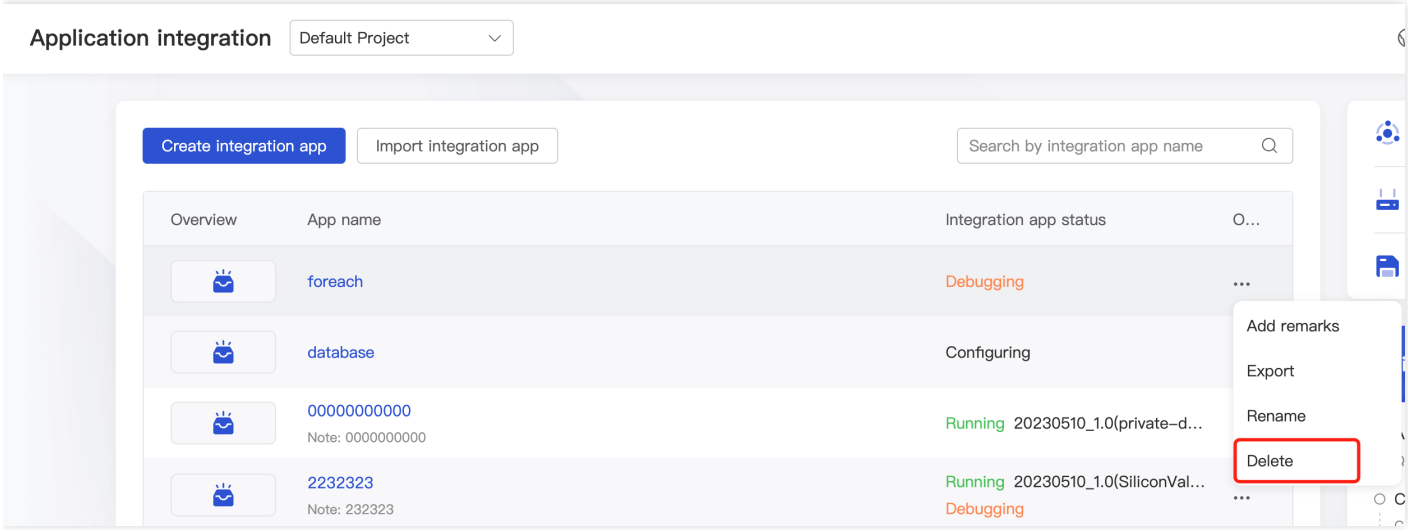
Deleting an integration app

An integration app that is not running can be deleted.

```
<blockquote class="d-mod-notice">
<div class="d-mod-title d-notice-title">
<i class="d-icon-notice"></i>注意：
</div>
<p>The data of a deleted app cannot be recovered. Therefore, export the data for backup or check whether the data is needed before deleting an app.</p>

</blockquote>
<ol>
```

- Log in to the iPaaS console and click **Integration apps** on the left sidebar.
- On the **Integration apps** page, find the target app and click **Delete** in the **Operation** column.

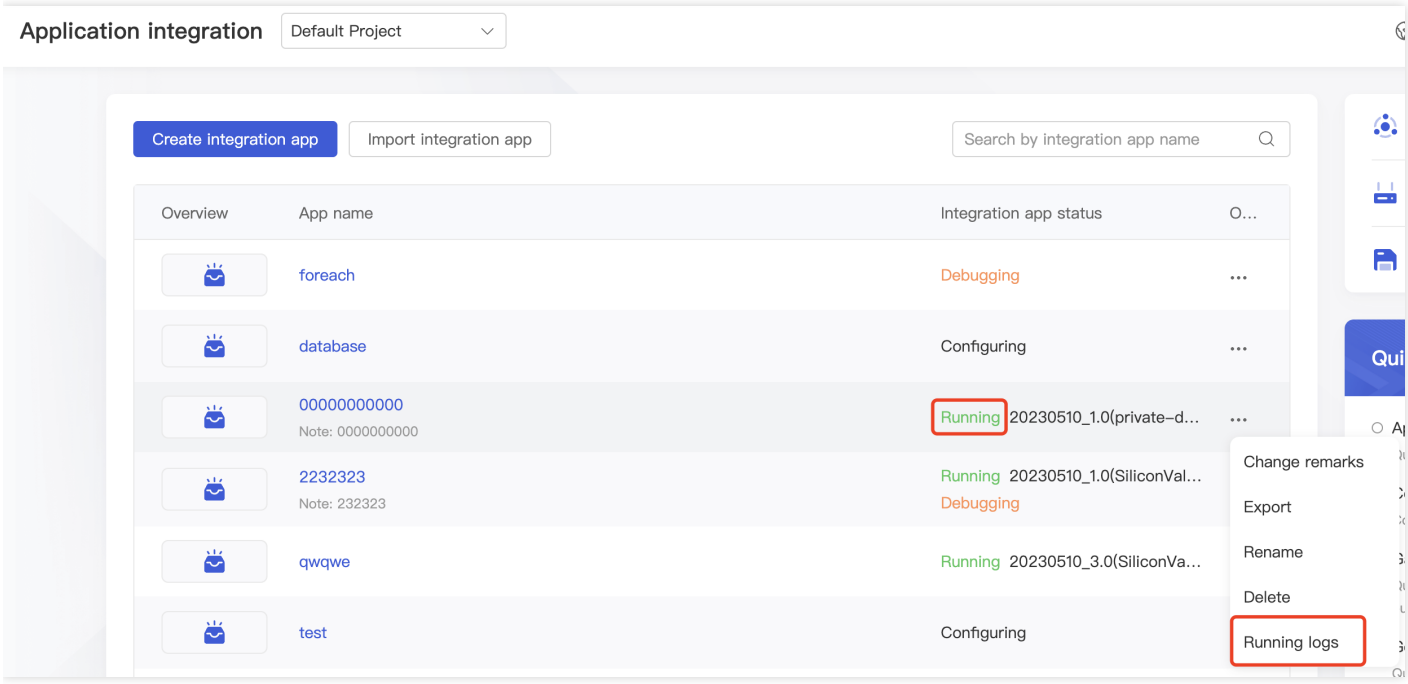


Viewing running logs

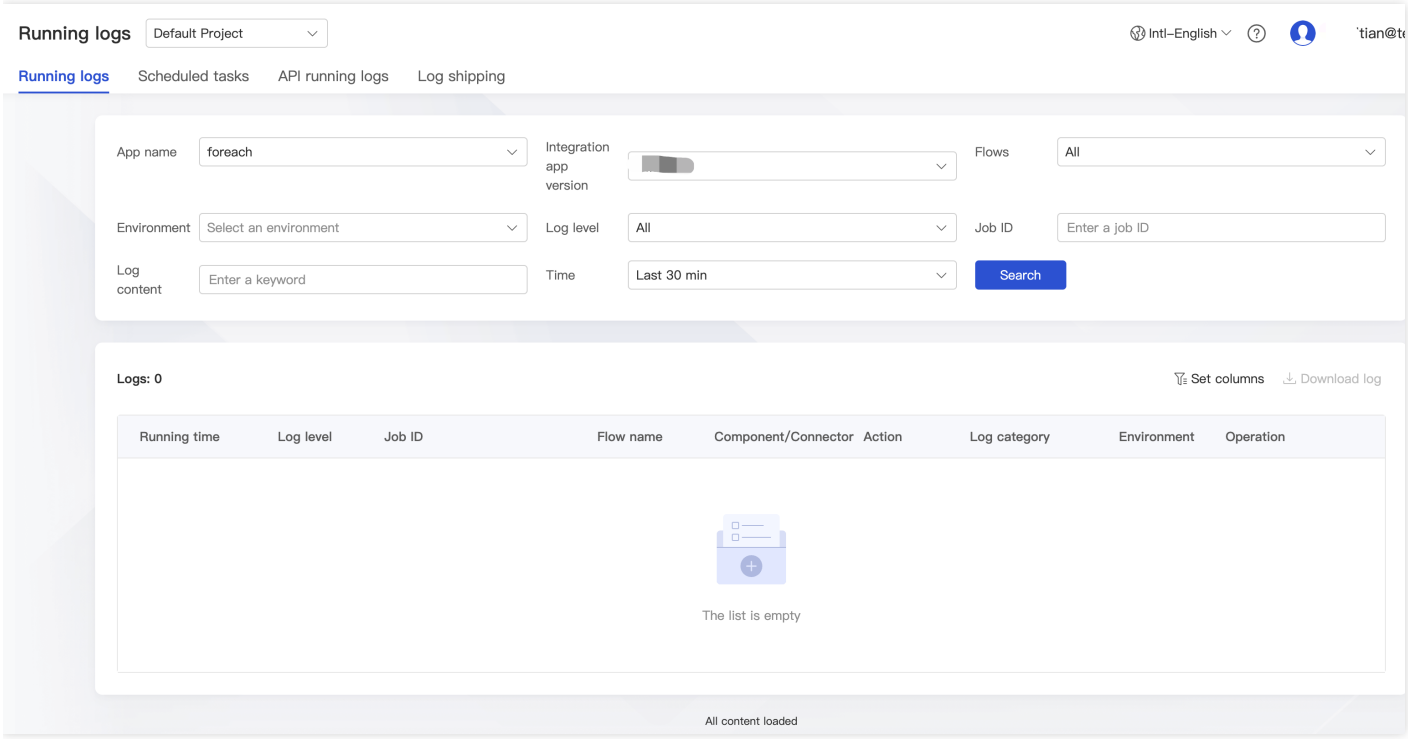
You can quickly view the running logs of a running integration app.

- Log in to the iPaaS console and click **Integration apps** on the left sidebar.

- On the **Integration apps** page, find the target app and click **Running logs** in the **Operation** column.

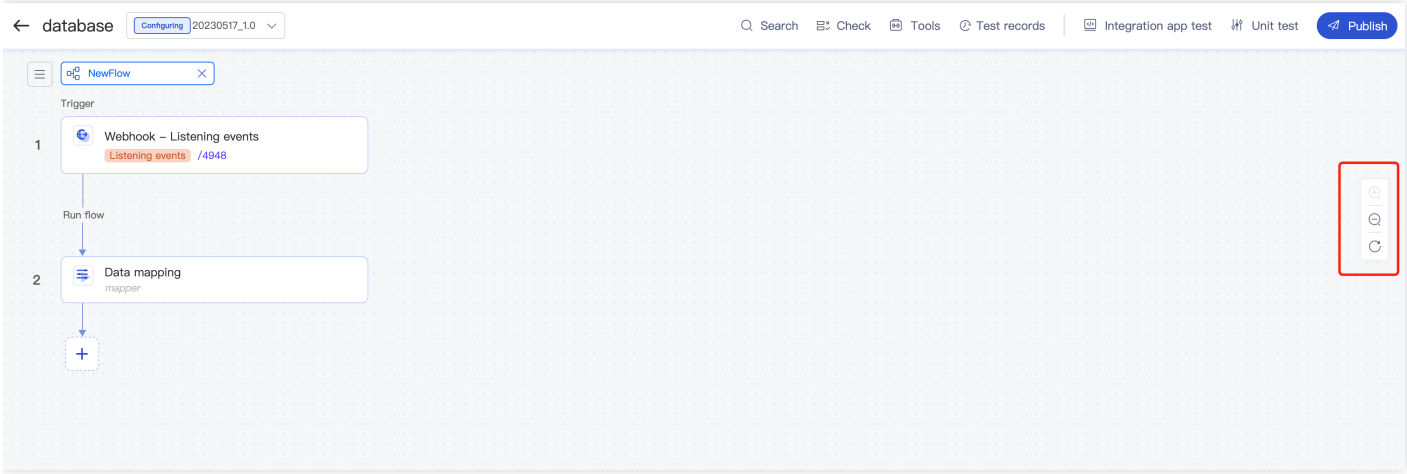


- On the **Running logs** tab, you can view the running logs of the currently running version of the currently running app (logs in the last 30 minutes are returned by default).



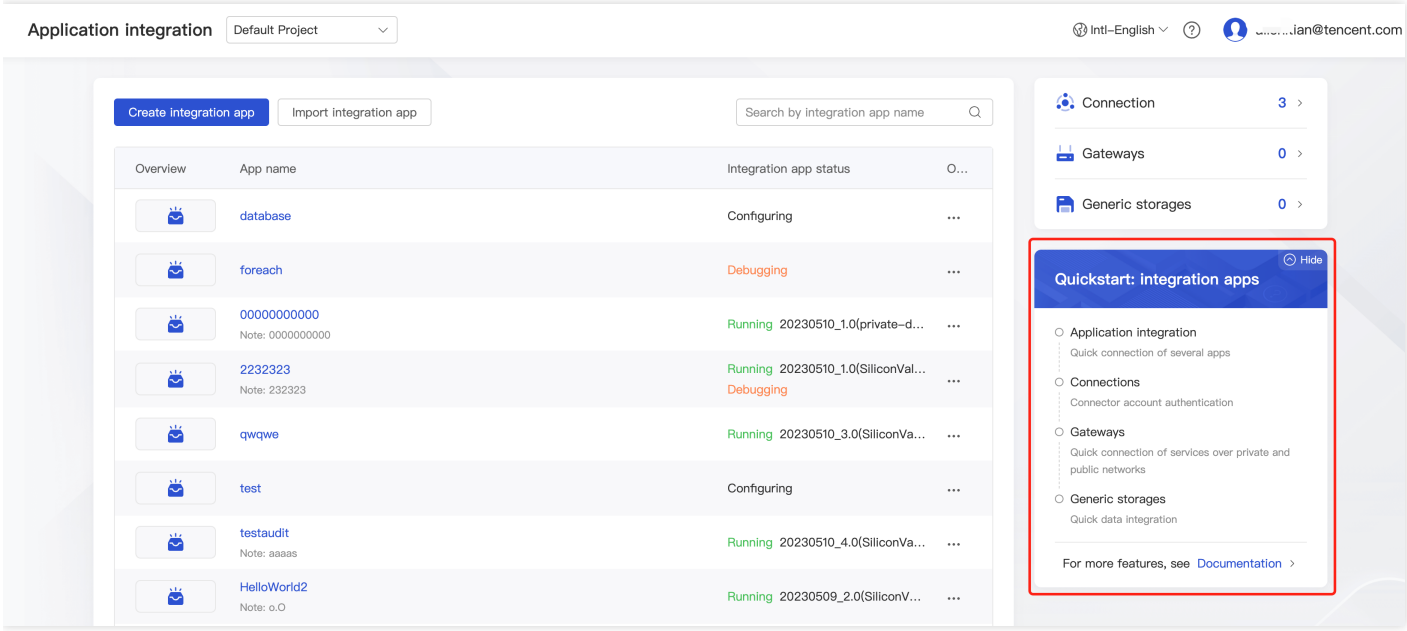
Zooming in/out the canvas

You can zoom in/out the canvas as needed.



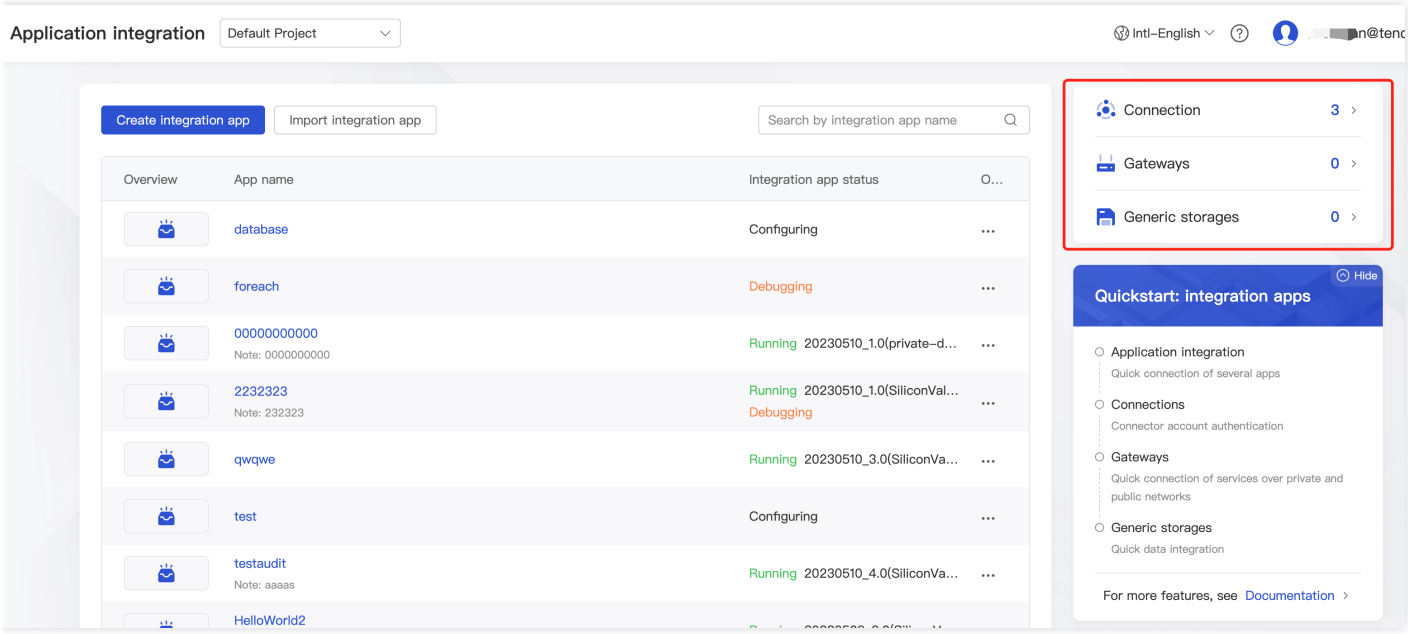
Quickly getting started with integration

Documents are provided to help you get started with integration and creating integration apps. The documents describe the concepts and feature modules of integration apps.

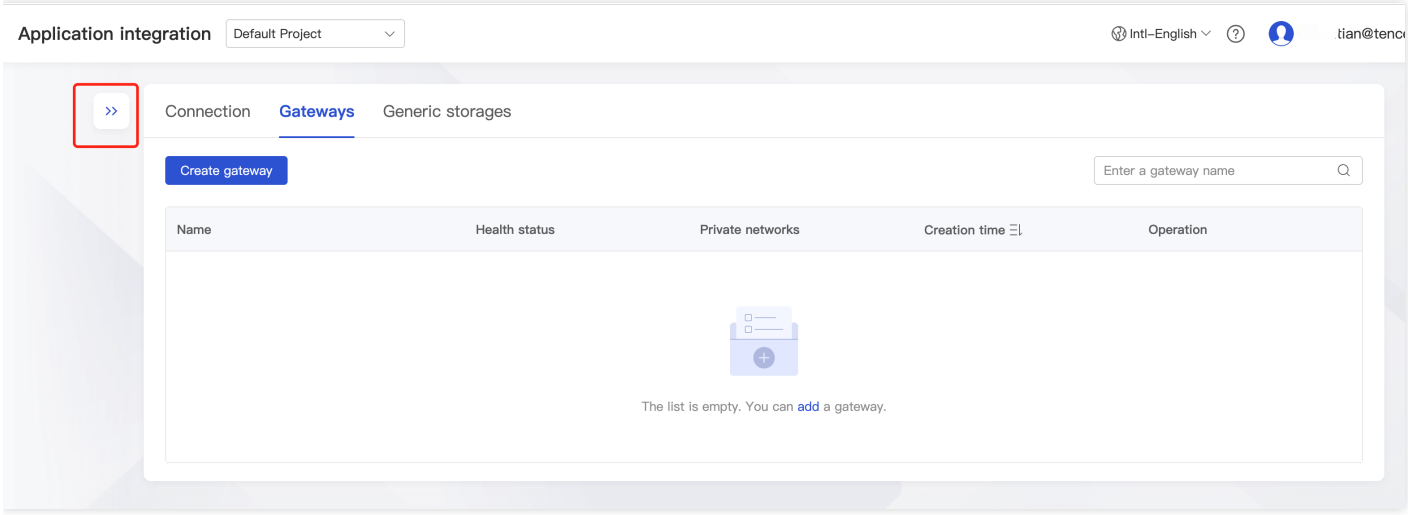


More features

You can quickly view the connections, security gateways, general storages, and global variables under the current project by clicking the corresponding feature module to enter the details page.



On this page, you can quickly add, delete, and edit connections, security gateways, general storages, and global variables. You can also click **>>** to return to the **Integration apps** page.



App Configuration

Last updated：2023-08-03 17:12:12

Overview

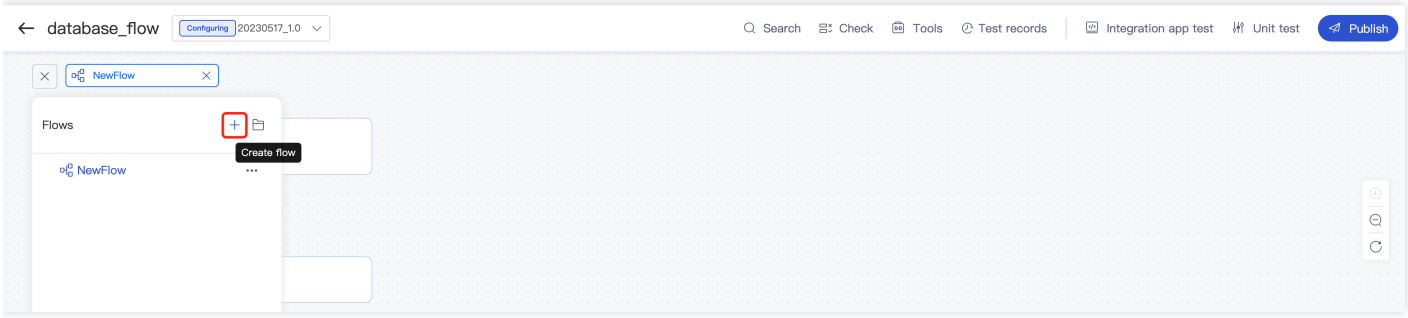
After entering an integration app, you can create multiple flows to implement your business logic. The flow canvas reserves a trigger component position for you to place a logical component or connector with trigger capabilities such as HTTP Listener, Kafka connector, and Scheduler. Flows with trigger capabilities are main flows, while those without trigger capabilities can only be referenced as subflows by other flows through the Flow Reference logical component.

Flow

In iPaaS, you can create a folder to aggregate relevant flows for hierarchical flow management. Within the same integration app, you can create and copy folders and flows for subsequent development and maintenance.

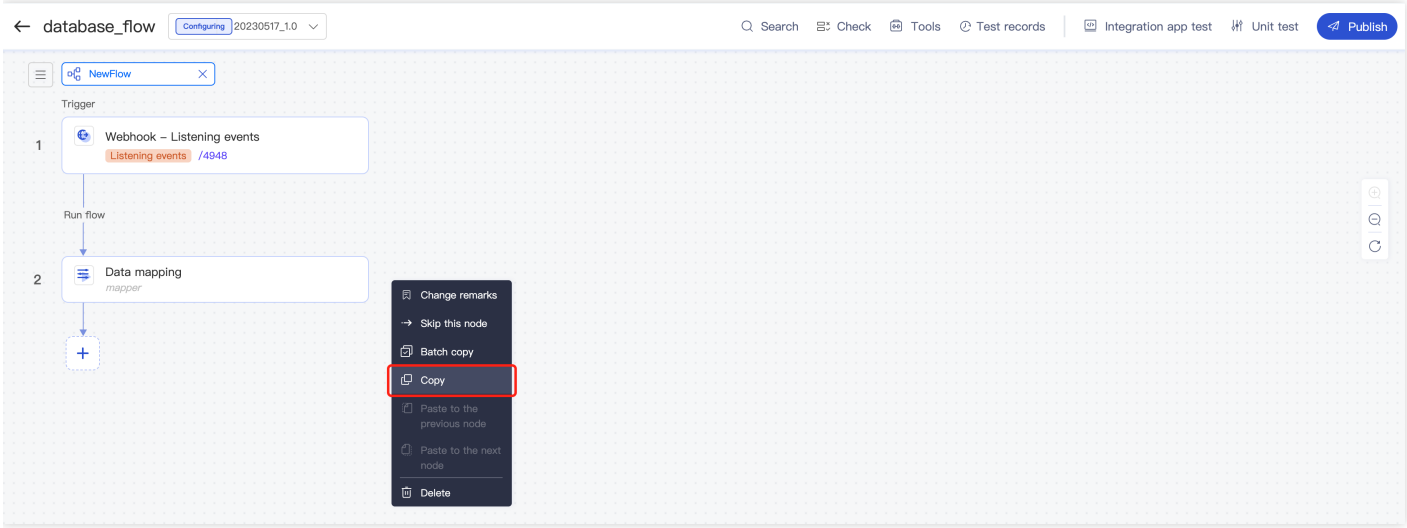
Creating a flow

You can create, copy, share, rename, and delete flows.

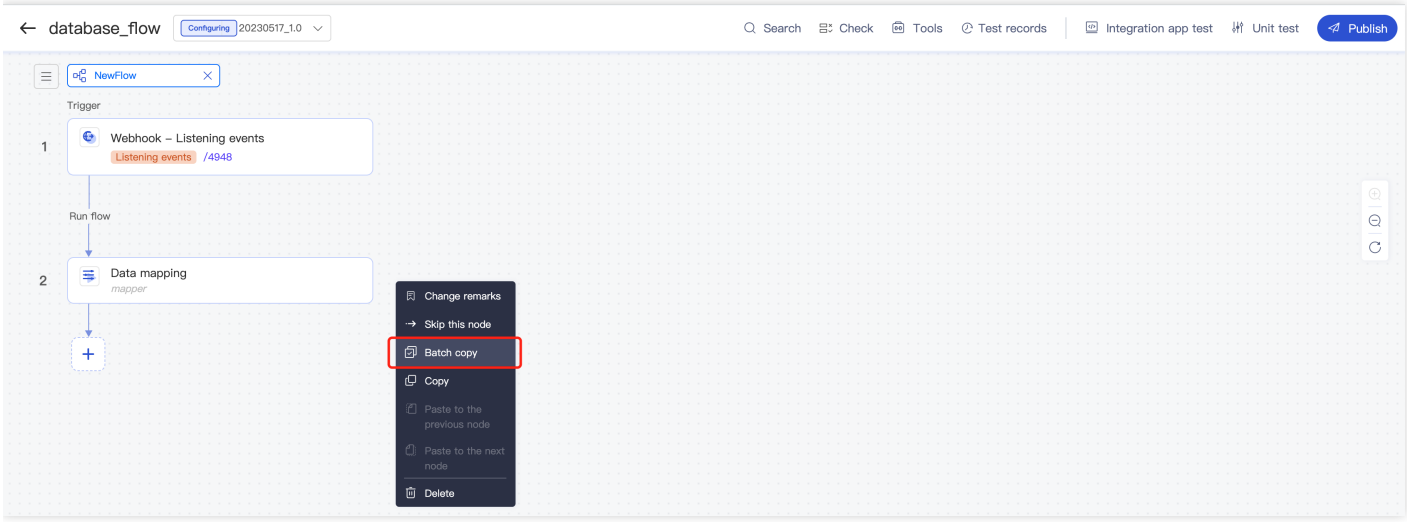


Copying a flow

You can copy flows within the same integration app. Click **Copy**, and the system will automatically create a flow named **XXX_copy**.



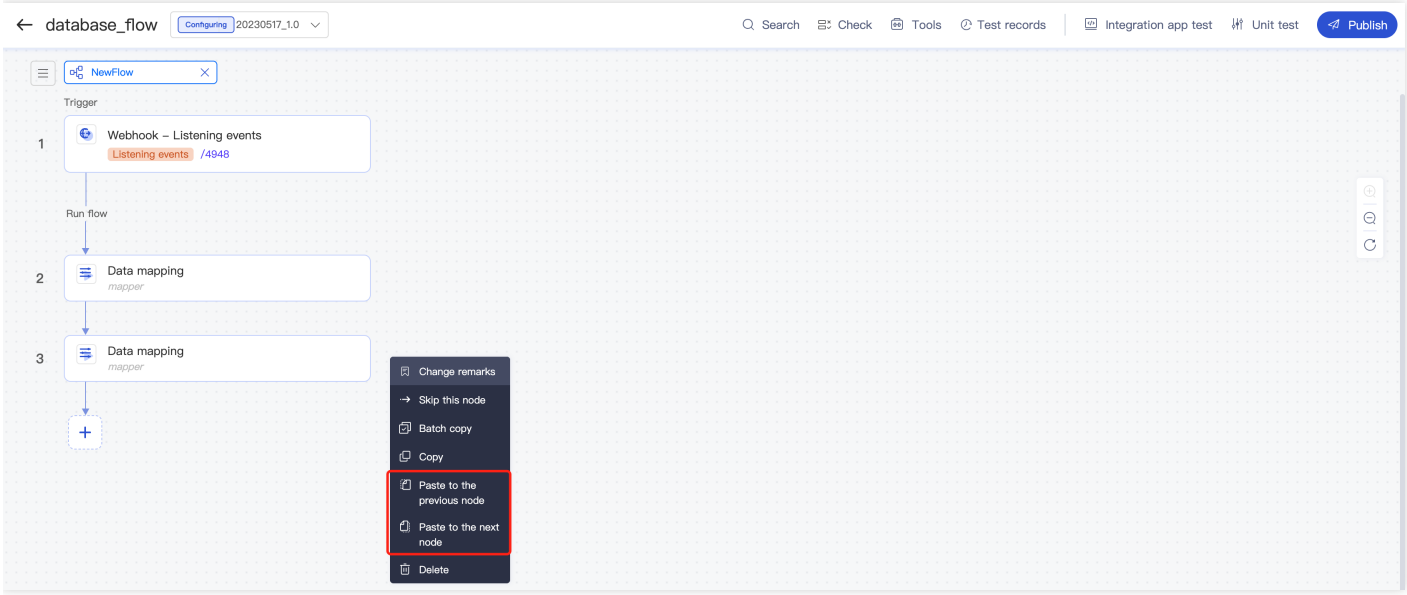
The **Batch copy** operation is also supported.



Click **Batch copy**, select all, a single, or multiple consecutive nodes (as with a unit test), and click **Copy** to batch copy the nodes.

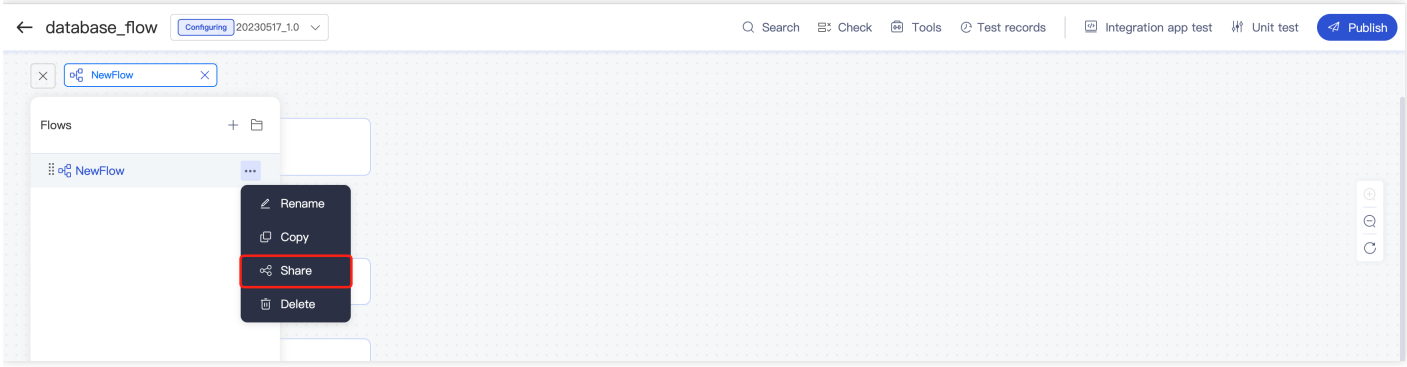
If no nodes are selected, the **Copy** button is grayed out. It will be clickable after you select one or more nodes.

After clicking **Copy**, you can paste the content into other flows within the same project.

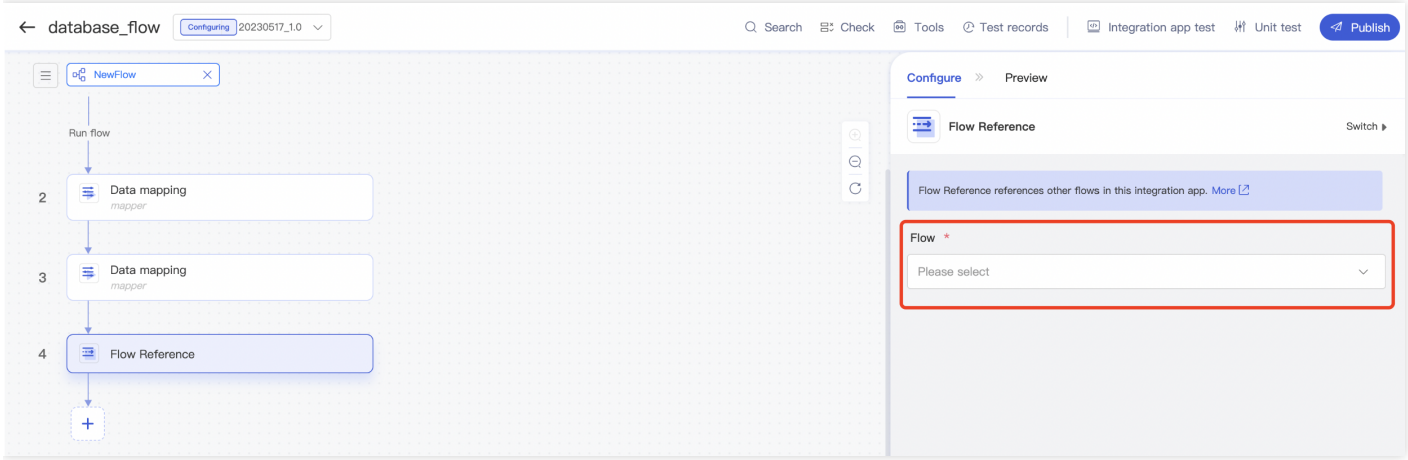


Sharing a flow

A shared flow can be referenced across integration apps within the same project through the Flow Reference component.

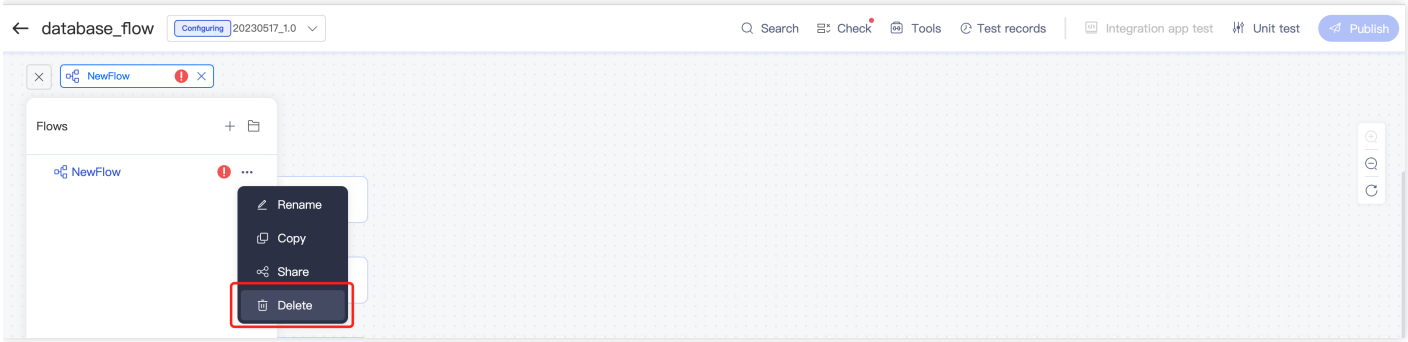


Select the Flow Reference component.



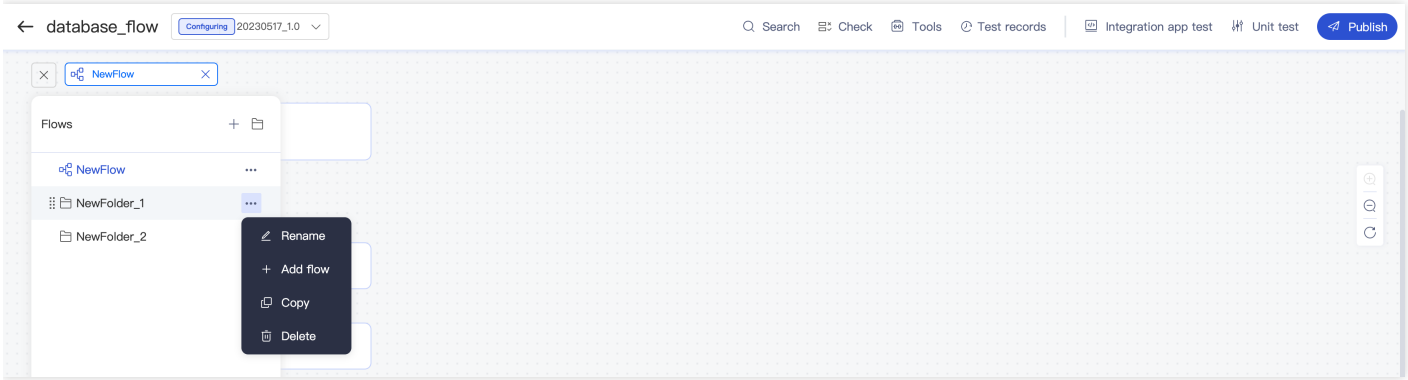
Deleting a flow

When a flow is deleted, all its configurations are cleared and cannot be recovered. Therefore, please exercise caution when deleting a flow.

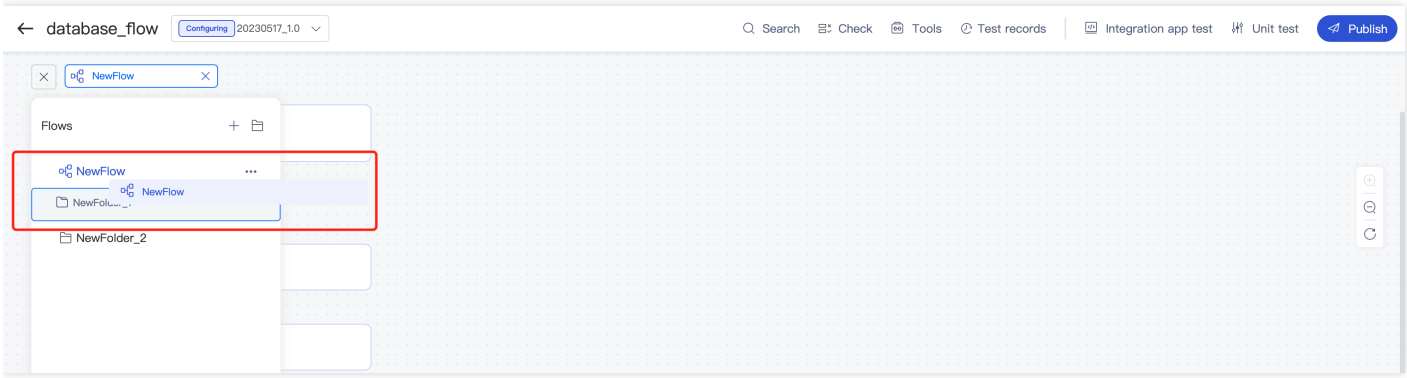


Folder

A folder can be used to aggregate relevant flows. You can create, copy, and add flows to folders for flow management and maintenance.



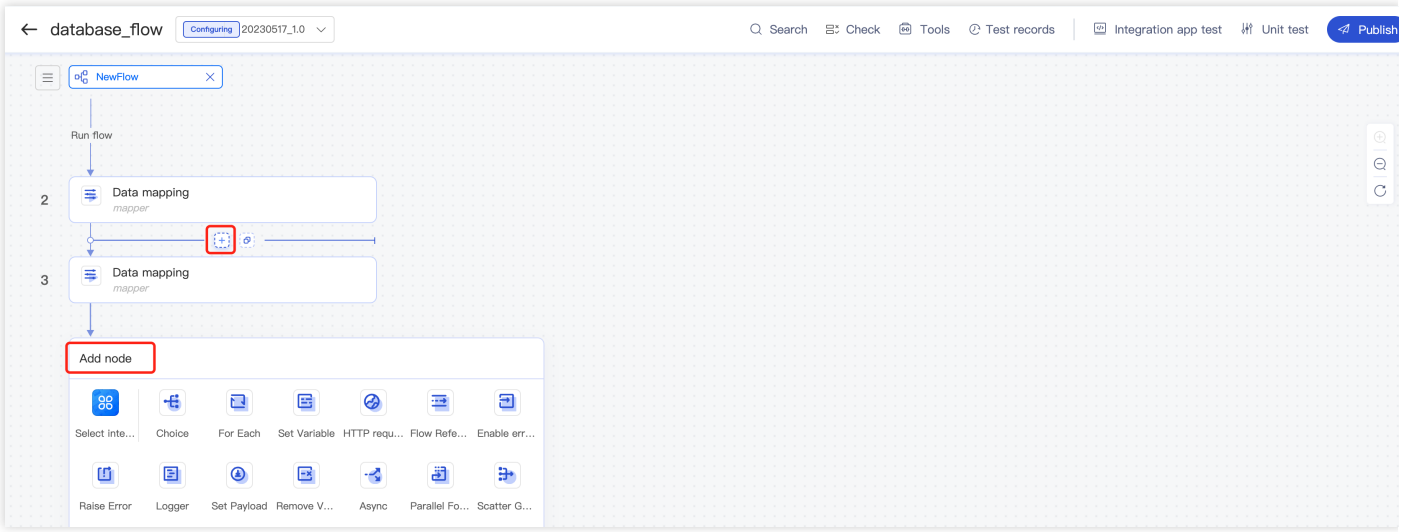
You can drag and drop existing flows to adjust their order or drag and drop them to a folder for unified management. When dragging a flow, you can drop it after a blue line appears.



Component Library

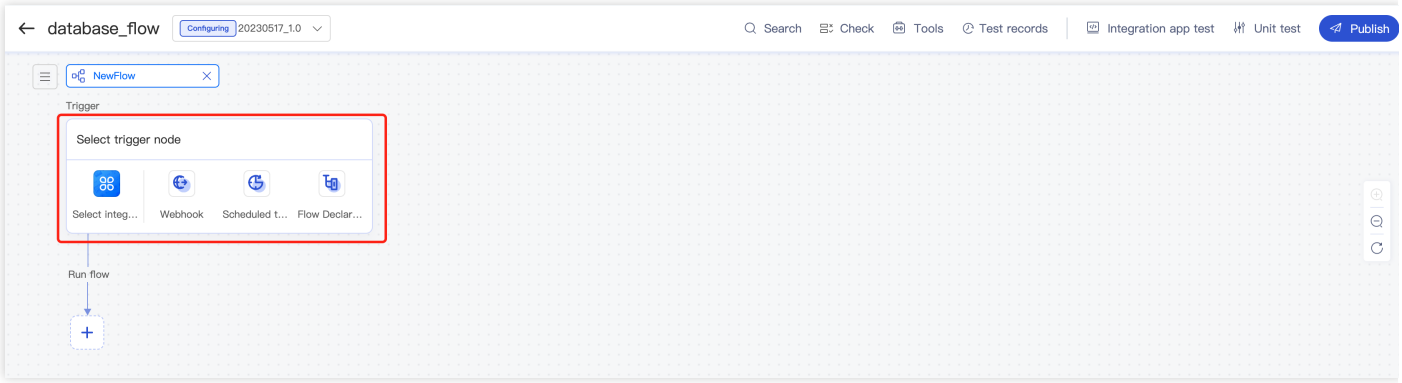
After a flow is created, it will be displayed on the canvas. Click **+** on the canvas to open the component library, select a component, and configure the flow. Components include connectors and logical components, the former of which include common connectors and app connectors.

- Common connector: Protocol for app system interaction.
- App connector: Encapsulation of an app. For example, if a system interacts with other systems over HTTP, you can select the HTTP connector and configure the parameters for connection; if you want to connect to Tencent Meeting, you can directly use the Tencent Meeting connector, which encapsulates the APIs, authentication information, and interaction protocols of Tencent Meeting.
- Logical component: Implementation of business logic for app interconnection, such as loop, choice, variable configuration, data conversion, parallel processing, and async processing.

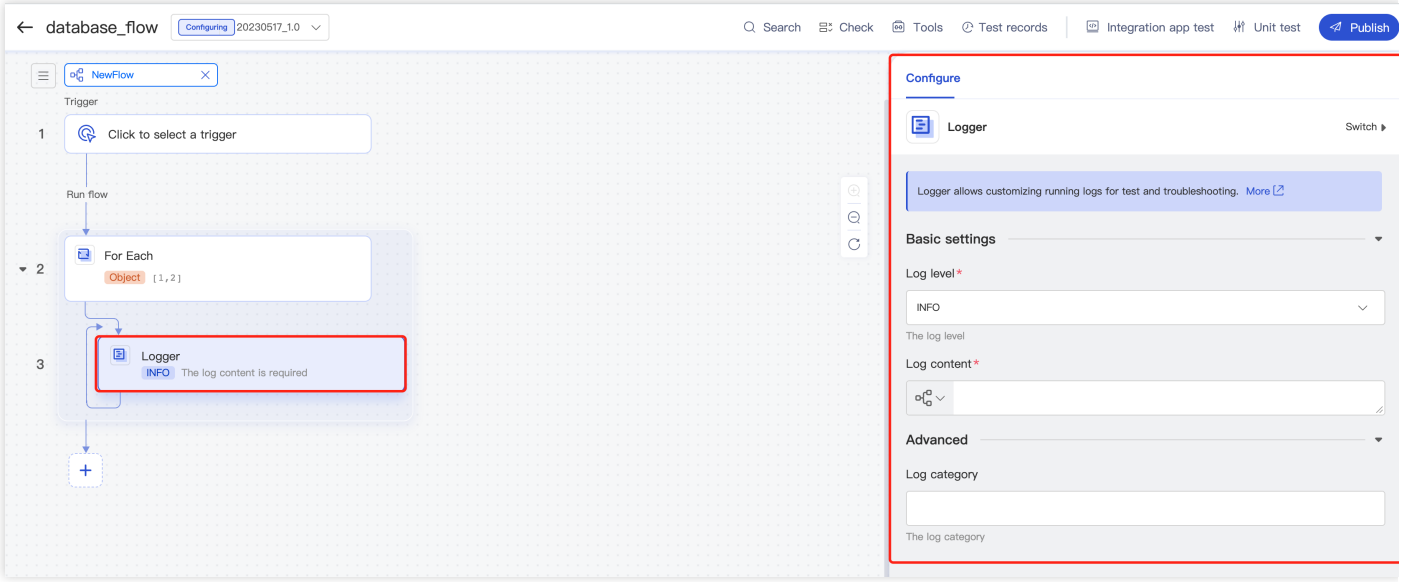


Logical component operation guide

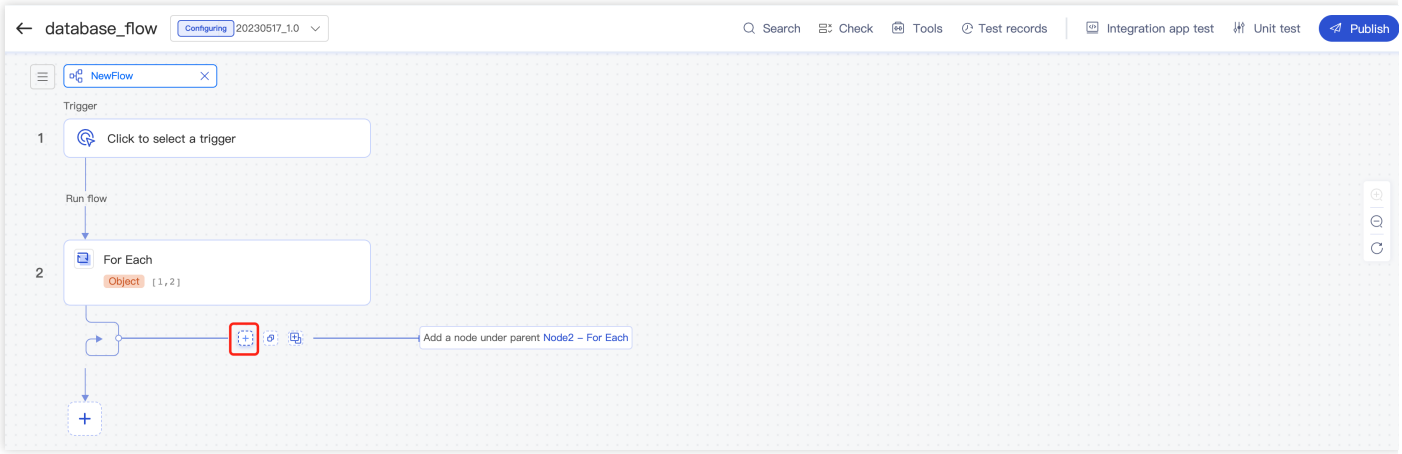
1. For a newly created flow, click + to select a trigger component.



2. When you add a sub-node, the information of the parent node will be displayed.

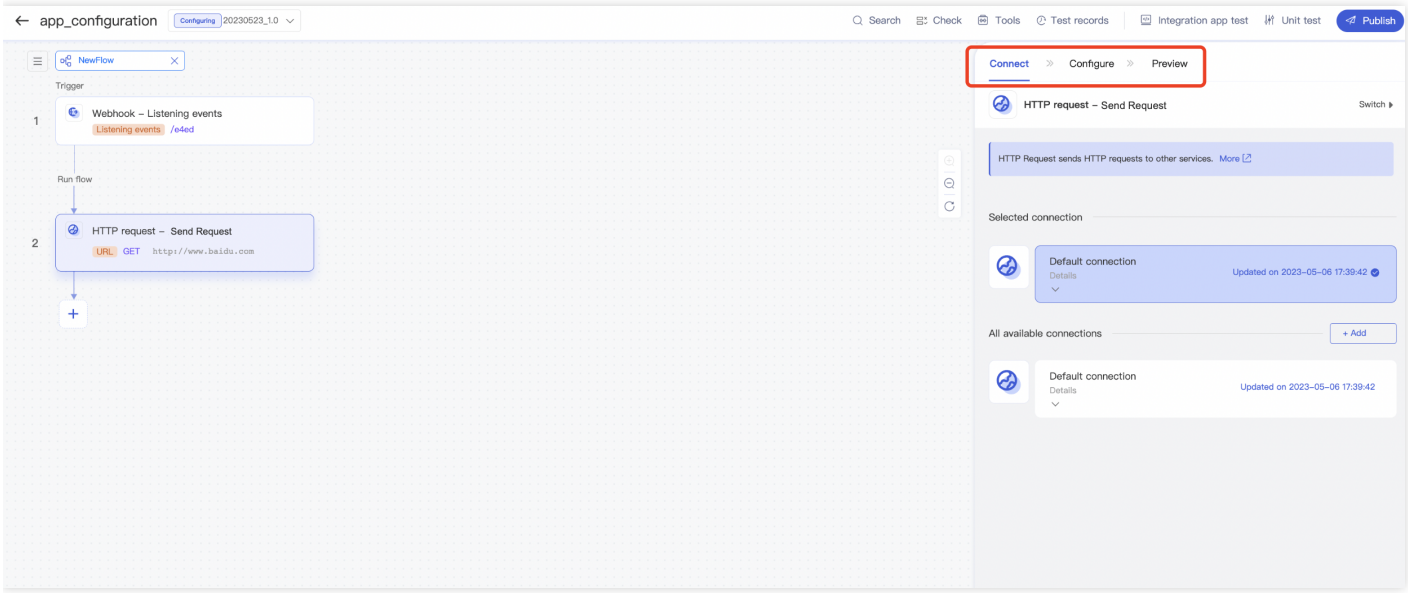


3. If you add a logical component, the component configuration page will be directly displayed. Then, configure the component as prompted.



Common/App connector operation guide

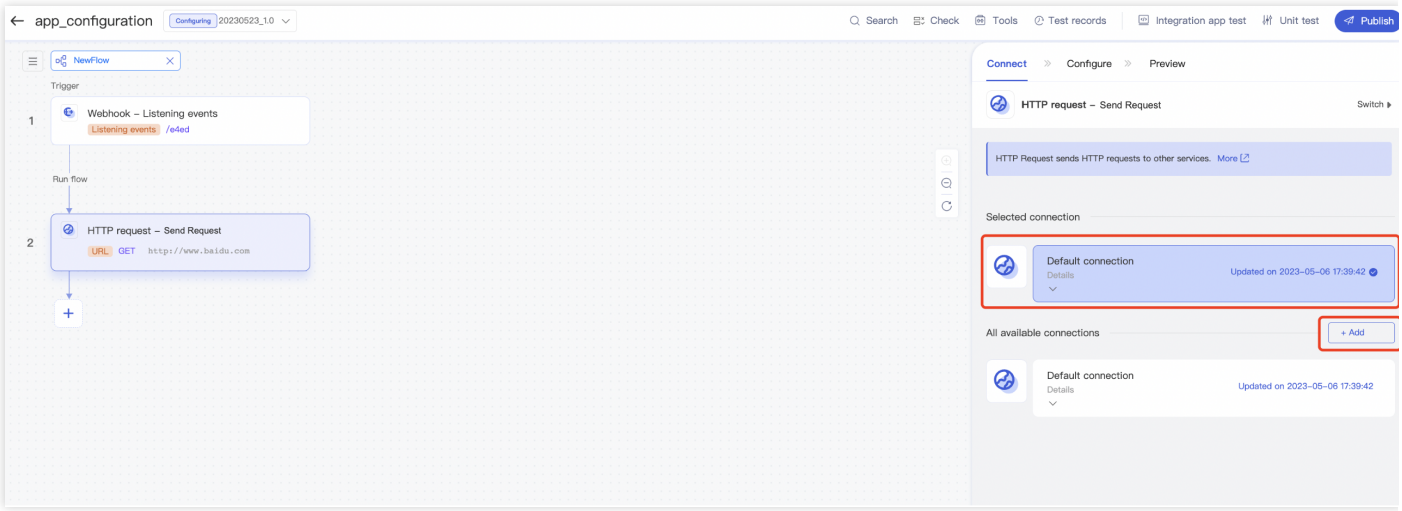
For a newly created flow, click **+** to select a trigger component. If you select an app connector or common connector, select the required operation first and then configure the app connector or common connector as prompted. General configuration items are required, while advanced configuration items are optional. After completing the configuration, click **Execute** in the **Preview** step to check whether the configuration is correct and whether the output data is as expected and can be used on the next node. You can **click the data to input it**.



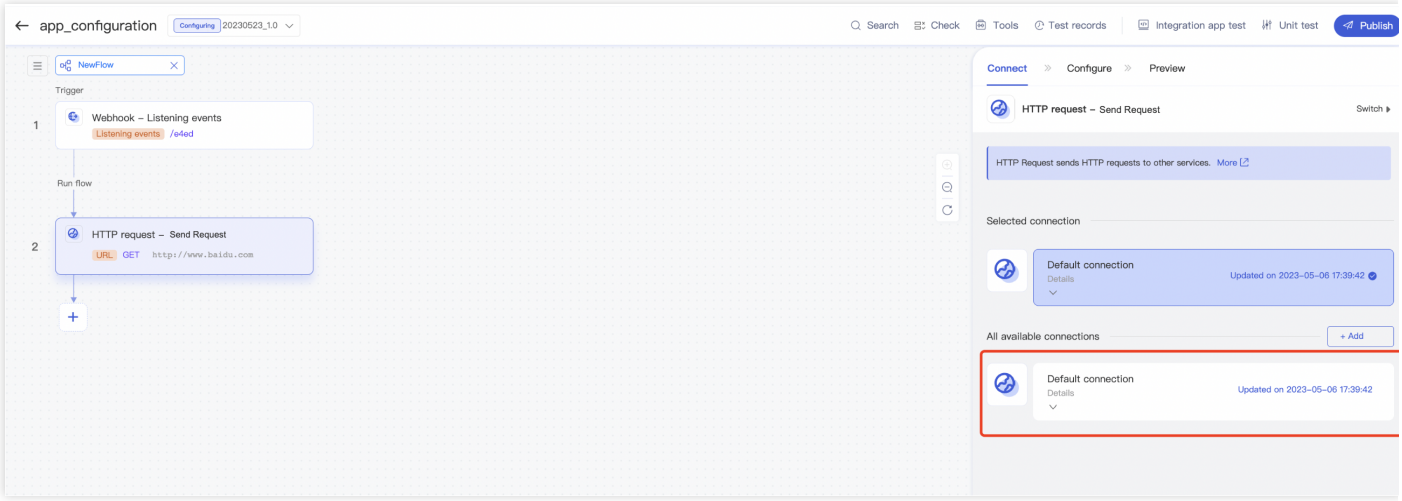
Connection

- Create a connection: After you select **Connector** and create a connector on the canvas, the message "No connections are bound yet. Select an existing connection or create one" will be displayed in the top-left corner of the

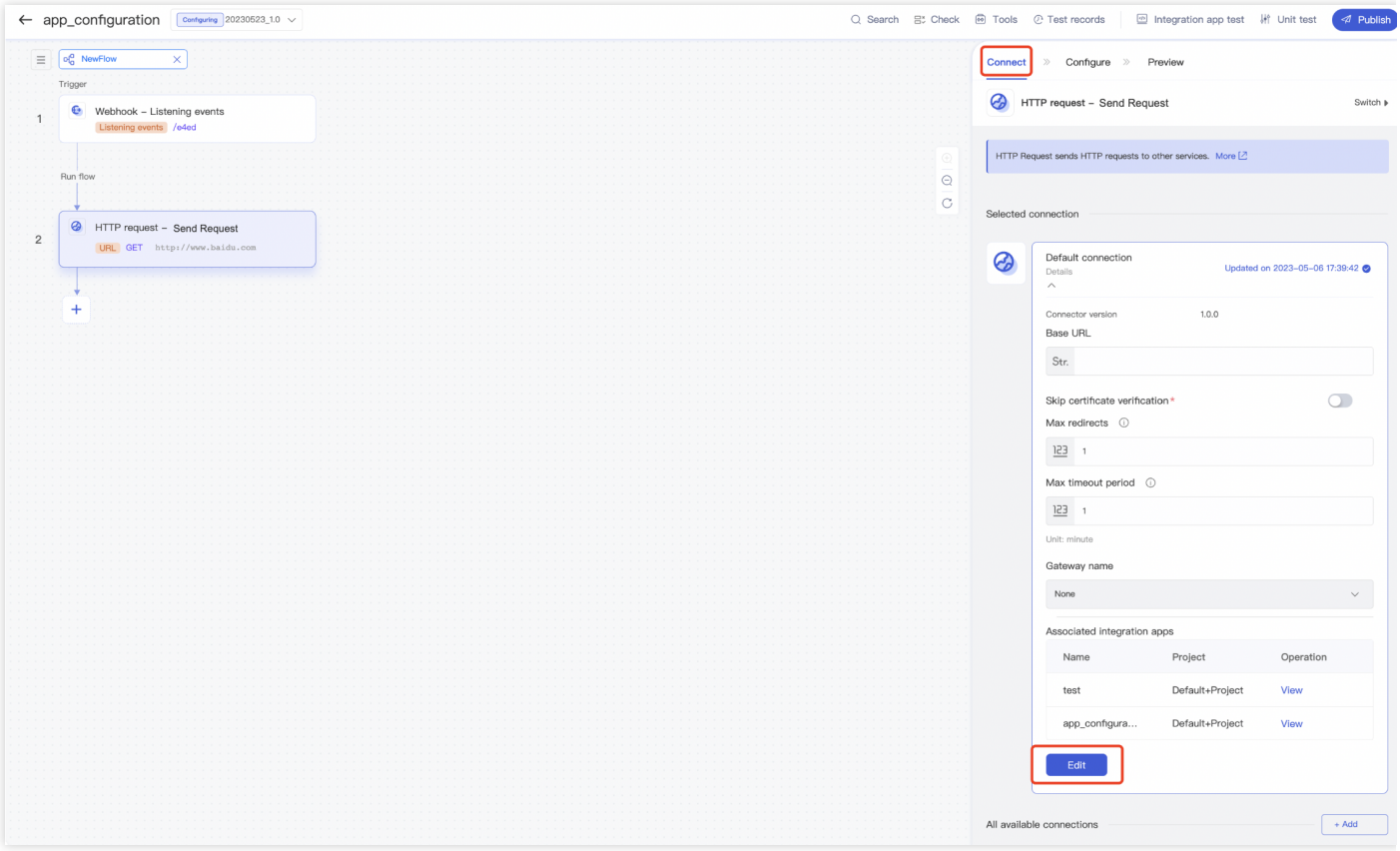
pop-up configuration page. Click **Add connection** and configure the information as prompted.



- Select an existing connection: For a non-newly created integration app, if there are existing connections in the same app, you can select one as needed. To modify the third-party credential information, you can return to the **Connection** page to create a connection or switch to another existing one.



- Connection: When a connector is created, its connections will be displayed on its **Connection** page. You can click **Edit** to enter the connection page and edit the connection information.



- Switch the app connector version: If the selected app connector has multiple versions, you can click **Tools** in the top-right corner and click **Connector version** to switch to the target version.

> Note：
> If there is a red exclamation mark after the selected version, the version is not the latest one.

- Common connector configuration

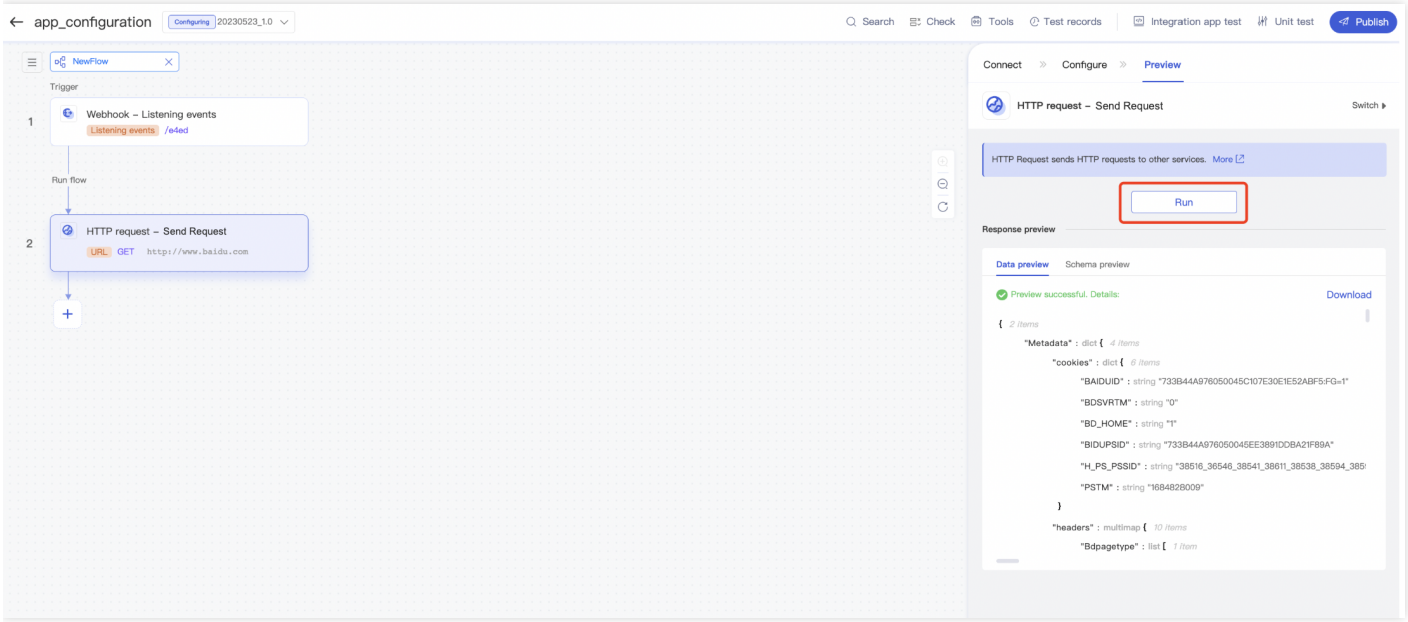
> Note：
> Common configurations are displayed only for apps created after the launch of the standalone console.
> Reason: Connection configurations of common connectors (like HTTP Request) of apps created before the launch of the standalone console are called common configurations. Configurations of both common connectors and app connectors of apps created after the launch of the standalone console are collectively called connection configurations; therefore, common configurations are no longer displayed.

- Edit a common connector configuration: Select the target common connector, click **Tools** in the top-right corner, select **Connection**, and click the name of the target configuration to edit the configuration.

- Delete a common connector configuration: Select the target common connector, click **Tools** in the top-right corner, select **Connection**, and click ••• to delete the target configuration.

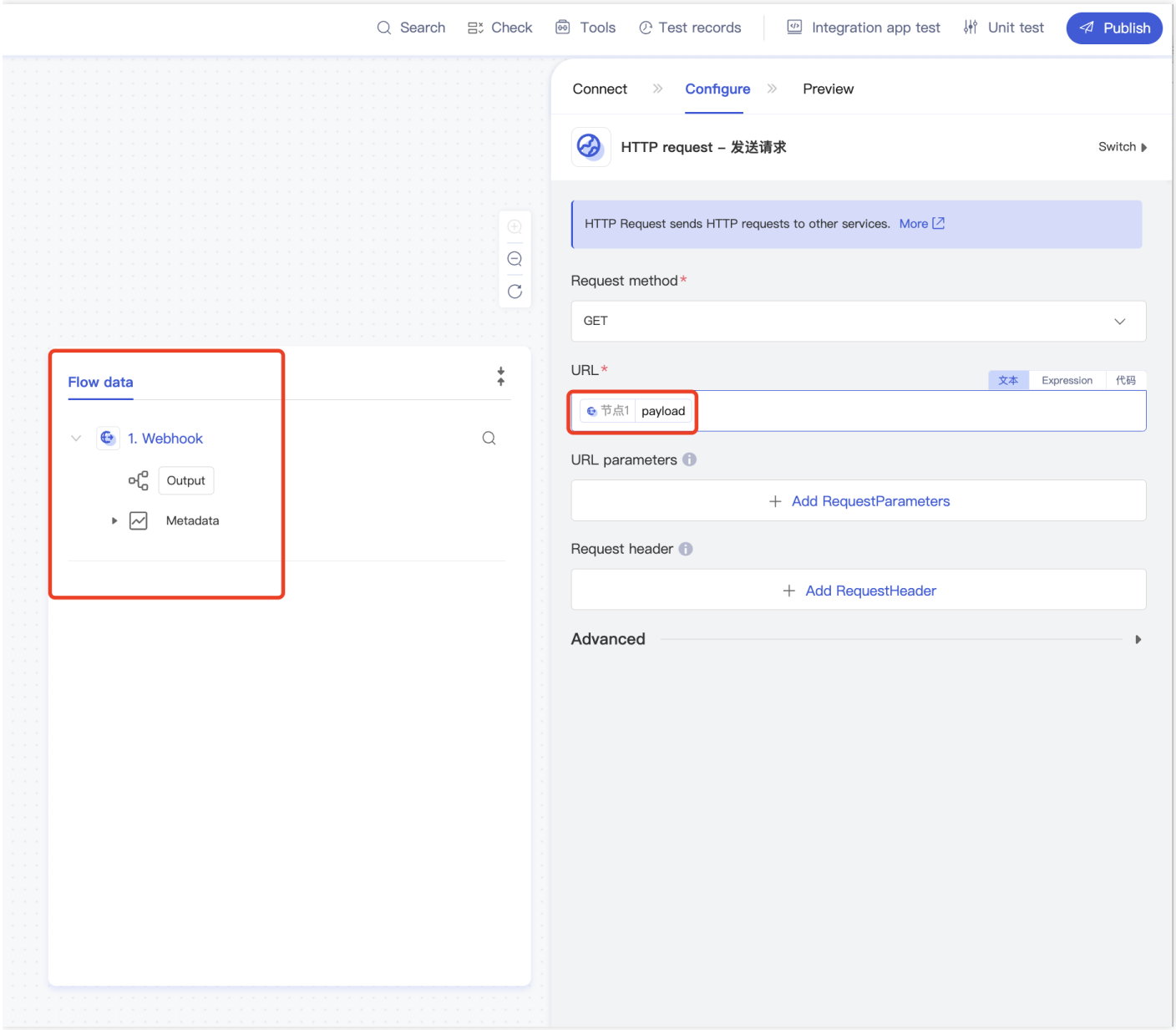
Online preview

After configuring a node, you can view the node's output data and schema immediately in the preview step.



Flow data panel

After you click a flow node, the flow data panel will be displayed. You can select the data content on the previous nodes to reference data variables easily. If multiple variables are selected, they will be spliced automatically.



Single-line expression input

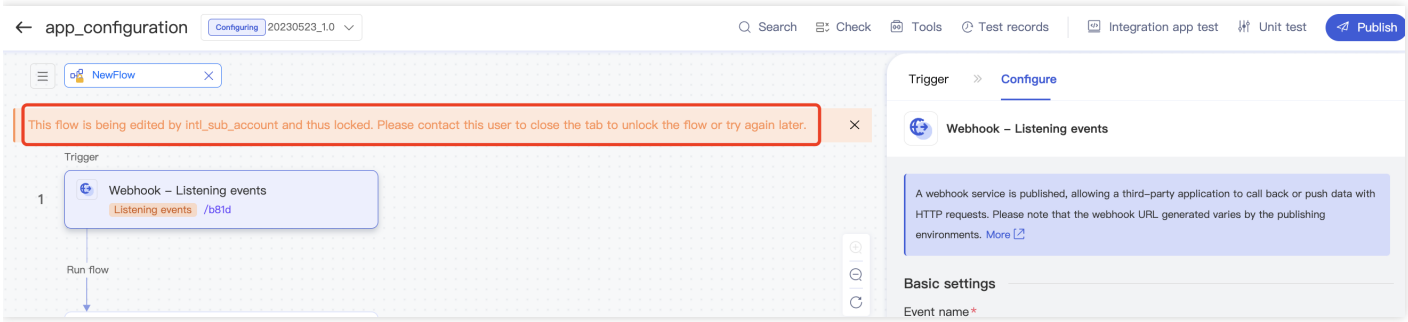
If you want to simply edit some variables when entering parameters, you don't need to open the code input box. Instead, you can switch to the **Expression** input mode to perform over 100 quick operations for orchestration, such as type conversion, operator, method reference, and attribute reference. The input process is further simplified through capabilities such as smart prompt and autocomplete.

Multi-User Collaboration Mode

iPaaS supports **multi-user collaboration**. This feature allows multiple members in your team to develop the same integration app at the same time. In addition, you can enable the editing lock to avoid version inconsistency caused by

simultaneous editing of the app by multiple users.

> Note：
> In multi-user development, multiple users can develop multiple flows together in the same integration app, but a single flow can be edited by only one user at a time and is locked (uneditable) for other users during editing. This helps avoid version inconsistency caused by simultaneous editing of the app by multiple users.



Testing Feature

Integration app test

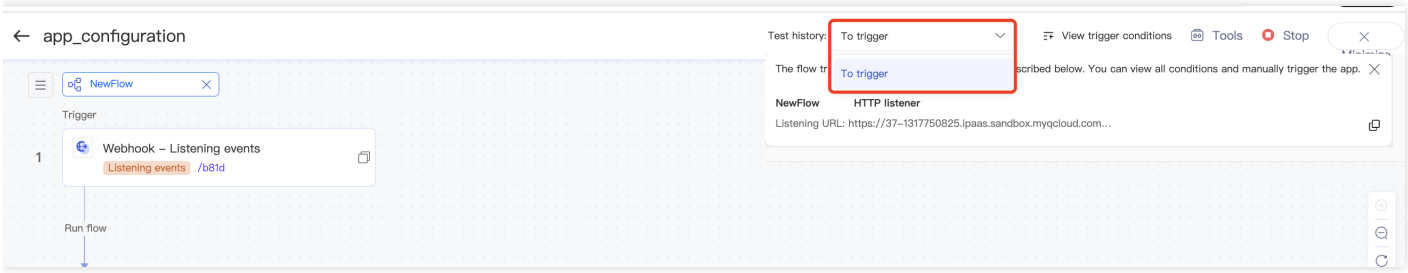
Viewing test information and errors

During integration app test, you can publish an integration app to the sandbox environment, directly copy the URL after the trigger, and simulate a trigger operation to debug the entire app. After the operation, the operation result will be displayed on the canvas. You can click the name of each component to view the detailed input and output information. All errors are displayed at the top of the page, and you can click **View** to view the error details of each component.

Viewing historical test records

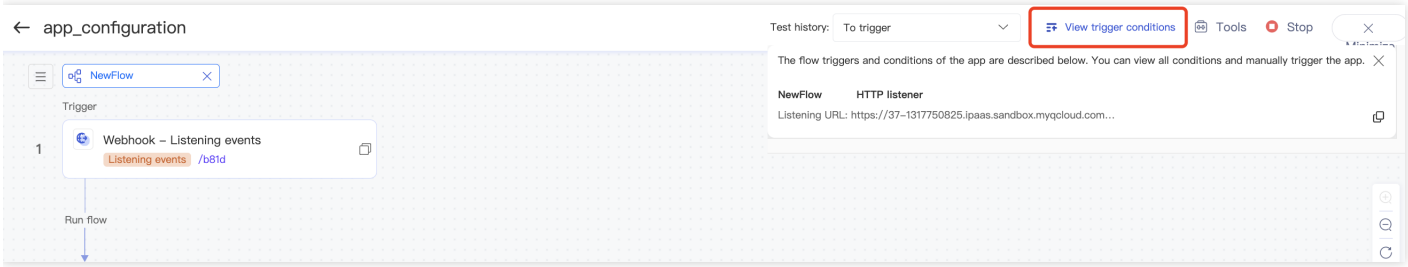
Click the **Test history** drop-down list to view the test snapshot at each time point. In each snapshot, you can view the specific test parameters and errors at the corresponding time point. The last 10 test records are displayed to help you

check the errors reported during each test.



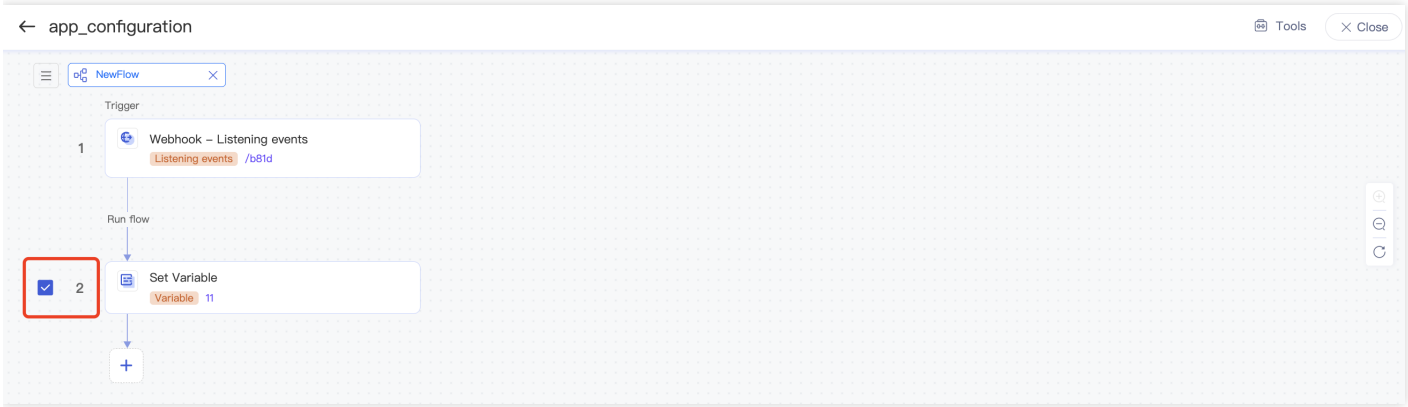
Viewing trigger conditions

You can view all trigger conditions of the current integration app and manually trigger the app. The following example shows how to manually trigger the Webhook component:



Unit test

In unit test, you can partially test one or multiple consecutive components. If the flow is complex, it may take a long time to test the entire app. In this case, you can perform partial testing to quickly locate possibly abnormal components (the first trigger component of the flow cannot be selected). To perform unit testing, click **Unit test** in the top-right corner and select the components to be tested. You can select only the first and last target components, and the nodes between them will be selected automatically, eliminating the need to click nodes one by one.



After you select the nodes for unit test, you need to simulate the input data for test. If you have performed unit or app test or previewed the component data before, the platform will automatically pull the historical test data, and you can

select different historical data or automatically construct input data for test. If there is no historical data, you need to manually enter the simulated data for test.

Minimizing the test window

The test feature in the standalone console manipulates a certain version of the current app. You can still configure the app after minimizing the test window.

- In app test, you can click **Minimize** to minimize the app test window.
- In unit test, after you click **Close**, the unit test window will be minimized automatically.

> Note：
> As the configured version and the tested version are separate, we recommend you test the app again after configuring it to verify the availability of the latest configured version.

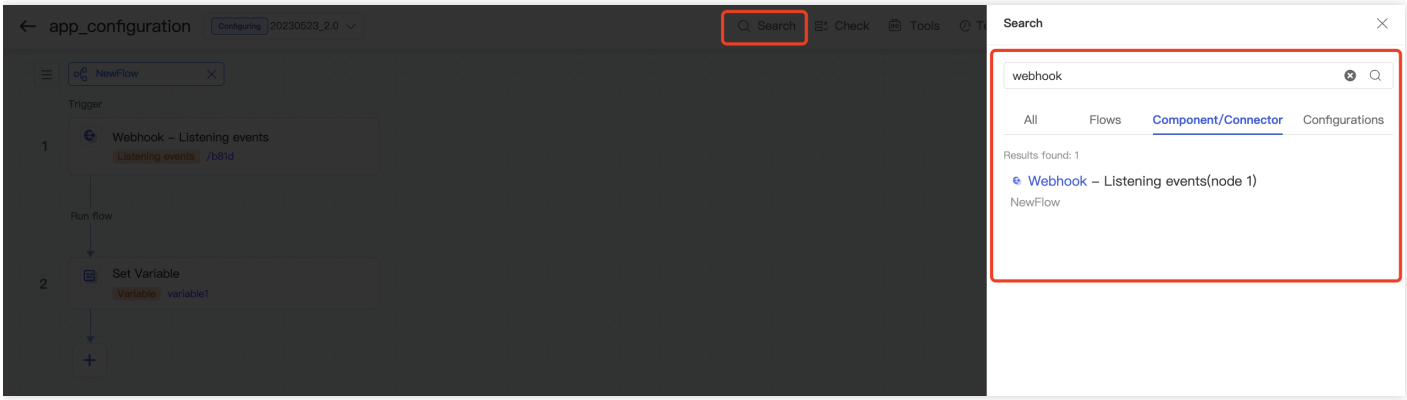
You can also click the minimized test window to restore it and view the test details.

Viewing test records

After minimizing the test window, click **Test record** to view the historical app or unit test records.

Search

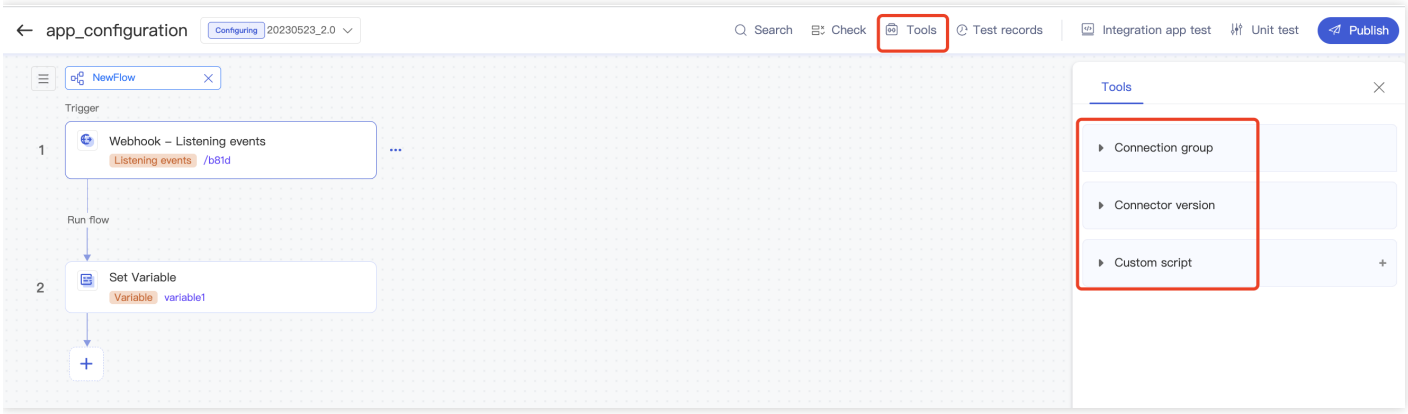
You can search for flows, components/connectors, and configuration content, so as to quickly confirm the valid content and built-in fields in the app.
Click **Search** and enter a keyword. Here, keyword `webhook` is used as an example.



Tools

The toolbar provides many features such as connection group switch, connector version switch, and custom script configuration.

Note：
Common configurations are displayed only for apps created after the launch of the standalone console.
Reason: Connection configurations of common connectors (like HTTP Request) of apps created before the launch of the standalone console are called common configurations. Configurations of both common connectors and app connectors of apps created after the launch of the standalone console are collectively called connection configurations; therefore, common configurations are no longer displayed.



Switching connection groups

You can switch between connection groups on the toolbar for preview and testing.

Switching connector versions

Switch the app connector version: If the selected app connector has multiple versions, you can click **Tools** in the top-right corner and click **Connector version** to switch to the target version.

Note：
If there is a red exclamation mark after the selected version, the version is not the latest one.

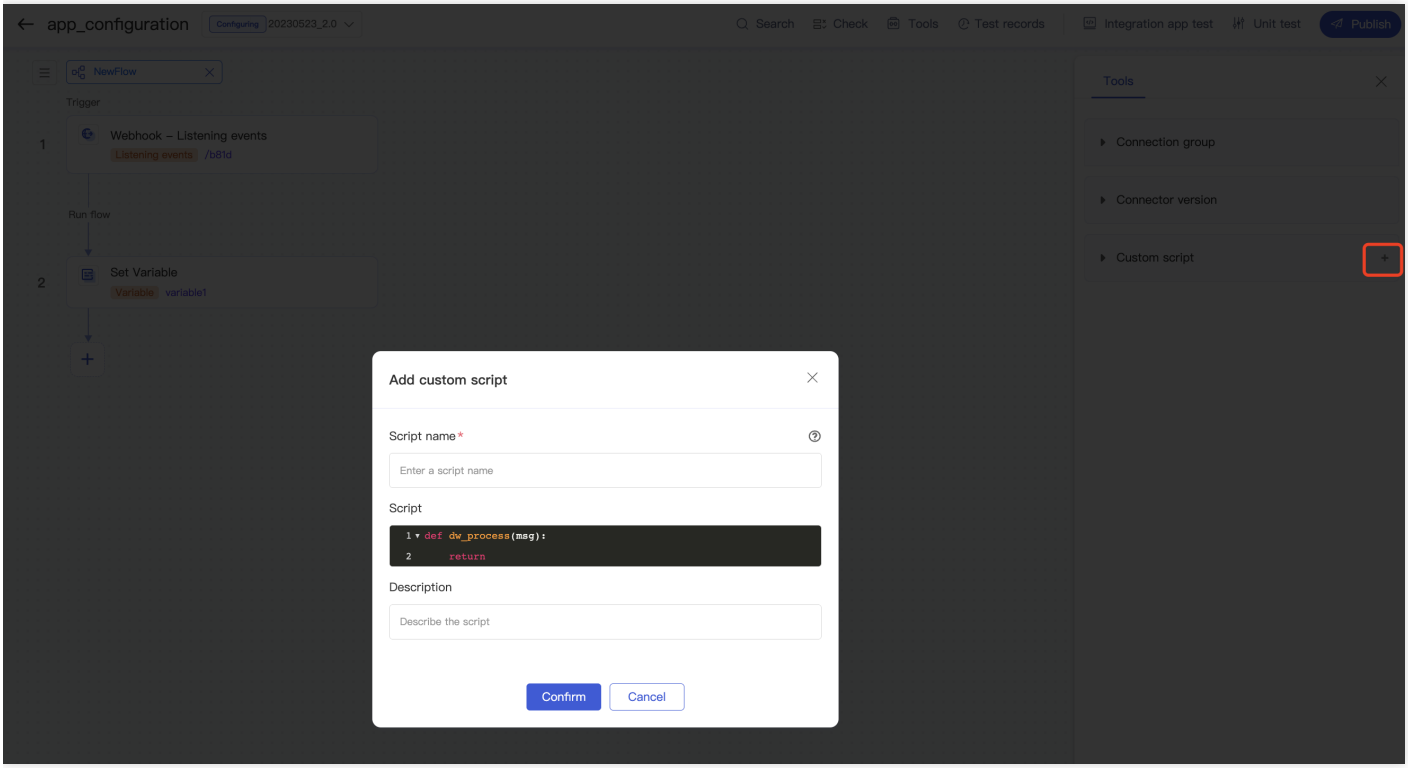
Custom script

iPaaS offers the custom Dataway script feature, which enables you to write custom functions and components. Custom scripts can be referenced only in the integration app where they are created. They cannot be referenced across projects or integration apps.

Adding/Deleting a custom script

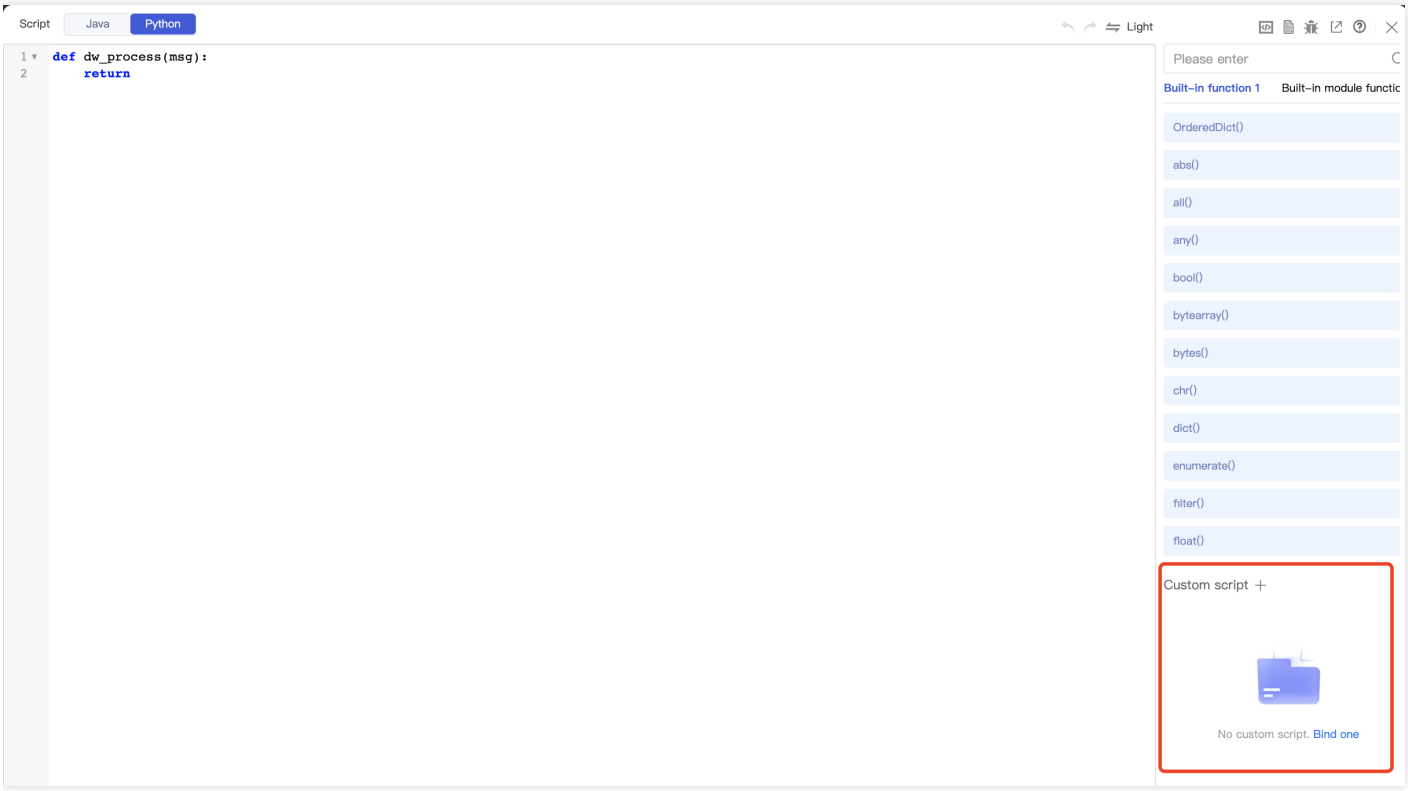
Click **Tools** in the top-right corner and click **+** next to **Custom script** to add a custom script. Click a custom script name to view or modify the script. Click ••• to delete a script.

> Note：
> Once a script is deleted, apps bound to it will throw an exception. Therefore, please exercise caution when deleting a script.



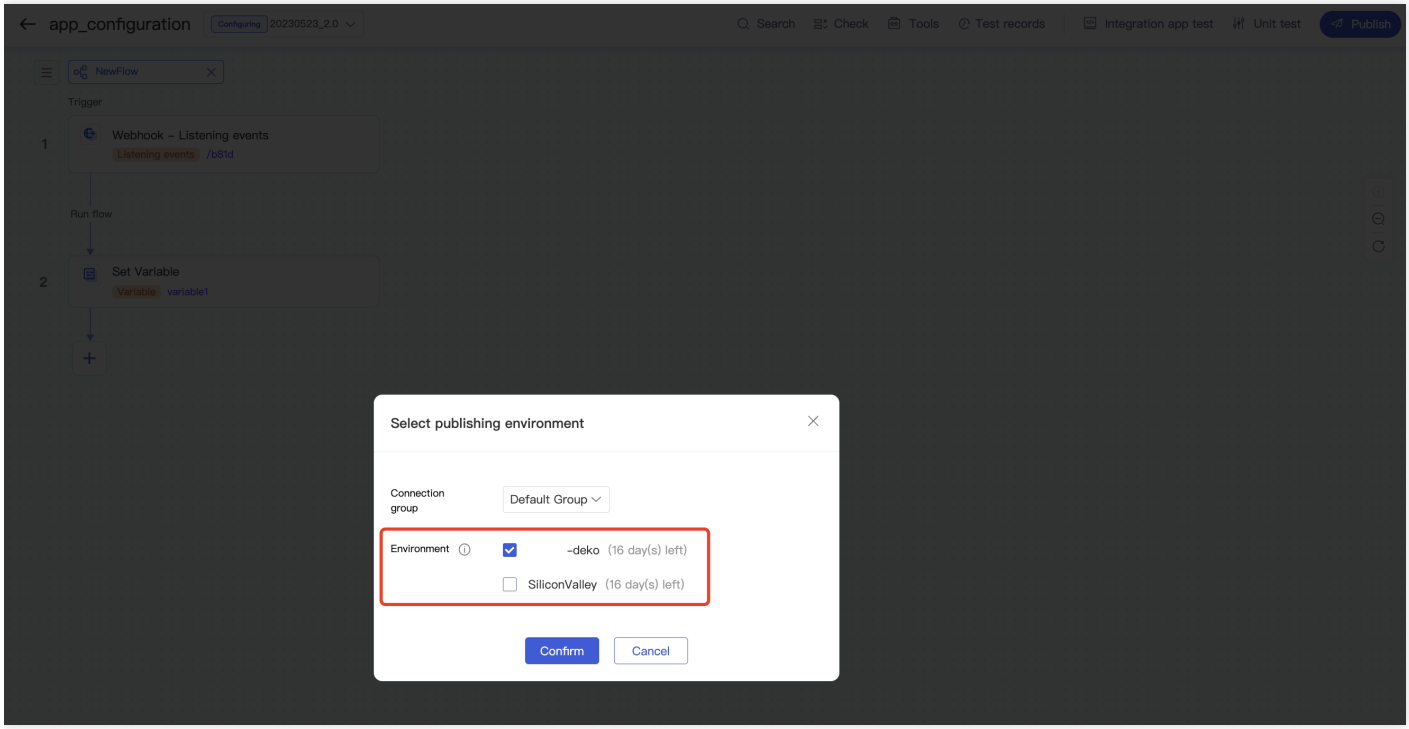
Referencing a custom script

After you select a component and click a Dataway expression for configuration, click </> in the top-right corner or **Click to bind** to reference an existing custom script.



Publishing

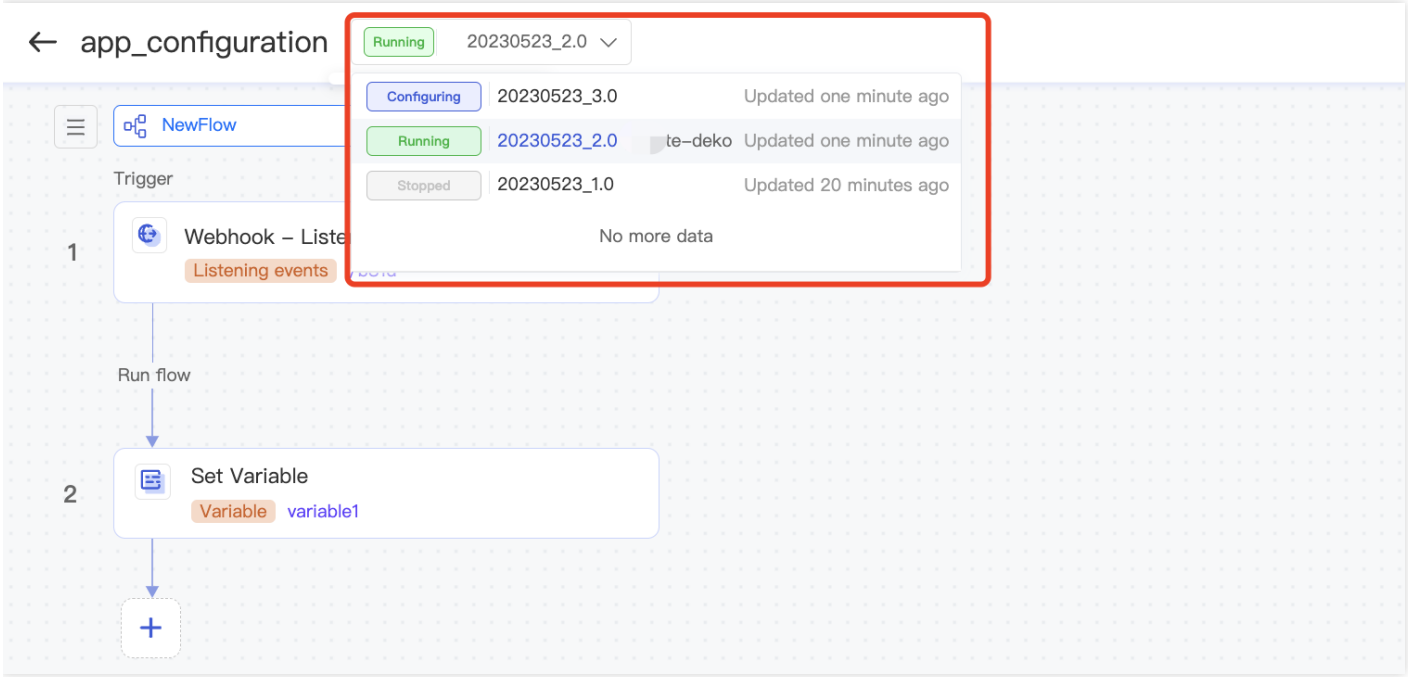
After you have configured an integration app, you can publish it in one or multiple regions.



Versioning

Each integration app has a version number. After it is published, an integration app cannot be modified while it is running. In this case, you can copy the integration app to modify and debug a new version, which will not affect the version that is currently running.

Note：
The **Publishing environment** field is added for apps created after September 20, 2022. When you publish an app again, the publishing environment selected last time is retained for easier management.



Logical Component User Guide
Remove Variable

Last updated：2023-08-03 17:12:12

Overview

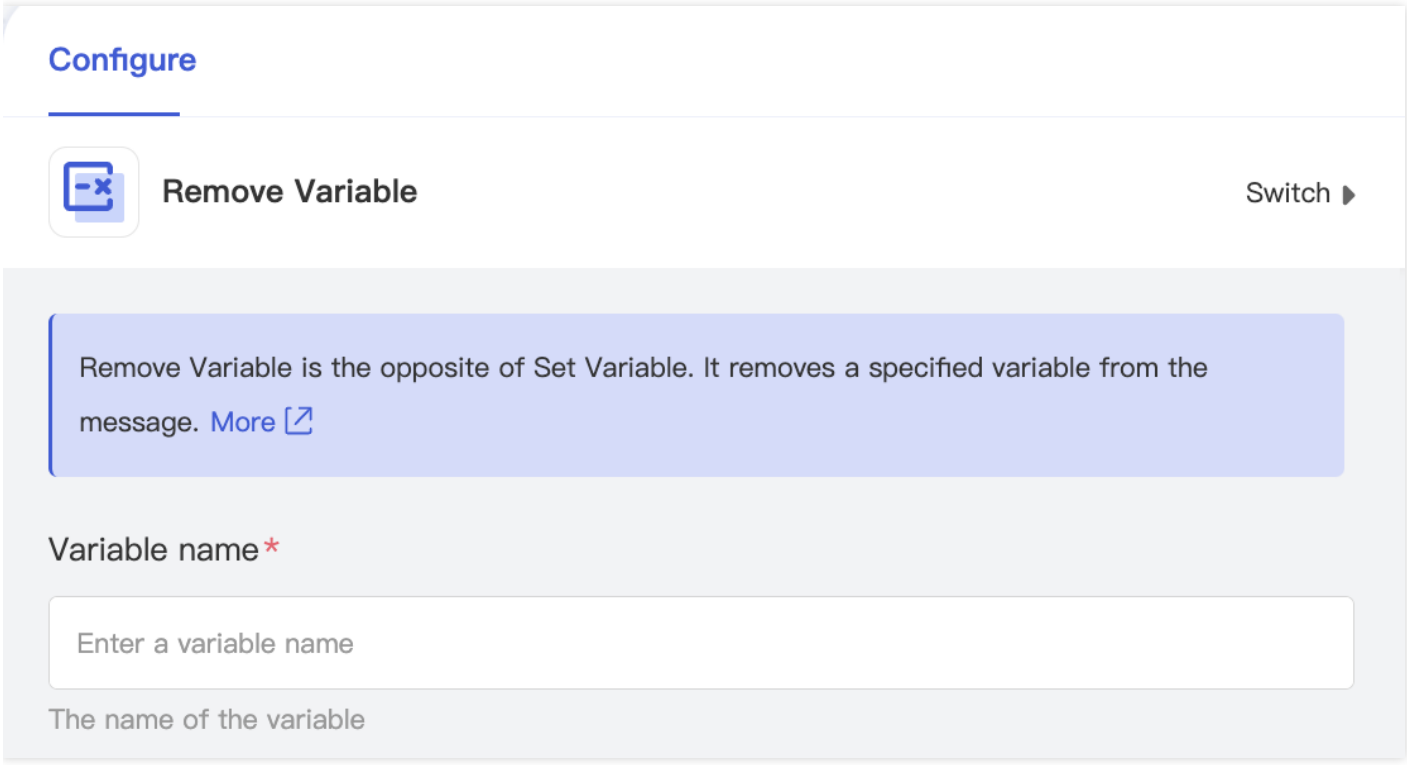
Contrary to the Set Variable component, Remove Variable is used to delete the specified variable in `message` .

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Variable name	string	Name of the variable to be removed.	Yes	None

Configuration page



Output

The output `message` no longer contains the deleted variable. The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	This attribute inherits the `payload` of the previous component.
error	`error` will be empty if the flow is executed successfully. `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute inherits the `attribute` of the previous component.
variable	The variable removed from `variable` in the previous component.

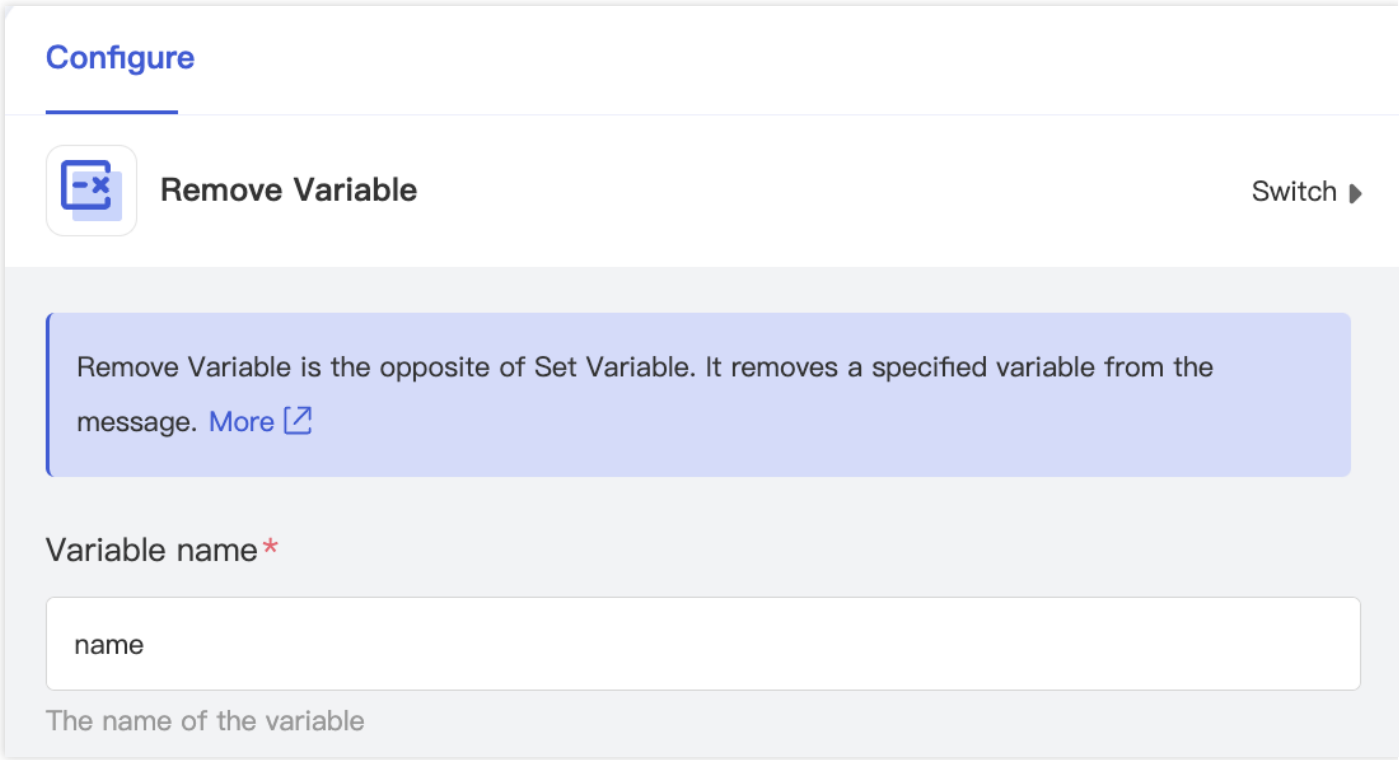
Data preview

None.

Example

1. Add a Remove Variable component.

2. Enter the name of the variable to be removed.



Flow Reference

Last updated：2023-08-03 17:12:12

Overview

Flow Reference is used to reference other flows in the current app. Unlike Async, Flow Reference is a sync process. It will continue to execute the next action only after the execution of the referenced flow. After the subflow in Flow Reference is executed, `message` will be passed to the main flow, and the next node will be executed based on the `message` of the subflow.

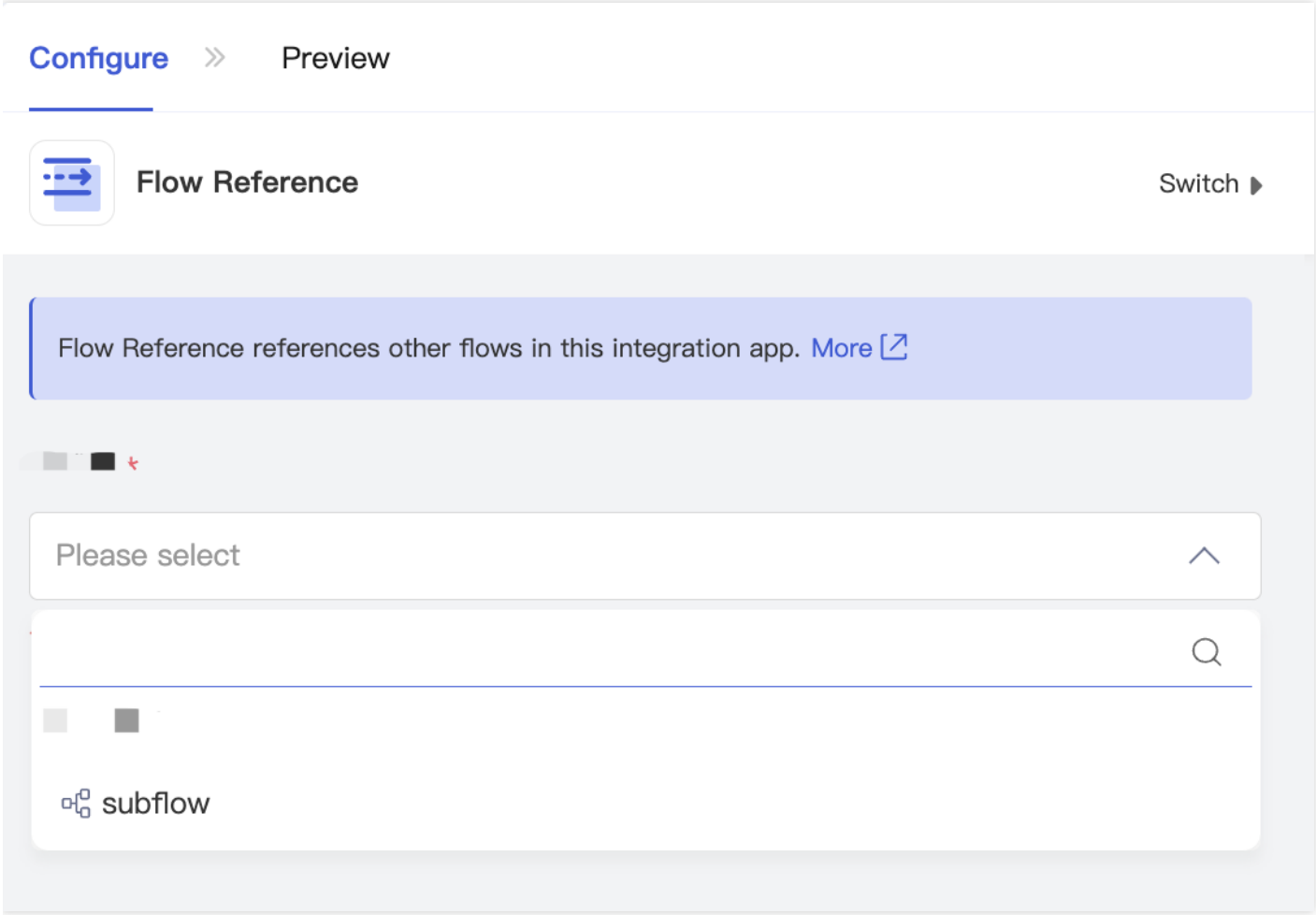
If the subflow in Flow Reference contains a trigger node, and the subflow execution is triggered by the Flow Reference component, the trigger node of the subflow won't be executed; that is, the subflow will be executed from the second node.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Flow	string	Flow name. You can select a flow shared by another app in the same project.	Yes	None

Configuration page



Data preview

None.

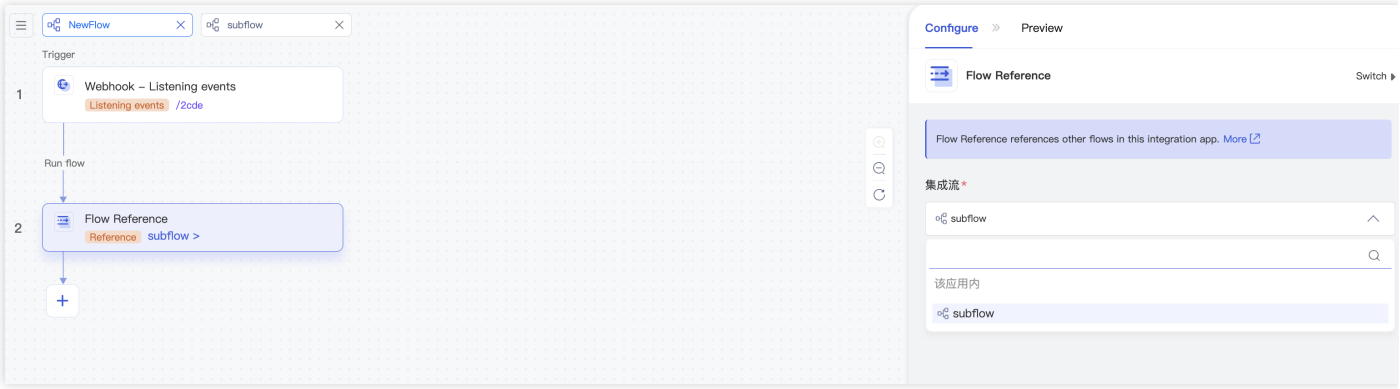
Output

`message` Attribute	Value
payload	The `payload` output by the last node of the subflow.
error	`error` will be empty if the flow is executed successfully.
attribute	The `attribute` output by the subflow.
variable	All set variables in the subflow.

Examples

Referencing a flow in the same app

1. Create a flow and name it `Subflow`. Add a **Set Payload** component to the subflow and output `payload`. Add a **Set Variable** component and set variable `a`.



2. Add a Flow Reference component and select **Subflow** in the drop-down list.

3. After the unit test is completed, click the Flow Reference component, select the Professional mode to view the output, and you can see that the `payload` and variable `a` configured in the subflow are passed to the current flow.

Referencing a flow from another app in the same project

Flow reference methods

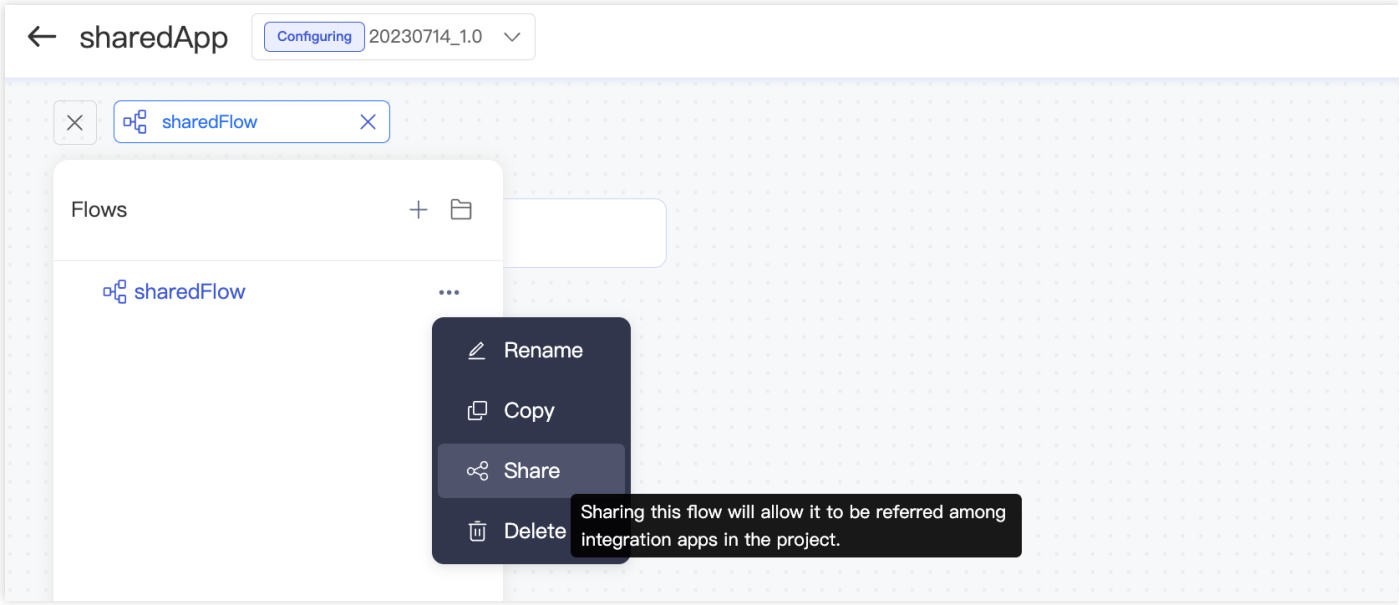
Multiple apps in the same project may use the flow of the same general feature, such as user login authentication. Generally, there are three methods for using this flow:

Method	Pros and Cons

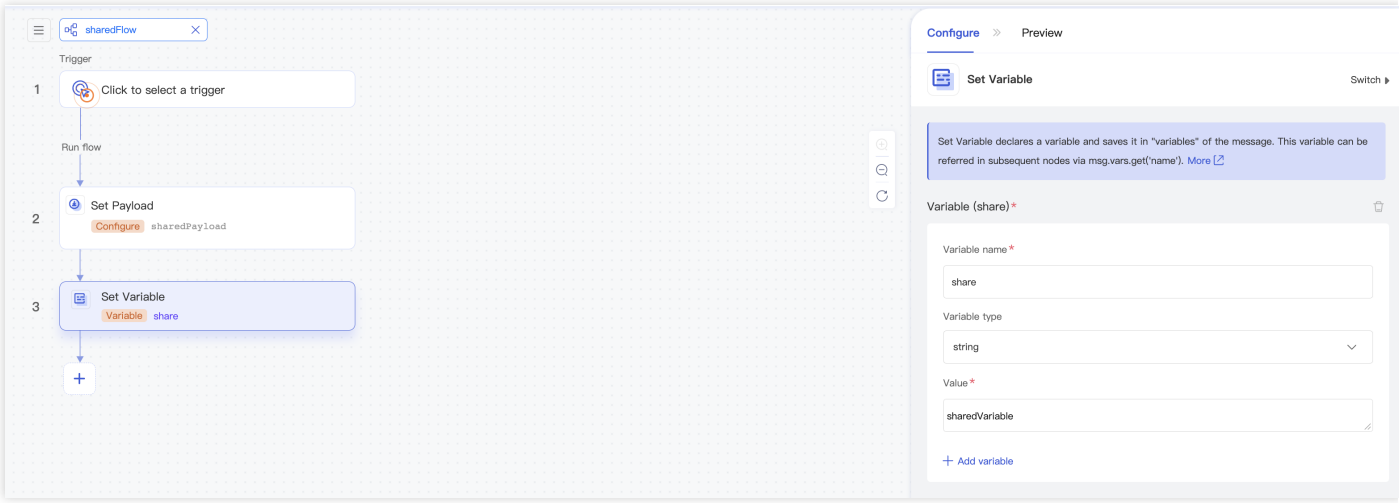
Method	Pros and Cons
Edit or copy a **User login authentication** flow in each app	There are a large number of repeated flows, which are difficult to maintain.
Call an app in other apps through a public HTTP API	HTTP calls consume traffic, and debugging and maintenance are difficult.
Use Flow Reference for cross-app flow reference	Like flow reference within an app, this method can be used when two apps run at the same time.

Using Flow Reference for cross-app flow reference

1. Create an app "Shared app" which provides a general feature flow "Shared flow" and share the flow.

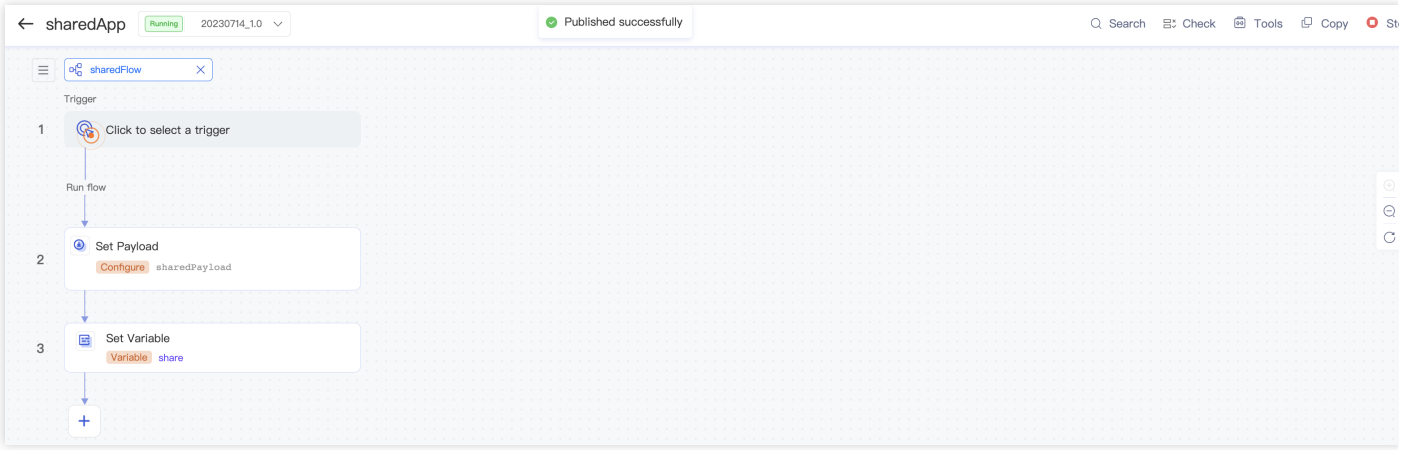


2. In the shared flow, set `payload` and variable `share`.

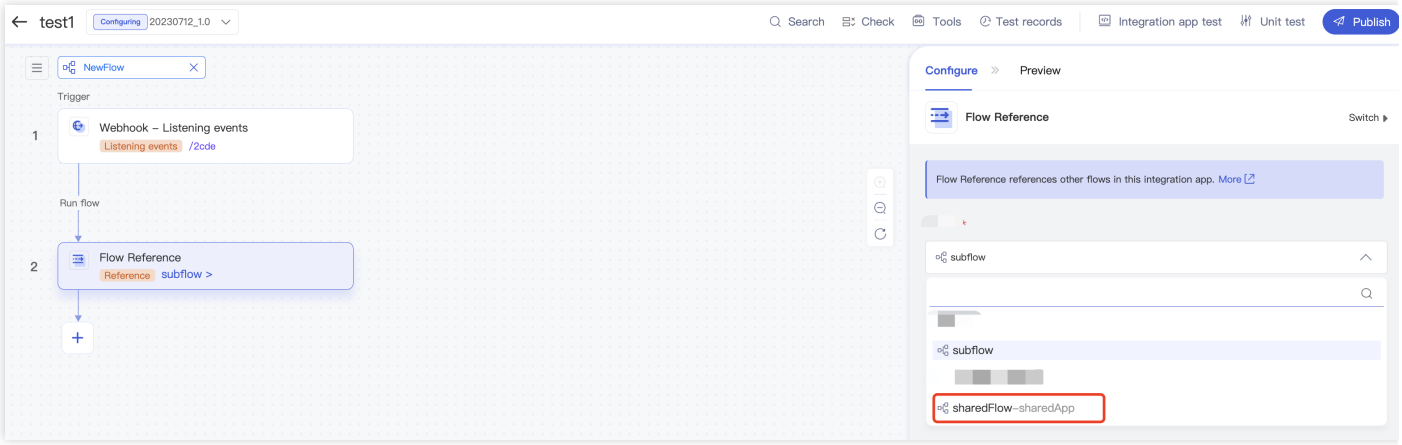


3. Publish "Shared app", and "Shared flow" can be selected and used in the Flow Reference list in other apps in the project.

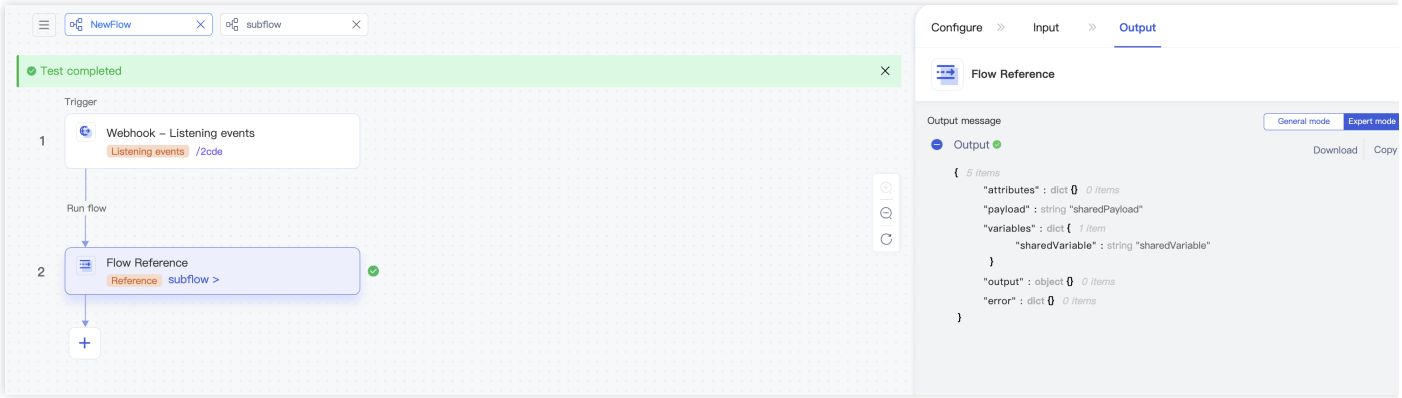
   Then, start an app test, so that a test can be performed after "Shared flow" is referenced in other apps.



4. Edit the flow in another app in the project, add a Flow Reference component, and select "Shared flow" of "Shared app" in the project to reference it.



5. Perform a unit test and view the output of the Flow Reference ("Shared flow") node. Switch to the Professional mode, and you can see that `payload` and variable `share` configured in the shared flow are passed to the current flow.



Scheduler

Last updated：2023-08-03 17:12:12

Overview

As a trigger, Scheduler is used to trigger a flow according to the configured rule at the scheduled time. The graphical Scheduler component supports three trigger modes:

- One-time trigger: The flow can be triggered at multiple specified time points.
- Regular trigger: The flow can be triggered regularly.
- Cron expression: The configuration includes one or more cron rules. When any cron rule matches the current time, the flow where the Scheduler component resides will be triggered.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Trigger mode	Int	You can select **One-time trigger**, **Regular trigger**, or **Cron expression**.	Yes	0 (Cron expression mode).
Cron expression	string	Trigger rule such as once every minute.	Yes	None
Time zone	string	Specified time zone.	Yes	Asia/Beijing UTC+08:00
Triggered only after the previous task is executed	bool	If this option is selected, the flow will be triggered only after the previous task is executed.	No	false

Scheduler contains one or multiple cron rules. To add multiple rules, separate them by `\r` . A cron expression is configured as follows:

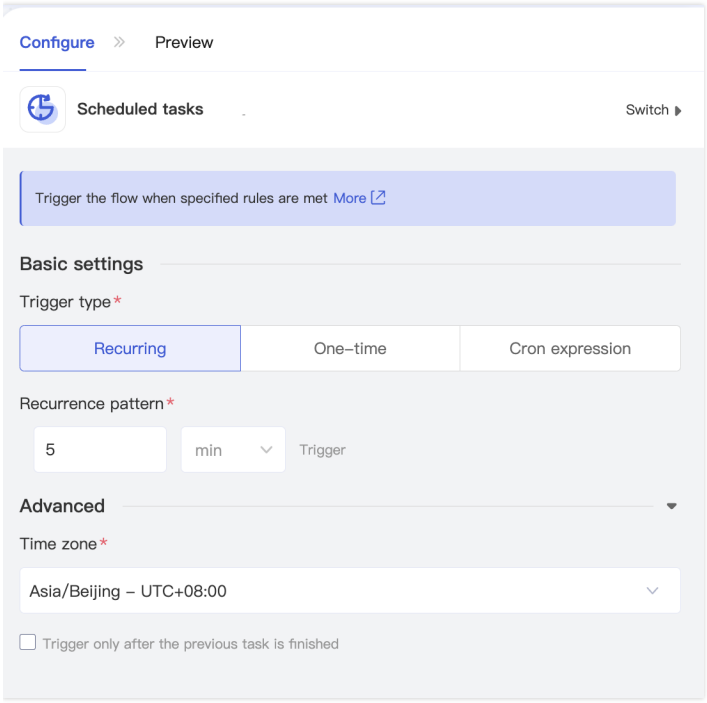
Parameter	Description	Value Range
seconds	Second	0–59

Parameter	Description	Value Range
minutes	Minute	0–59
hours	Hour	0–23
days	Date. This parameter is optional and is set to every day by default.	1–31
months	Month. This parameter is optional and is set to every month by default.	1–12
weekdays	Day of the week. This parameter is optional and is not specified by default.	1–7
years	Year. This parameter is optional and is set to every year by default.	1970–2099

You can use the following operators when configuring a cron expression:

- `*` indicates all valid values. For example, `hours="*"` indicates every hour.
- `-` indicates a range. For example, `weekdays="1-5"` indicates Monday to Friday.
- `,` indicates enumeration. For example, `months="1,3,5,7,8,10,12"` indicates all months with 31 days.
- `/` indicates increment. For example, `hours="8/2"` indicates every two hours from 08:00.
- `L` indicates the last period. For example, `weekdays="6L"` indicates the last Saturday of the current month.
- `?` indicates an unspecified value. There is a restraint that at least one of the parameters year, month, date, and day of the week must be left unspecified to avoid a conflict; for example, both February 20, 2020 (which should be Thursday) and Wednesday are specified. The day of the week is unspecified by default.

Configuration page



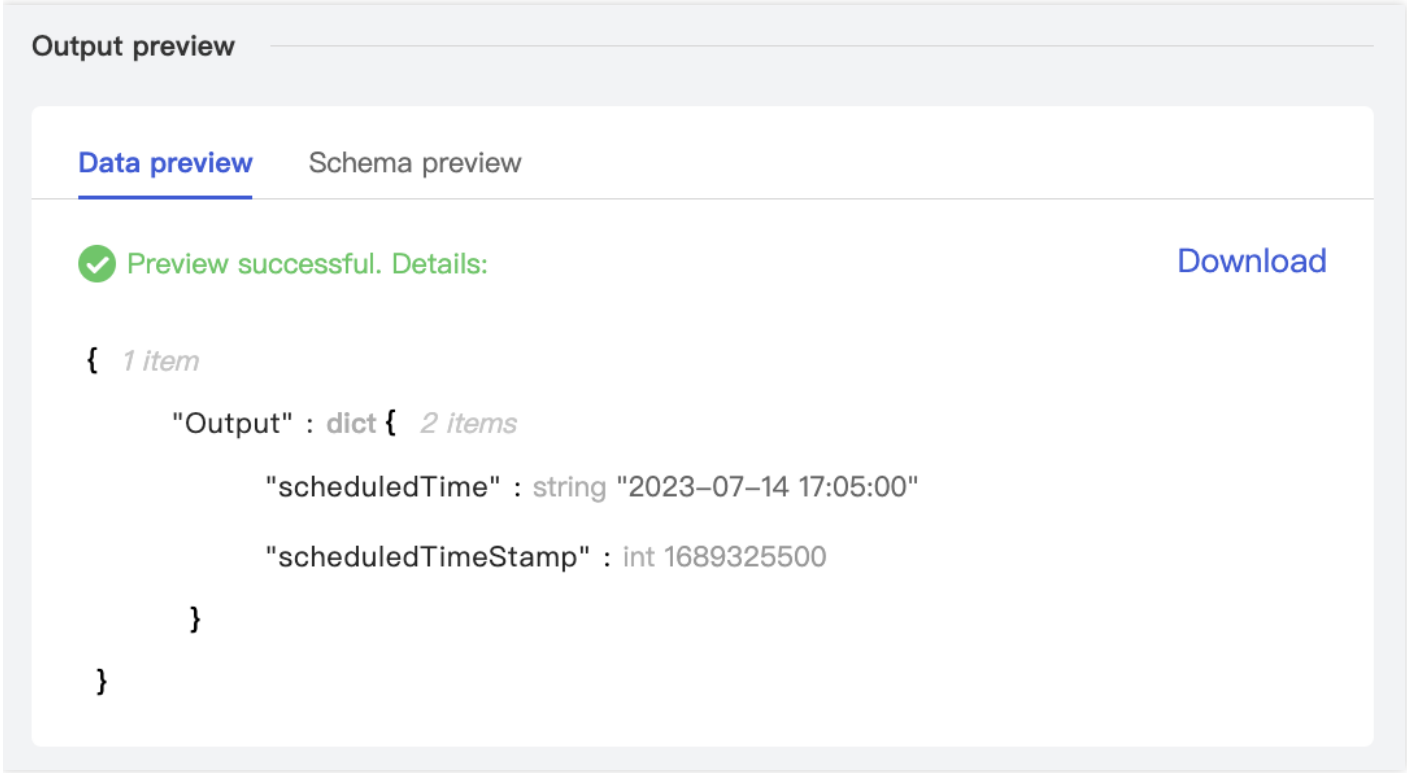
Output

As a trigger component, Scheduler is the first component in a flow. It will generate an empty `message` to trigger the flow execution.

The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	Null.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	Null.
variable	Null.

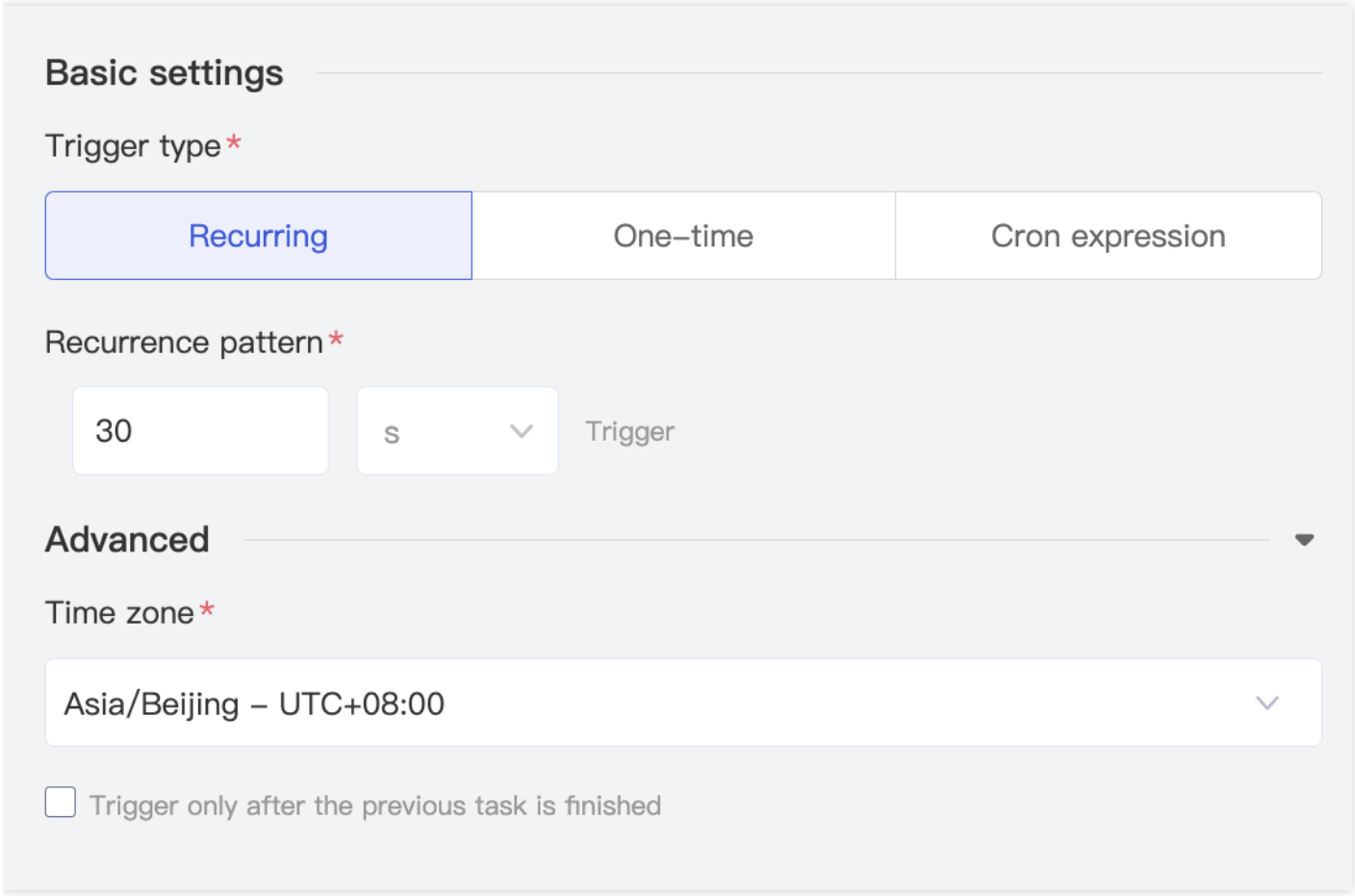
Data preview



Examples

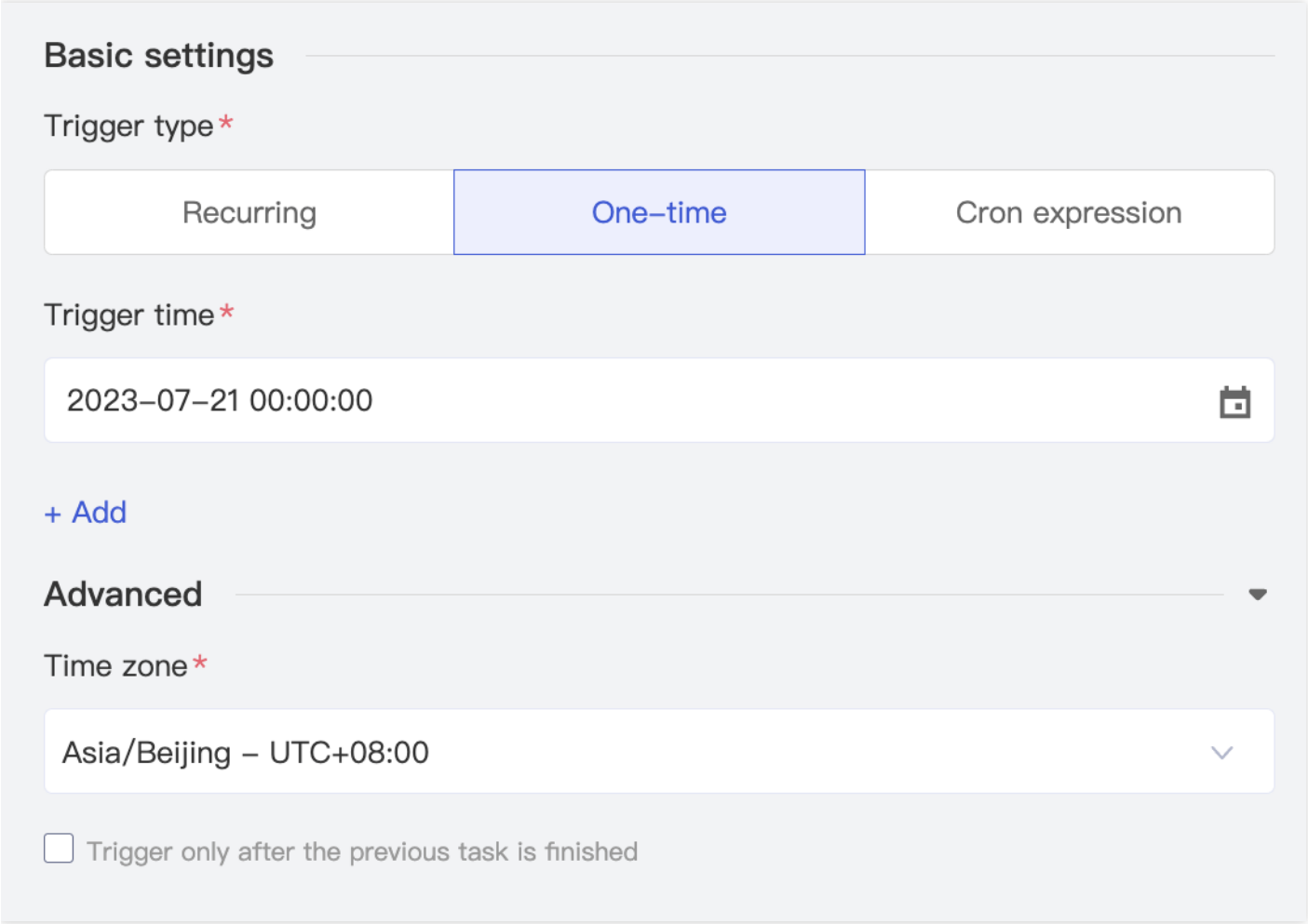
Regular trigger mode

Trigger the flow once every 30 seconds:



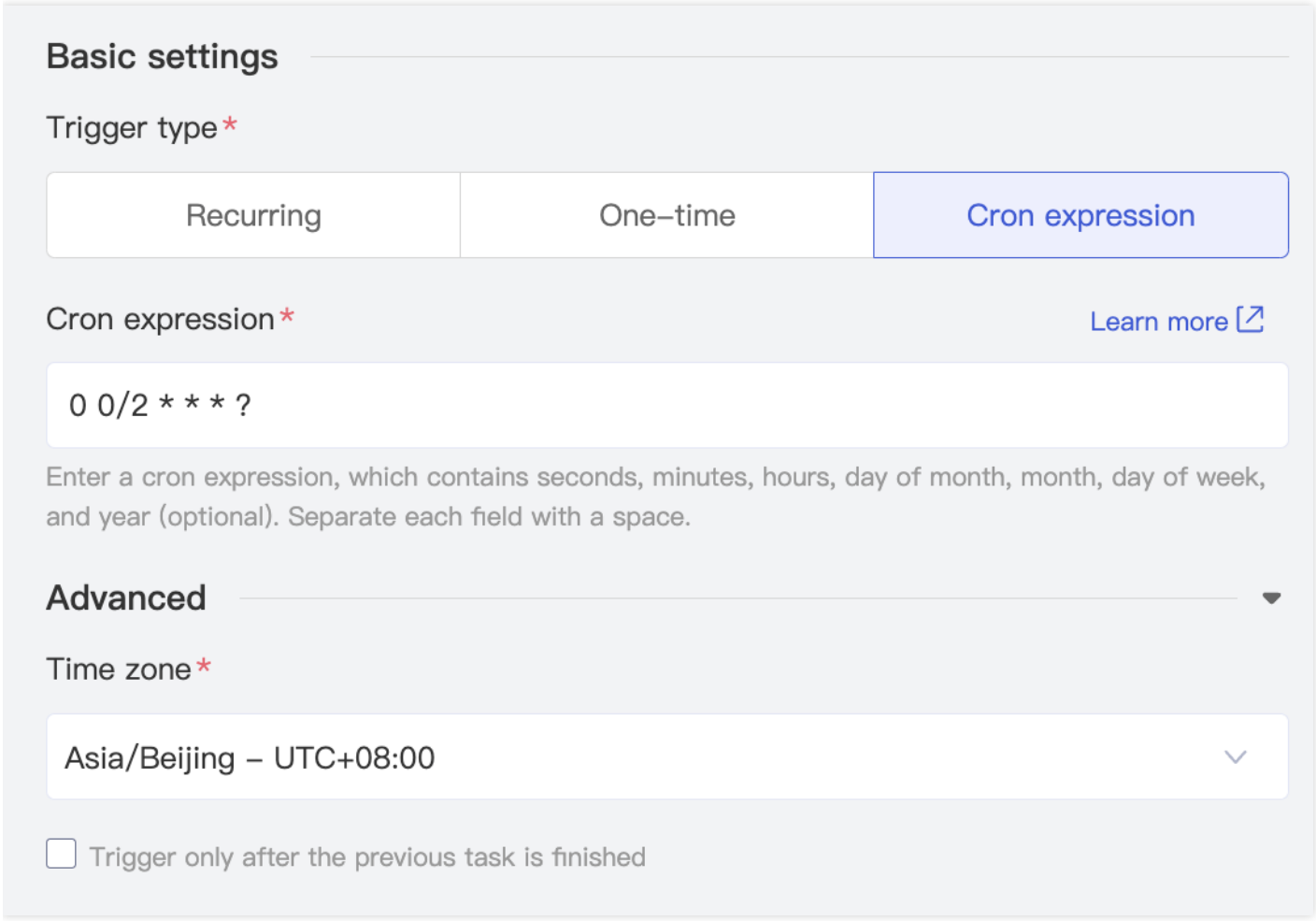
One-time trigger mode

Trigger the flow once at 00:00:00 on January 1, 2023:



Cron expression mode

Trigger the flow once every five minutes:



Scatter Gather

Last updated：2023-08-03 17:12:12

Overview

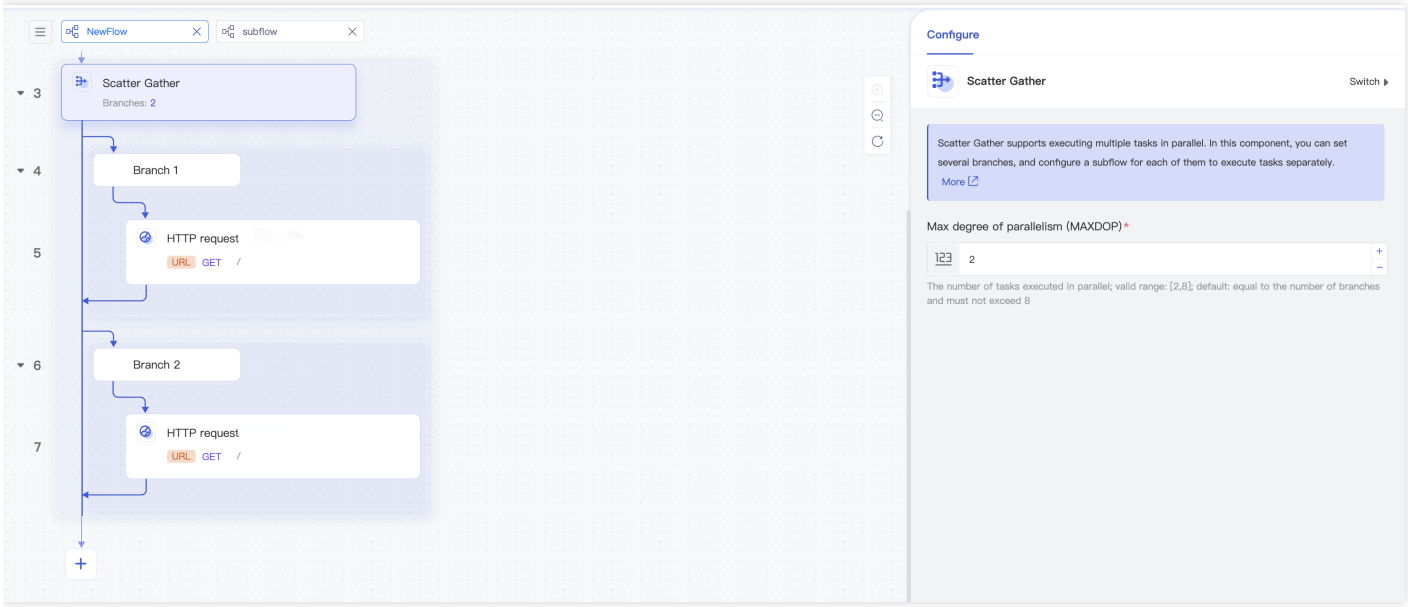
Scatter Gather can execute multiple tasks in parallel. In this component, you can add multiple branches and configure a subflow in each branch to execute a task independently.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Maximum parallelism	int	Maximum number of tasks executed in parallel. Value range: 2–8. The actual parallelism is the lesser value between the number of branches and the maximum parallelism.	Yes	4
Root message	string	The root message is a variable, which stores the `message` of the main flow. You need to enter a variable name. You can enter `msg.vars.get('#root message name#').payload` to access the `payload` data of the main flow. If the default value `rootMessage` is used, you can use `msg.vars.get('rootMessage').payload` to access the `payload` data of the main flow in the subflow.	Yes	rootMessage

Configuration page



Data preview

None.

`message` input to the subflow

`message` Attribute	Value
payload	This attribute inherits the `payload` in `message` of the main flow.
error	Null.
attribute	This attribute inherits the `attribute` in `message` of the main flow.
variable	This attribute inherits the variable of the main flow.

Output

The output result of Scatter Gather doesn't contain the `variable` variable used in the processing logic but only the data in `payload`.

The output `payload` is of `dict` type and aggregates the processing result of each branch.

The `message` output by the component is as detailed below:

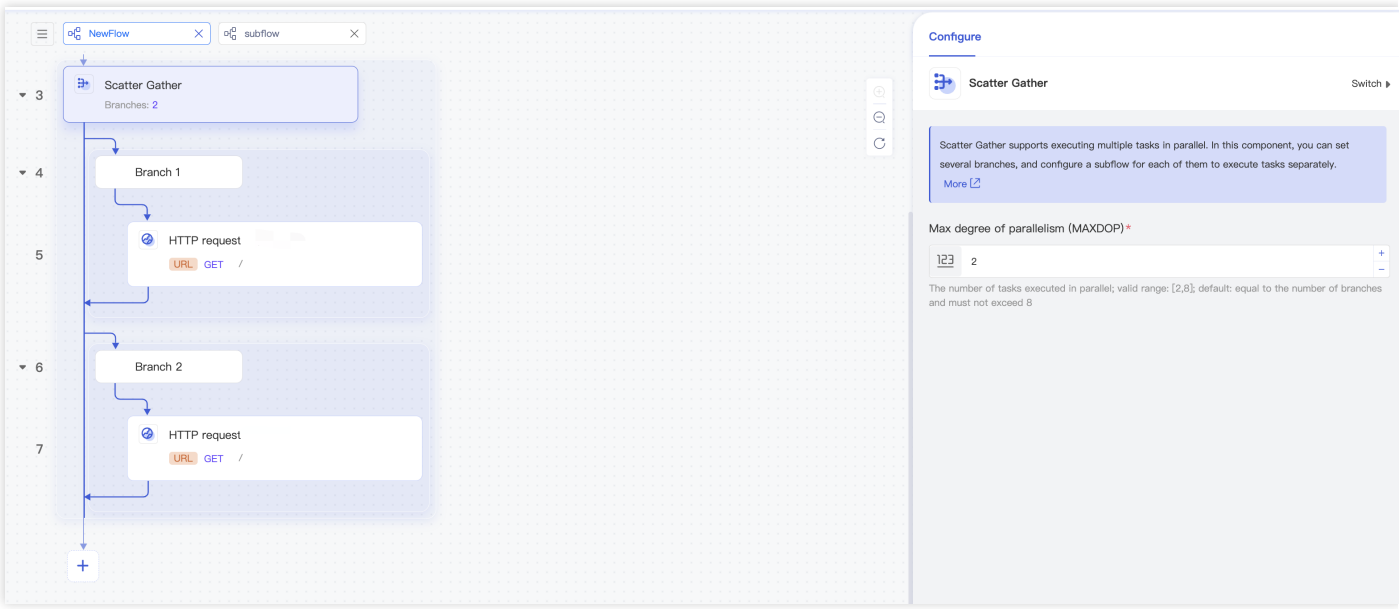
`message` Attribute	Value

`message` Attribute	Value
payload	This attribute is of `dict` type. `key` is the branch number, which starts from `1`. `value` is the branch execution result (the `payload` output by the last component).
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	The value is the same as that of the input `attribute`.
variable	The value is the same as that of the input variable.

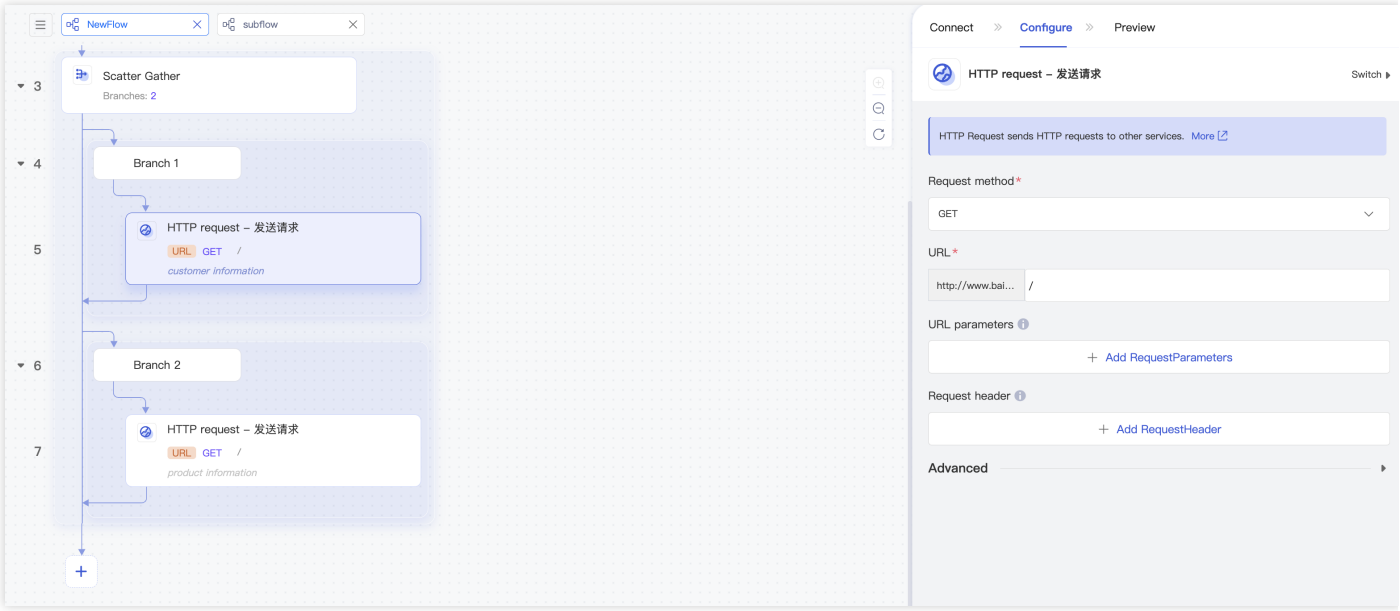
Example

We recommend you use Scatter Gather to execute different tasks in parallel. For example, if you need to query the customer and product information based on the user order data, you can configure two branches to query the two types of information respectively.

1. Add a Scatter Gather component and two branches and use the default configuration.

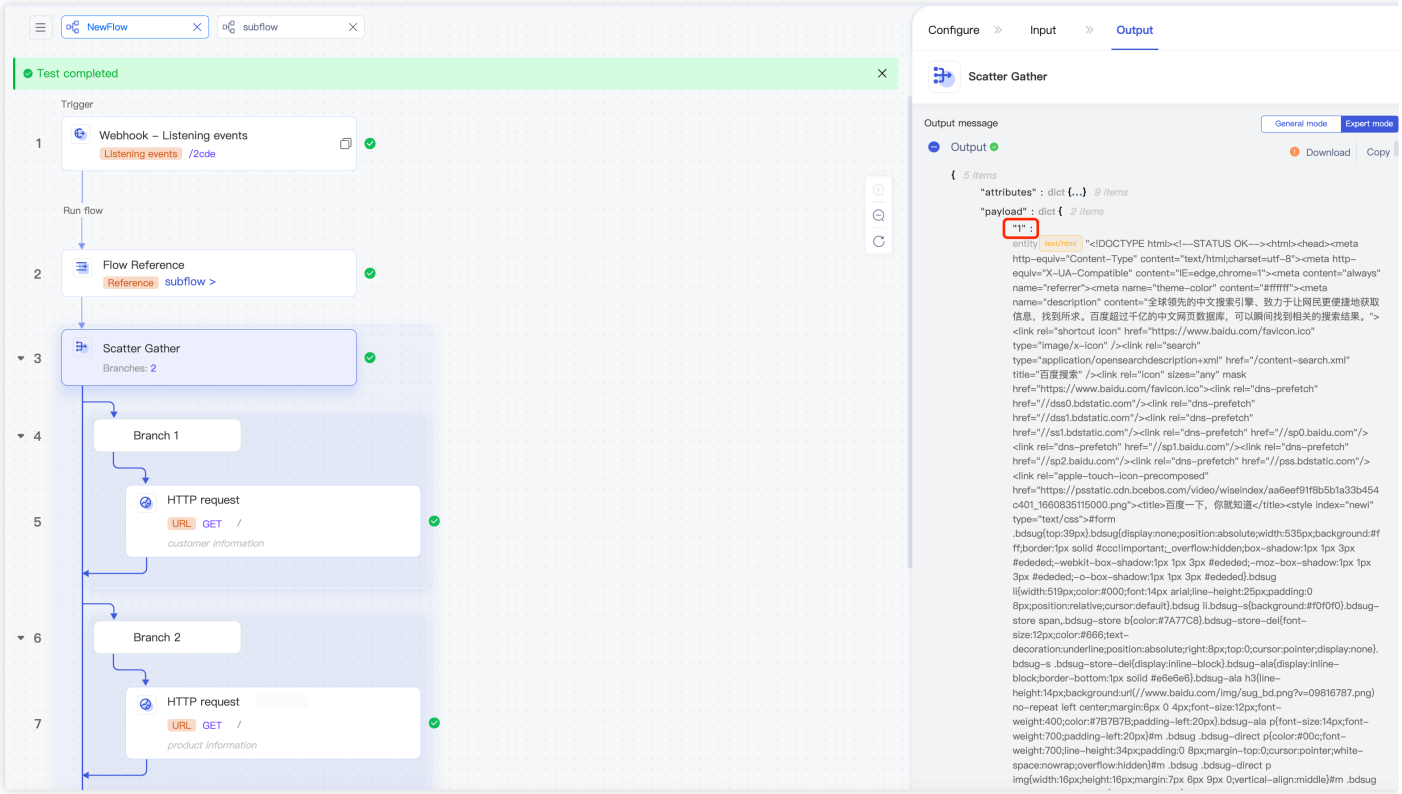


2. Configure customer information query in the first branch and product information query in the second branch. Here, two simple HTTP requests are used for simulation.



3. After execution, view the output of Scatter Gather. Switch to the Professional mode, and you can see that `payload` is a dictionary containing two keys.

   Key `1` represents the result of the first branch, i.e., the queried customer information, and key `2` represents the result of the second branch, i.e., the queried product information.



Parallel Foreach

Last updated：2023-08-03 17:12:12

Overview

Parallel Foreach is used to execute tasks in parallel. It executes the same processing logic on all elements in a data set in parallel. The actual parallelism is the lesser value between the number of remaining elements and the configured maximum parallelism.

A Parallel Foreach subflow has read-only access to the variables in the main flow and the output of other components, and modifications performed by the subflow will not affect the main flow.

After processing, the result of each element will be output to `payload` in `message` in the original sequence. Parallel Foreach is generally used in batch data processing scenarios, such as batch query and batch data import.

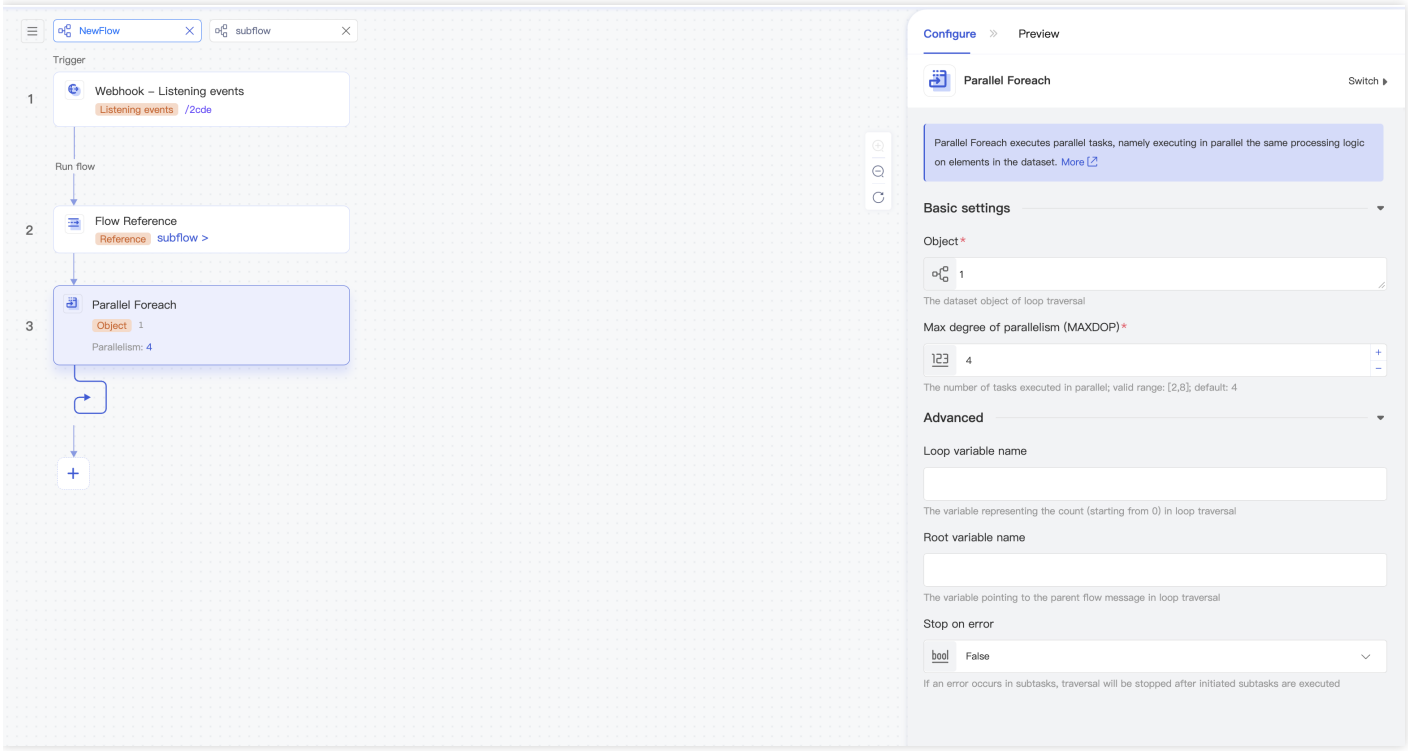
Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Data set	string, list, dict, and int	The data set to be traversed. • If the data type is `string`, all characters in the string will be traversed. • If the data type is `list`, all elements in the list will be traversed. • If the data type is `dict`, all values in the dictionary will be traversed. • If the data type is `int`, for example, if the data set is `3`, the data set `[0,1,2]` will be traversed.	Yes	None
Maximum parallelism	int	The maximum number of tasks executed in parallel. Value range: 2–8.	Yes	4

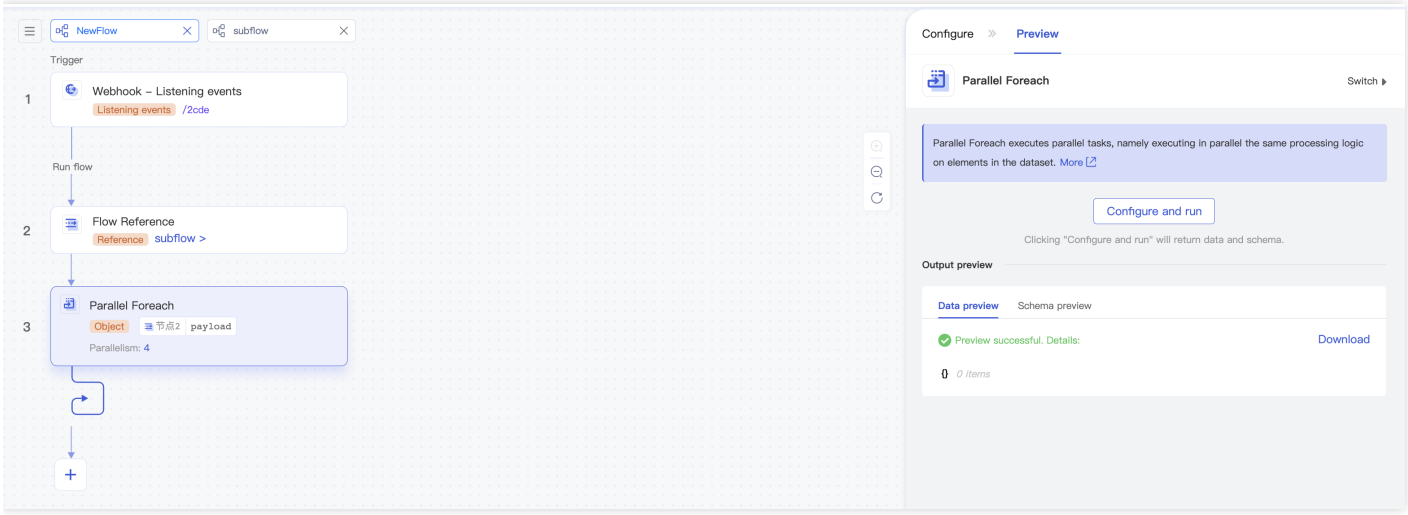
Parameter	Data Type	Description	Required	Default Value
Counter	string	The counter is a variable, which stores the current number of iterations and starts from `0`. You need to enter a variable name such as `msg.vars.get('#counter variable#')` to use the counter. For example, if the counter variable is set to the default value `counter`, in the first loop, `msg.vars.get('counter')` will be `0`, and in the second loop, it will be `1`.	No	counter
Root message	string	The root message is also a variable, which stores the `message` of the main flow. You need to enter a variable name. You can enter `msg.vars.get('#root message name#').payload` to access the `payload` data of the main flow. If the default value `rootMessage` is used, you can use `msg.vars.get('rootMessage').payload` to access the `payload` data of the main flow in the Parallel Foreach subflow.	No	rootMessage
Stop while error occurred	bool	If a subtask throws an error, traversal will stop after the execution of the initiated subtask is completed.	No	False

Configuration page

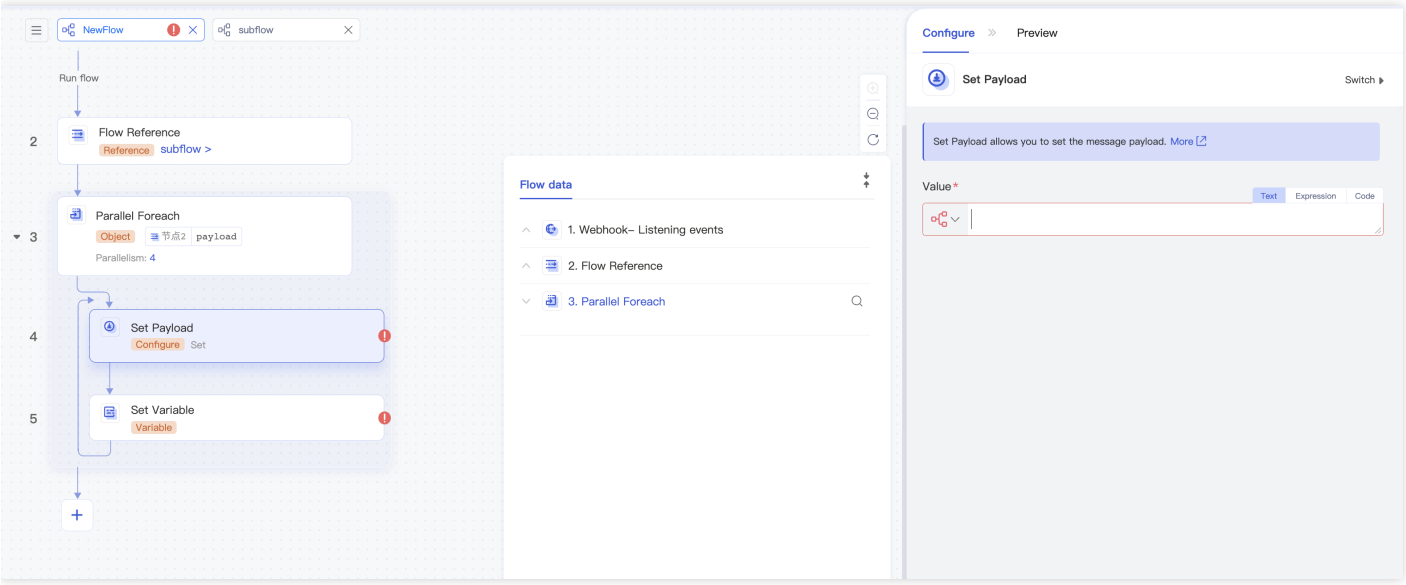


Data preview

Preview Field	Data Type	Description
payload	any	Input value in each traversal, which is also an element in the data set.
index	int	Position in each traversal. This field represents the subscript position of the current input value in the data set, which starts from `0` .



The data preview content is visible only in the subflow. Components in the subflow can directly use `payload` and `index` in the Parallel Foreach component as shown below:



`message` input to the subflow

`message` Attribute	Value
payload	An element in the data set. For example, if the data set to be iterated is `[1,2,3]`, in the first loop, `payload` in the subflow will be `1`, and in the second loop, it will be `2`. If the data set to be iterated is `{"key":"key1", "value":"value1"}` of `dict` type, in the first loop, `payload` in the subflow will be `value1`, and in the second loop, it will be `value2`.
error	Null.
attribute	Null.
variable	This attribute inherits the `variable` of the main flow and has two new variables: counter and root message. If you use the default values of the two new variables, you can use expressions `msg.vars.get('counter')` and `msg.vars.get('rootMessage')` to access them. If Set Variable is used in For Each, the new variables will be added to `variable` during subflow execution.

Output

The output result of Parallel Foreach doesn't contain the `variable` variable used in the processing logic but only the data in `payload`. The output `payload` is a variable of `list` type, which contains the iteration result of each element in the raw data set in the original sequence. `attribute` inherits the value of the previous component.

The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	This attribute is of `list` type and contains the processing result of each element in the input sequence in the raw data set.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute inherits the `attribute` of the component previous to Parallel Foreach.
variable	This attribute inherits the `variable` of the component previous to Parallel Foreach.

Example

The following describes how to use the Parallel Foreach component to multiply all elements in the list by 2. The raw data set is `[1,2,3,4]`.

1. Add a Parallel Foreach component, configure the data set `[1,2,3,4]`, and set the maximum parallelism to `3`.
2. Add a Set Payload component to Parallel Foreach to multiply the elements in `payload` in the subflow by 2.
3. Output the result.

Sleep

Last updated：2023-08-03 17:12:12

Overview

The Sleep component is used to execute a flow after the specified period of time.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Delay	int	The specified delay in milliseconds.	Yes	1000

Output

The `message` output by the component is as detailed below:

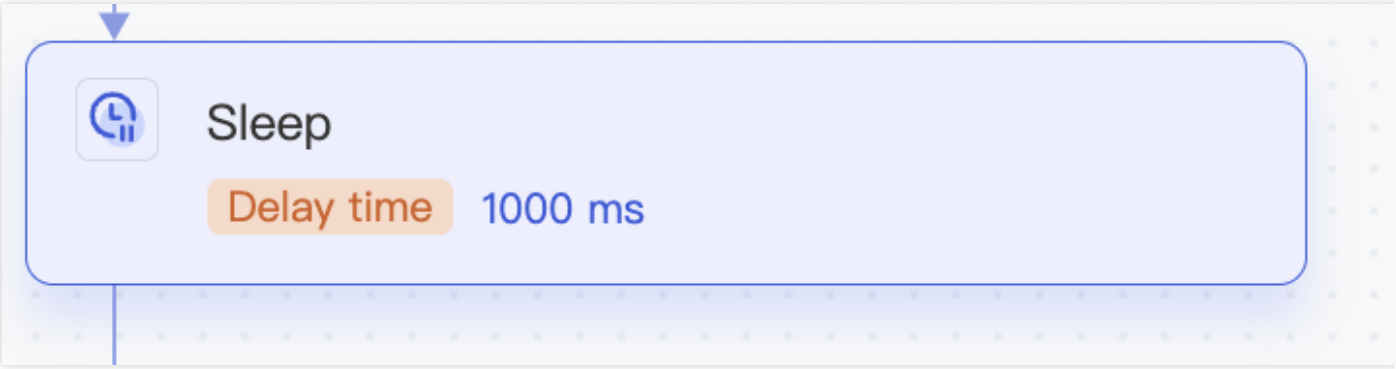
`message` Attribute	Value
payload	This attribute inherits the `attribute` of the previous component.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute inherits the `attribute` of the previous component.
variable	This attribute inherits the `variable` of the previous component.

Data preview

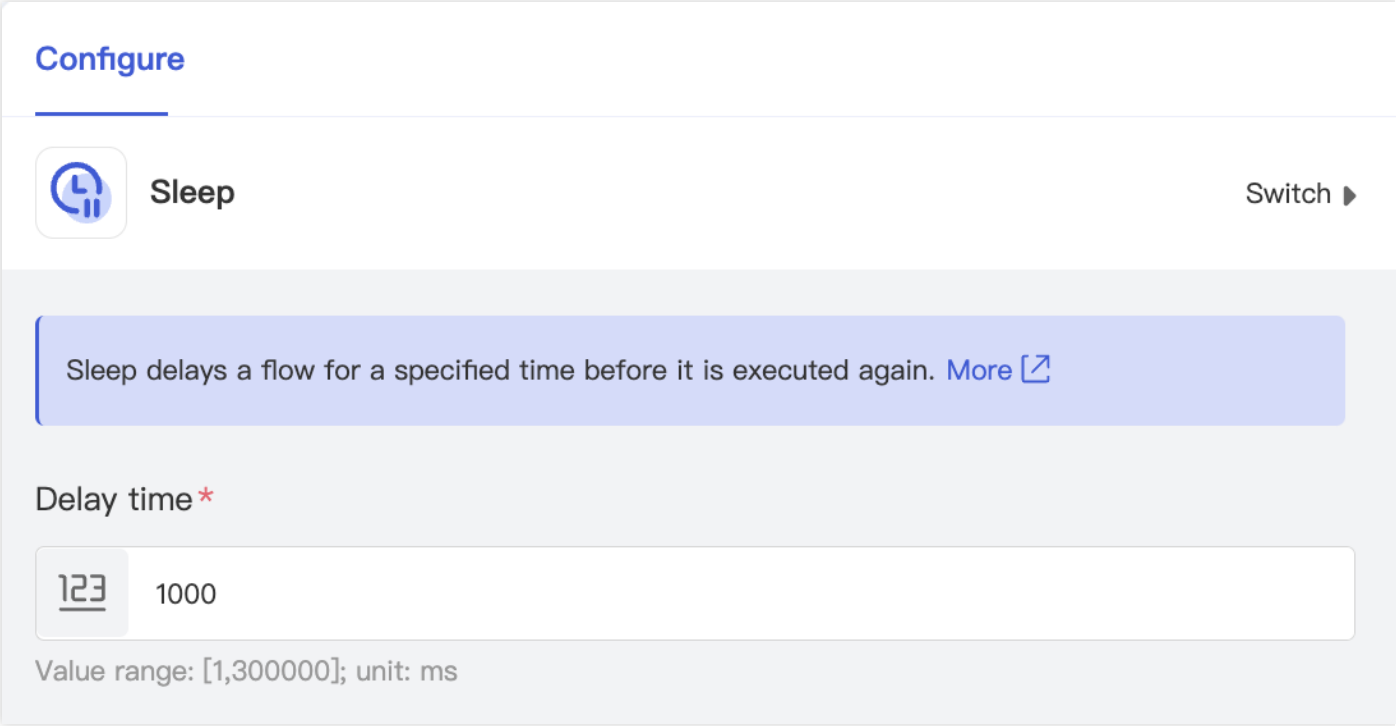
None.

Example

1. Add a Sleep component.



2. Enter the delay.



Raise Error

Last updated：2023-08-03 17:12:12

Overview

Raise Error is used to throw exceptions and stop flow execution. This component can be used alone or together with the Try-Catch component. When it is used alone and is hit, the flow will stop execution and return an error message. When it is used together with Try-Catch, Try-Catch can capture the exception defined in it and execute the subflow configured in Try-Catch.

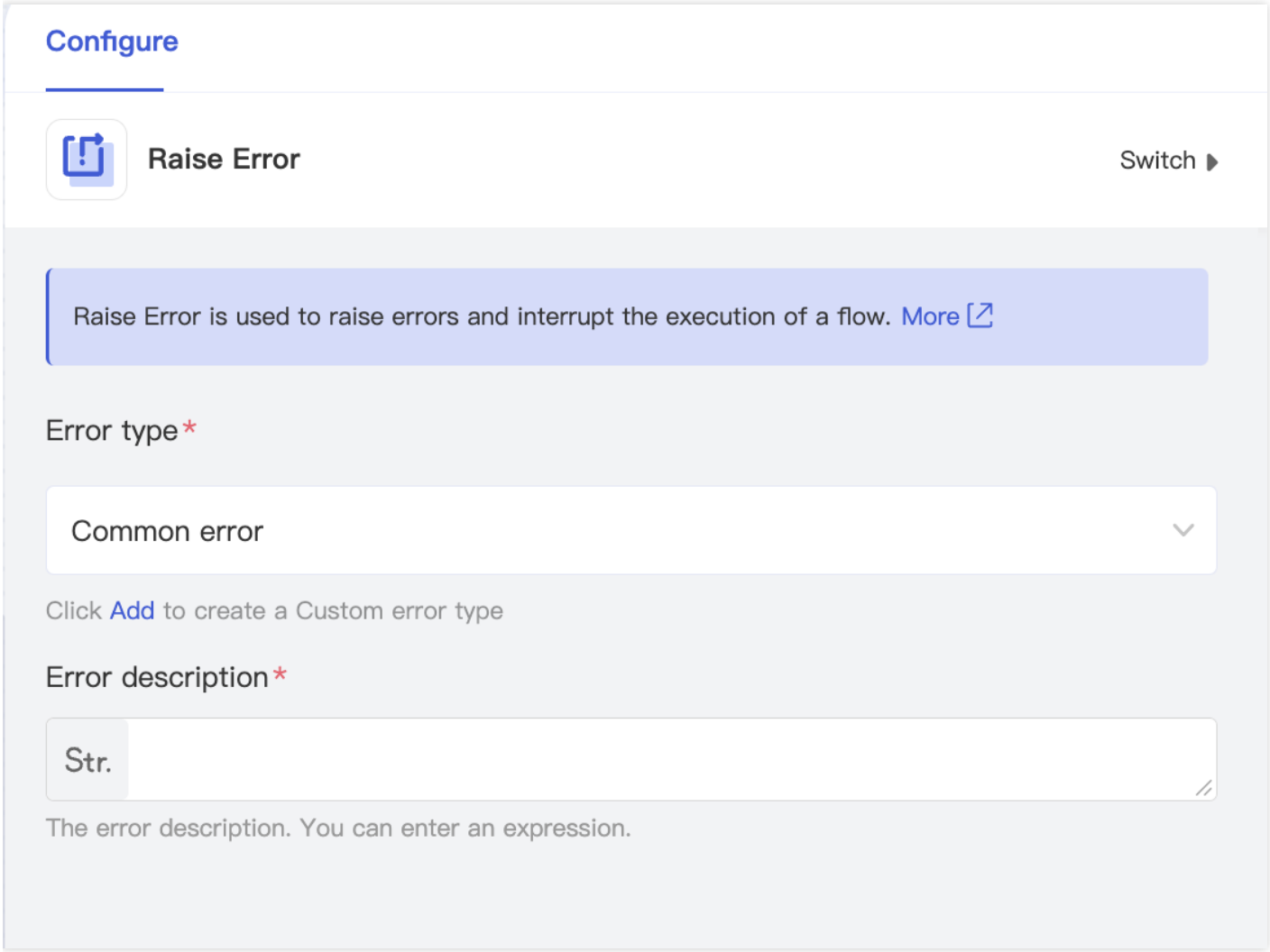
Operation Configuration

Connection description

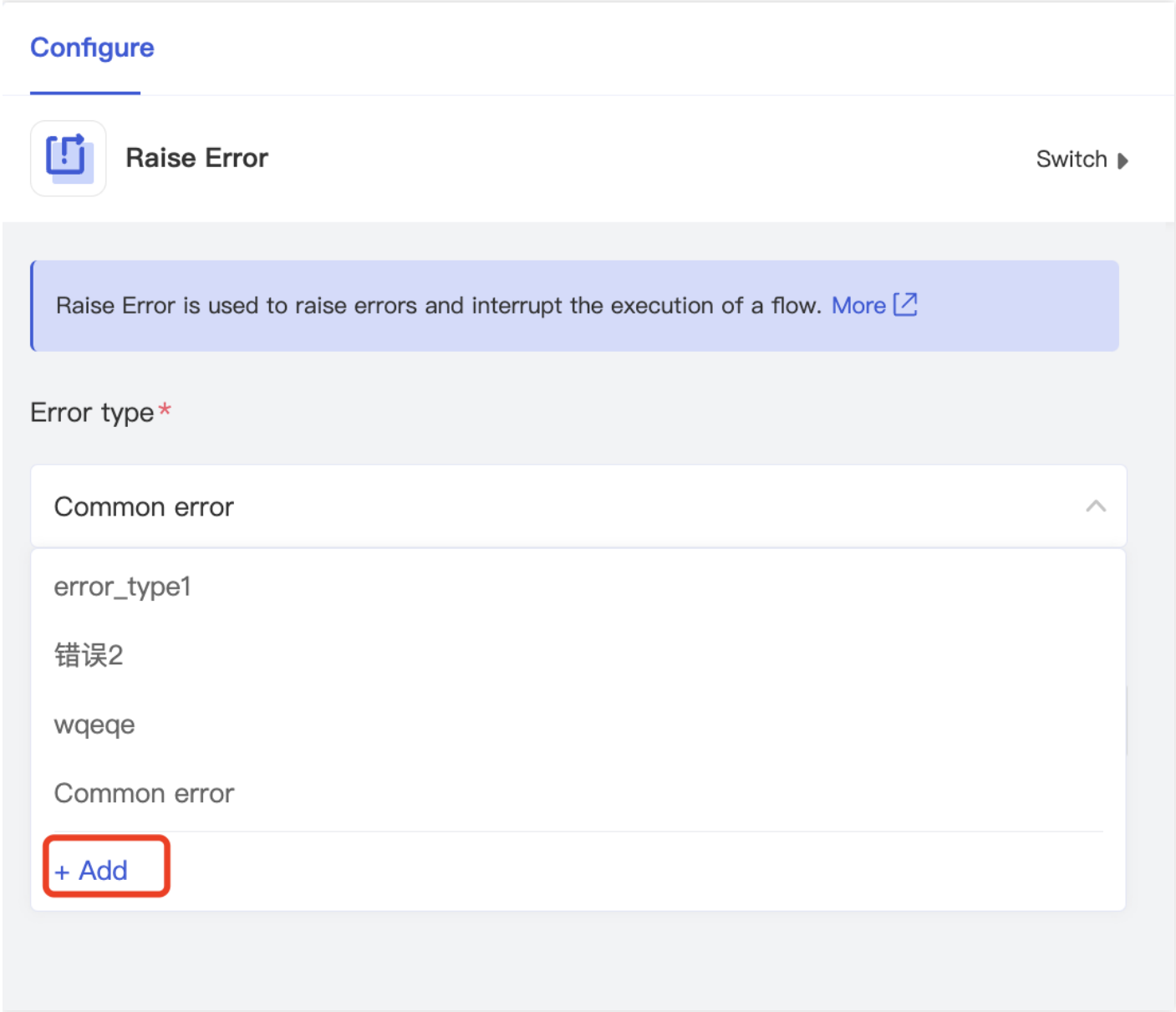
None.

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Error type	string	Custom error type.	Yes	General error
Error description	string	Error description.	Yes	None



You can select an error type in the drop-down list and click **Add** to add a new error type. Error types are visible in all apps in the project.



Data preview

None.

Output

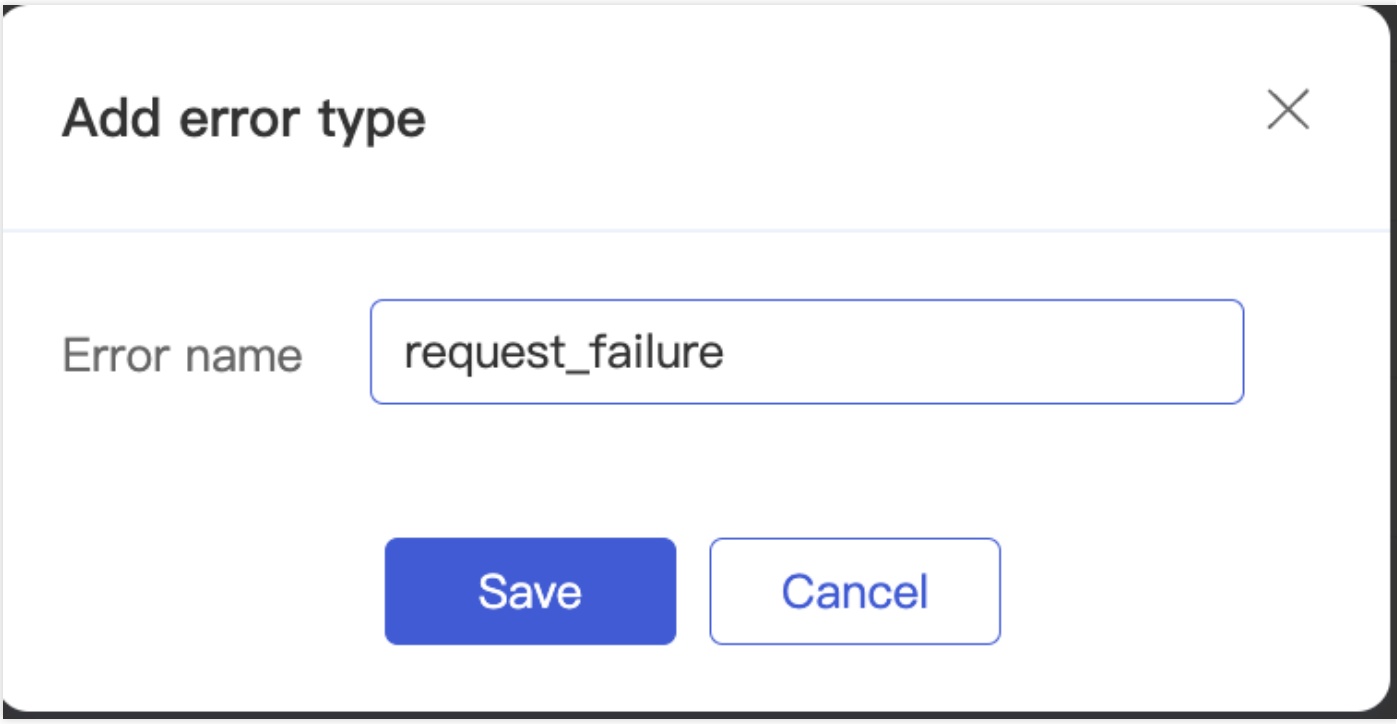
The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	This attribute inherits the `payload` of the previous component.

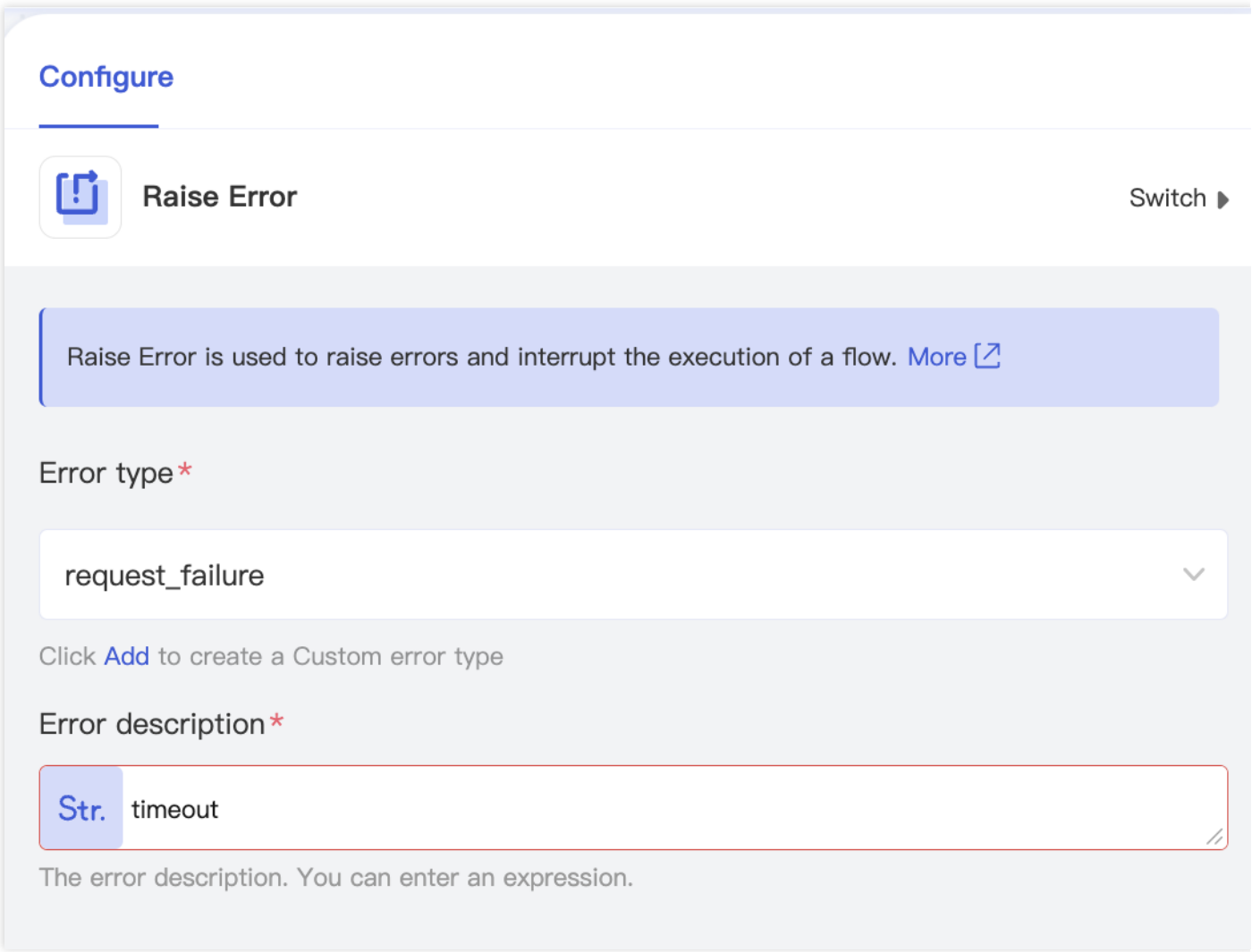
`message` Attribute	Value
error	`error` is of `dict` type and contains the `Code` and `Description` fields. The `Code` field indicates the error type and is represented by an internal code, and the `Description` field indicates the error description.
attribute	This attribute inherits the `attribute` of the previous component.
variable	This attribute inherits the `variable` of the previous component.

Example

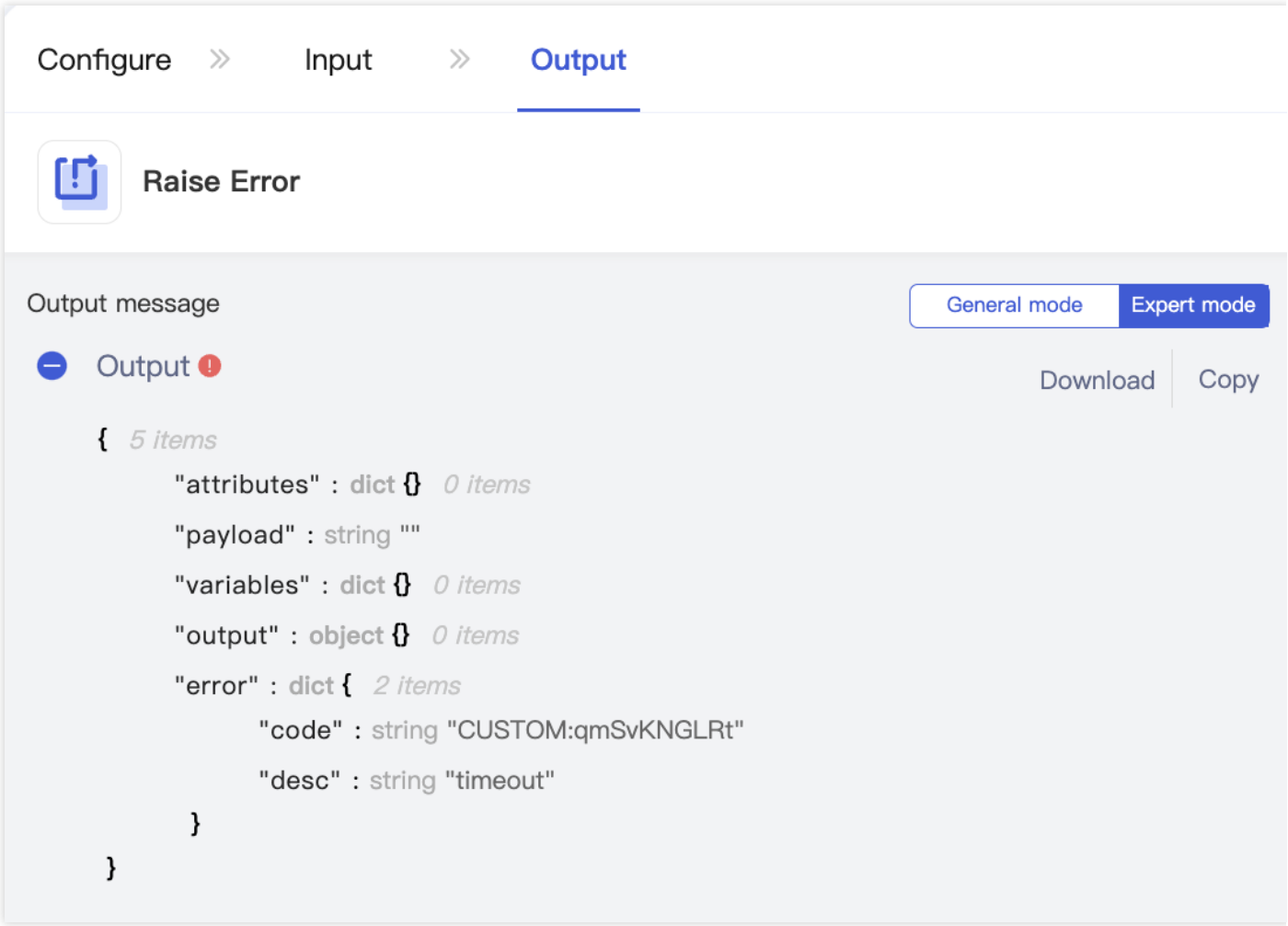
1. Add a Raise Error component.
2. Add the error type **Request failure**.



3. Select the error type and configure the error description.



4. Output the message.



Transform

Last updated：2023-08-03 17:12:12

Overview

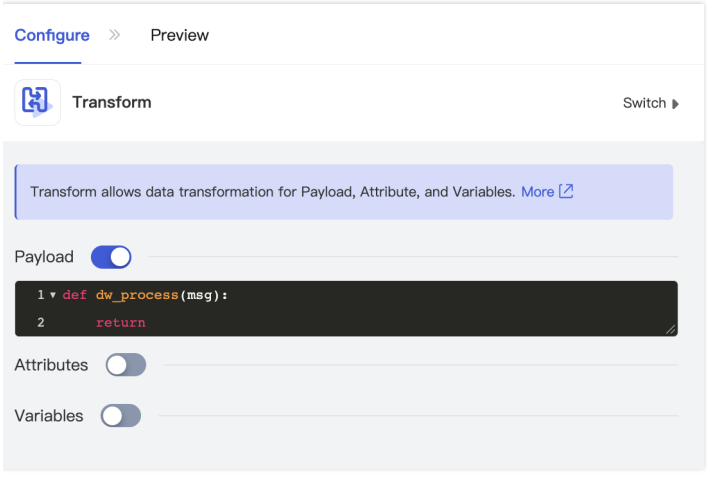
Transform is used to orchestrate and convert the format of the data in `message`. It can modify `payload`, `attribute`, and `variable`.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
payload	any	The configured payload.	No	None
attribute	dict	The configured attribute.	No	None
variable	dict	The configured variable.	Yes	None

Configuration page



Output

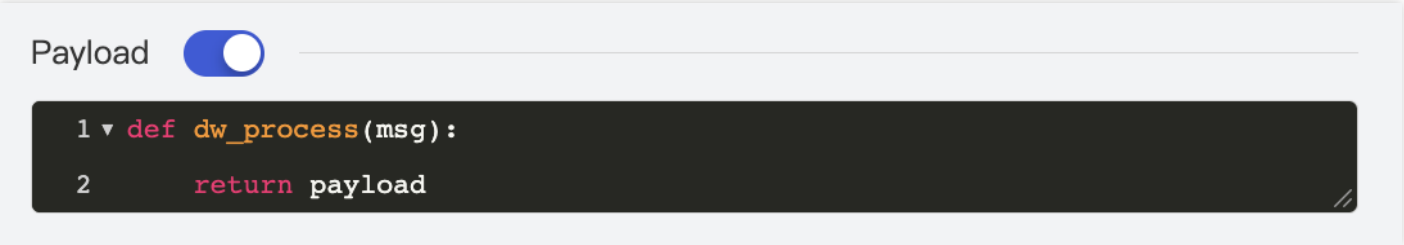
The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	If `payload` is added to **Output**, the execution result in `payload` will be output; otherwise, the `payload` of the previous component will be inherited.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute is of `dict` type. If `attributes` is added to **Output**, the execution result in `attributes` will be output; otherwise, the `attribute` of the previous component will be inherited.
variable	If `variables` is added to **Output**, the `variable` of the previous component and the new `variable` added in the Transform component will be output together; otherwise, the `variable` of the previous component will be inherited.

Examples

Setting `payload`

Add `payload` .



Setting `attribute`

Add and edit `attributes` . As `attributes` is of `dict` type, the expression output also must be of `dict` type.

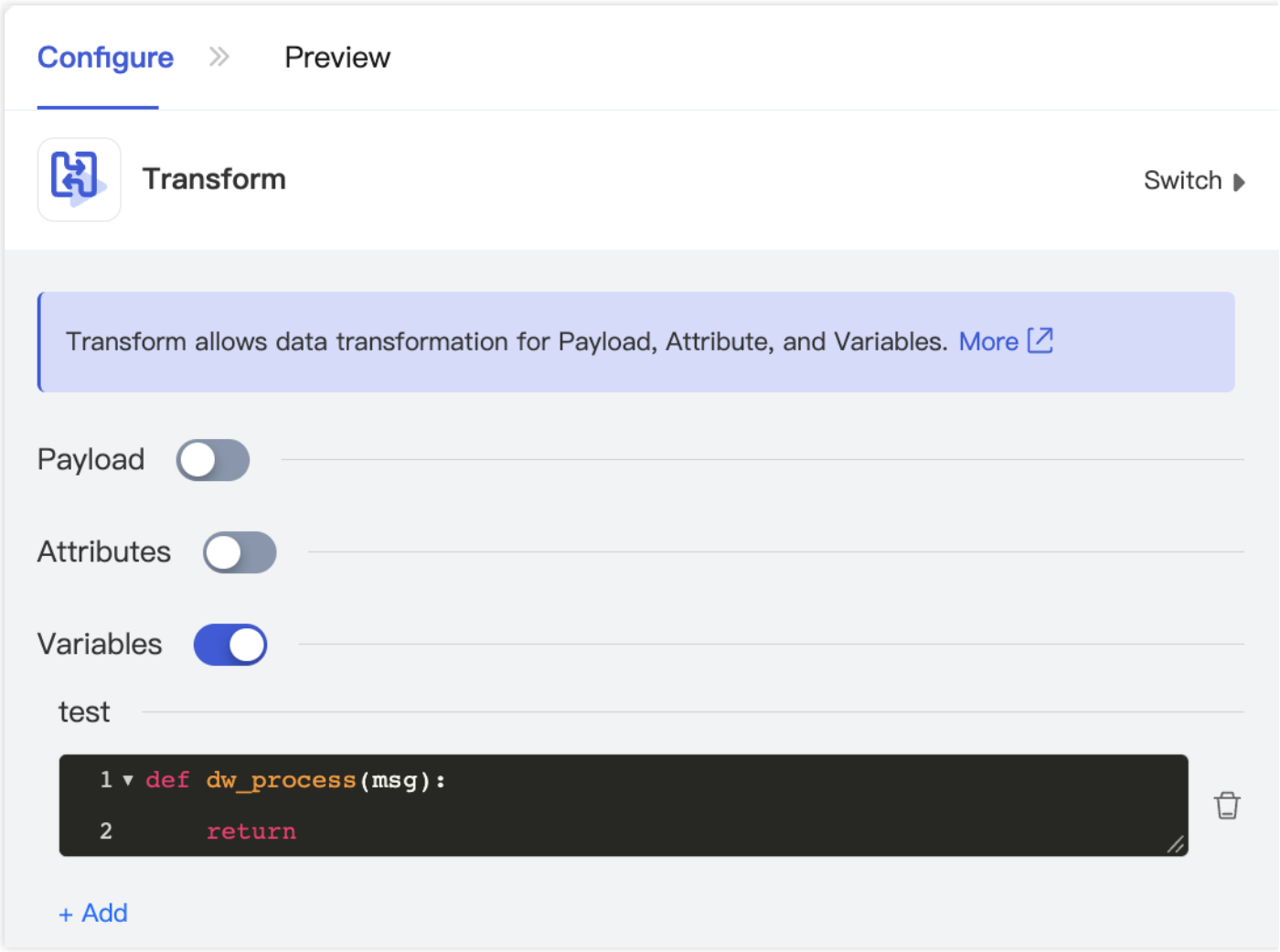
Attributes

```
def dw_process(msg):
    return {"company":"tencent"}
```

Setting `variable`

1. Add `variables` . Enter the name of the variable to be declared for **Variable name**.

2. Add an expression and edit the variables.



Until Successful

Last updated：2023-08-03 17:12:12

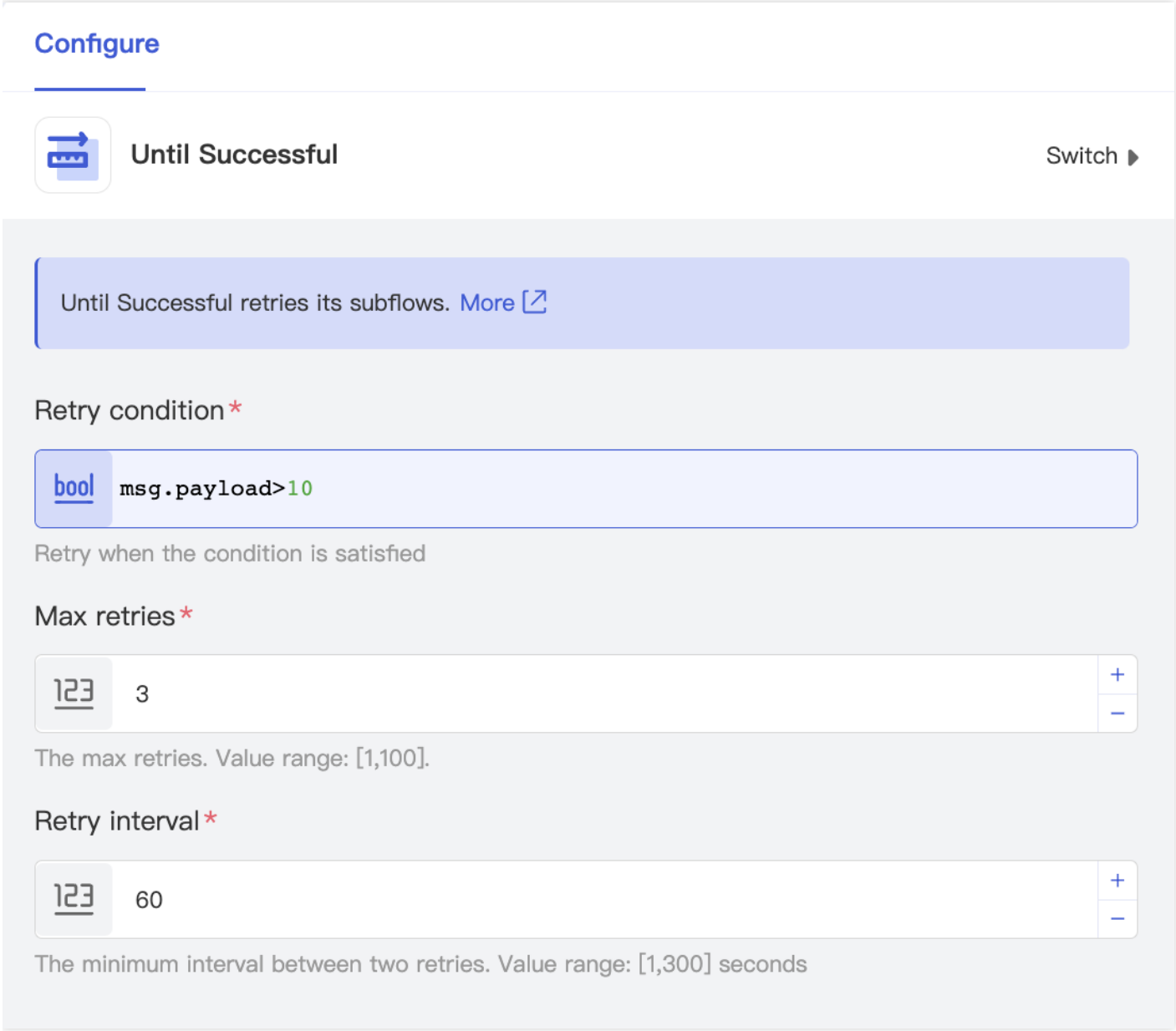
Overview

Until Successful is used to retry the subflow execution.

Method

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Retry condition	bool	Retry condition. When it is met, retry will be triggered.	No	-
Number of retries	int	The number of retries. Value range: 1–100	Yes	3
Retry interval	int	Retry interval in seconds. Value range: 1–300.	Yes	60



Data preview

None.

`message` input to the subflow

`message` Attribute	Value
payload	This attribute inherits the `payload` of the component previous to **Until Successful**.
error	Null.

`message` Attribute	Value
attribute	This attribute inherits the `attribute` of the component previous to **Until Successful**.
variable	This attribute inherits the `variable` of the component previous to **Until Successful**.

Output

`message` Attribute	Value
payload	This attribute inherits the `payload` output by the subflow.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error description.
attribute	This attribute inherits the `attribute` of the subflow.
variable	This attribute inherits the `variable` of the subflow.

Break

Last updated：2023-08-03 17:12:12

Overview

Break needs to be used together with the For Each or While component to stop a loop. It can stop executing the loop statement even if the sequence is still in recursion or the loop condition is not `false`. When there is a nested loop, Break will jump out of the loop at the current layer and start to execute the next component.

Operation Configuration

Connection description

None.

Parameter configuration

None.

Data preview

None.

Output

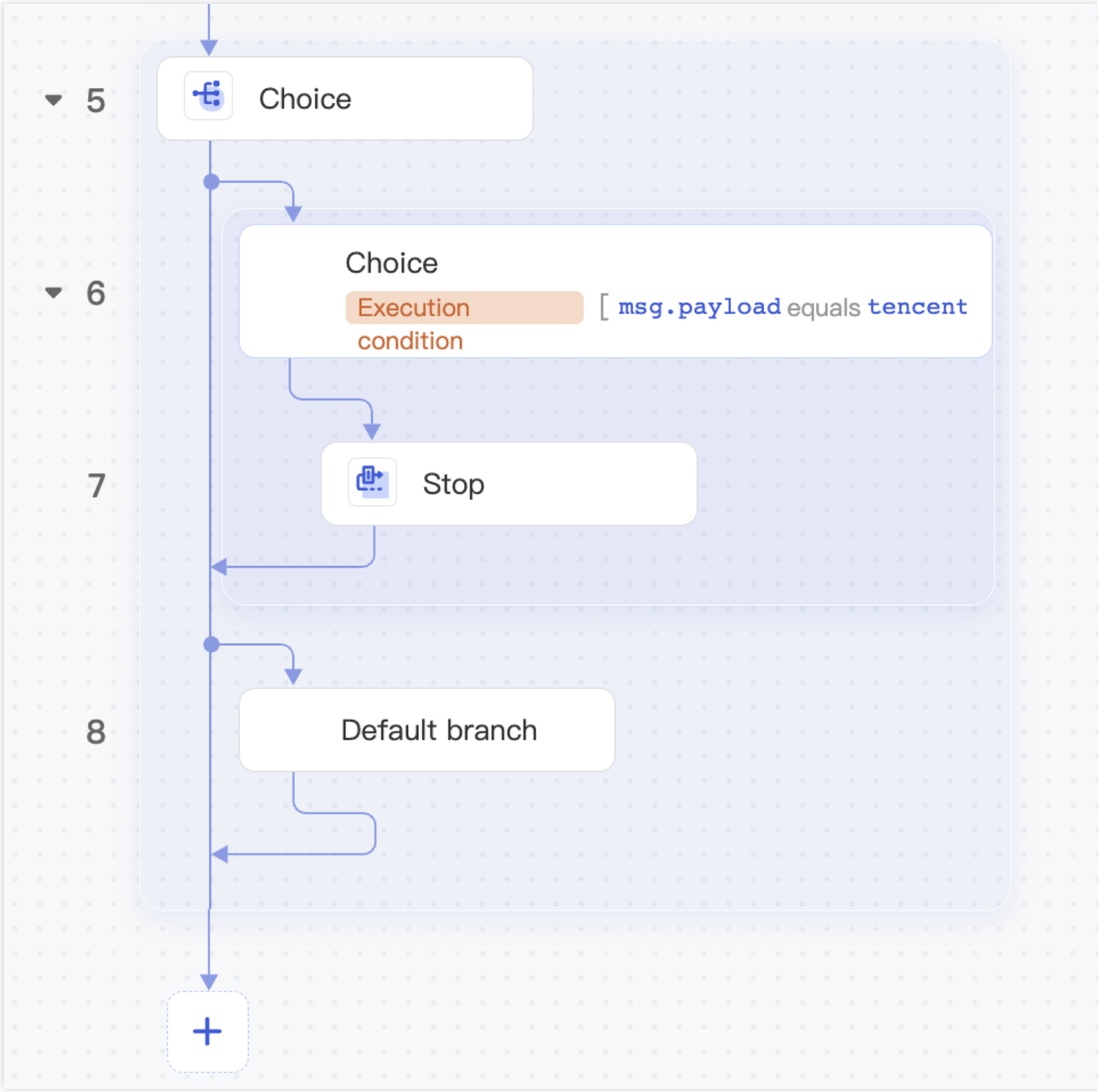
The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	• In the For Each component, after the Break component is executed, `payload` will inherit the `payload` of the component previous to For Each. • In the While component, after the Break component is executed, `payload` will inherit the `payload` output by the component previous to Break.
error	Null.

`message` Attribute	Value
attribute	• In the For Each component, after the Break component is executed, `attribute` will inherit the `attribute` output by the component previous to For Each. • In the While component, after the Break component is executed, `attribute` will inherit the `attribute` output by the component previous to While.
variable	• In the For Each component, after the Break component is executed, `variable` will be the `variable` output by the component previous to For Each and the `variable` declared in the For Each subflow. • In the While component, after the Break component is executed, the `variable` will be the `variable` output by the component previous to While and the `variable` declared in the While subflow.

Example

1. Add a For Each component and set the list to be traversed.
2. Use a Choice component and add a condition in **Conditional branch**.
3. On the **Conditional branch** node, add a **Break** component to jump out of the traversal process when `BMW` is traversed.



Continue

Last updated：2023-08-03 17:12:12

Overview

Like the Break component, Continue needs to be used together with the For Each or While component. It is used to jump out of the current loop and execute the next loop.

Operation Configuration

Connection description

None.

Parameter configuration

None.

Data preview

None.

Output

The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	• In the For Each component, after the Continue component is executed, `payload` will be the data to be traversed next time. • In the While component, after the Continue component is executed, `payload` will inherit the `payload` output by the component previous to Continue.
error	Null.
attribute	• In the For Each component, after the Continue component is executed, `attribute` will inherit the `attribute` output by the component previous to For Each. • In the While component, after the Continue component is executed, `attribute` will inherit the `attribute` output by the component previous to While.

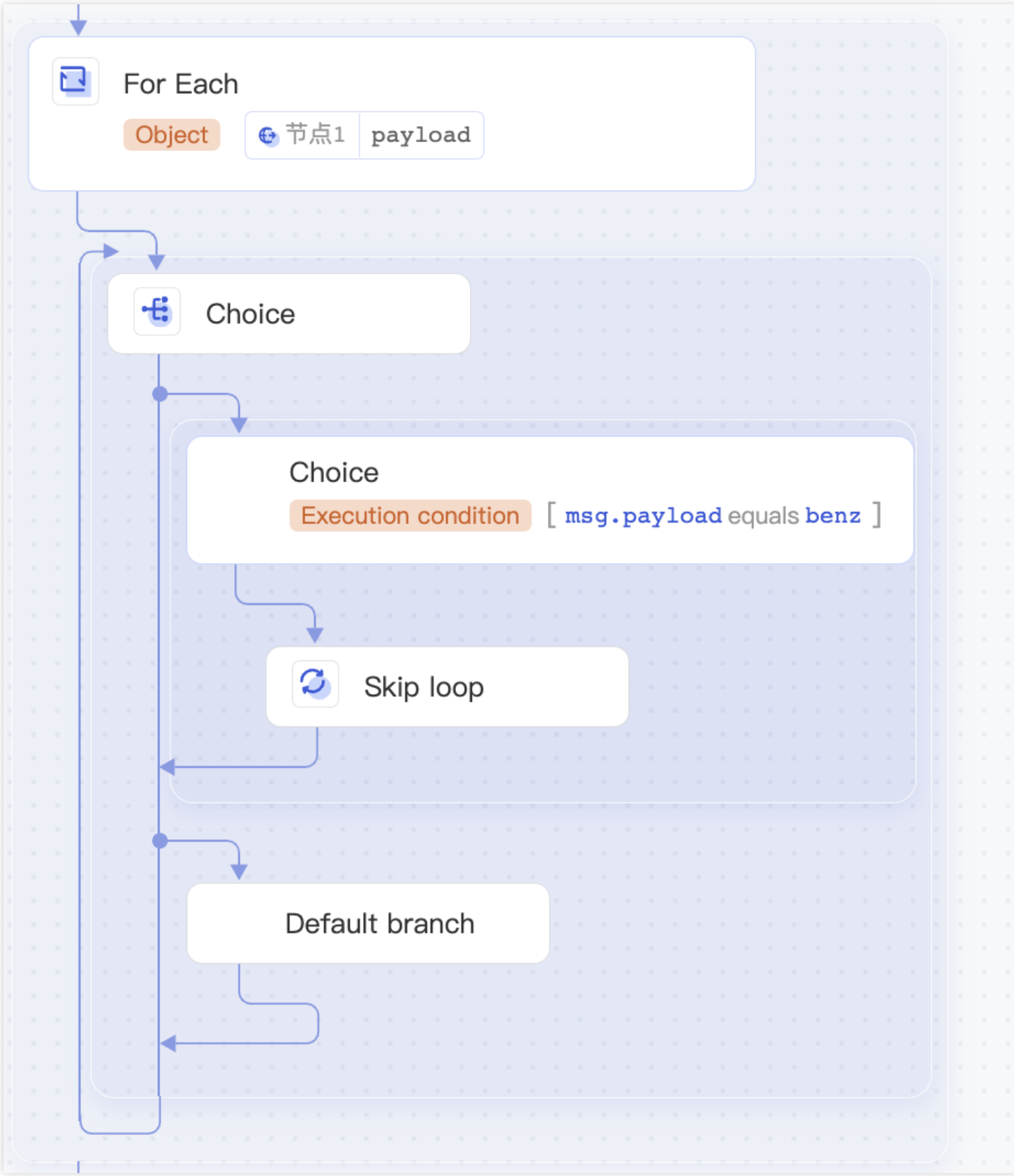
`message` Attribute	Value
variable	• In the For Each component, after the Continue component is executed, `variable` will inherit the `variable` output by the component previous to Continue. • In the While component, after the Continue component is executed, the `variable` will inherit the `variable` output by the component previous to Continue.

Example

In the following example, the For Each component is used to calculate the sum of all odd numbers in the list, and the Continue component is used to jump out of the loop for an even number.

1. Add a For Each component and enter the set to be traversed `[1,2,3,4,5]` .
2. Add a Choice component and filter data on the When node.

3. Add a Continue node on the When node.



For Each

Last updated：2023-08-03 17:12:12

Overview

For Each is a loop control component, which is similar to for/foreach in several programming languages. In For Each, you can configure a subflow to execute the subflow logic for each element in the specified data set.

Operation Configuration

Parameter configuration

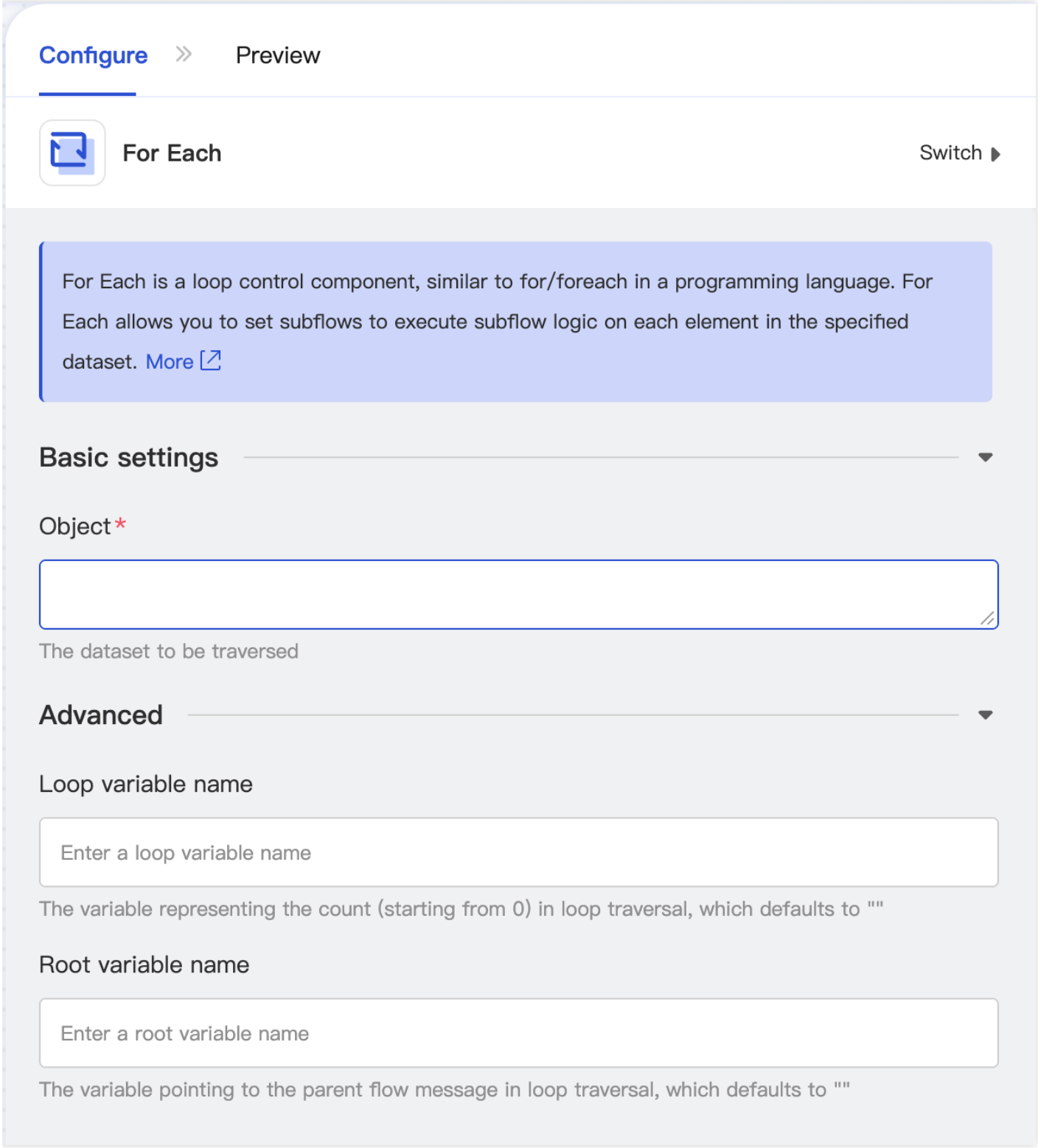
Parameter	Data Type	Description	Required	Default Value
Data set	string, list, dict, and int	The data set to be traversed. • If the data type is `string`, all characters in the string will be traversed. • If the data type is `list`, all elements in the list will be traversed. • If the data type is `dict`, all values in the dictionary will be traversed. • If the data type is `int`, for example, if the data set is `3`, the data set `[0,1,2]` will be traversed.	Yes	None
Counter	string	The counter is a variable, which stores the current number of iterations and starts from `0`. You need to enter a variable name such as `msg.vars.get('#counter variable#')` to use the counter. For example, if the counter variable is set to the default value `counter`: In the first loop, `msg.vars.get('counter')` will be `0`. In the second loop, `msg.vars.get('counter')` will be `1`.	Yes	counter

Parameter	Data Type	Description	Required	Default Value
Root message	string	The root message is also a variable, which stores the message of the main flow. You need to enter a variable name. You can enter `msg.vars.get('#root message name#').payload` to access the payload data of the main flow. If the default value `rootMessage` is used, you can use `msg.vars.get('rootMessage').payload` to access the payload data of the main flow in the For Each subflow.	Yes	rootMessage

Note：

Generally, you only need to configure the data set.

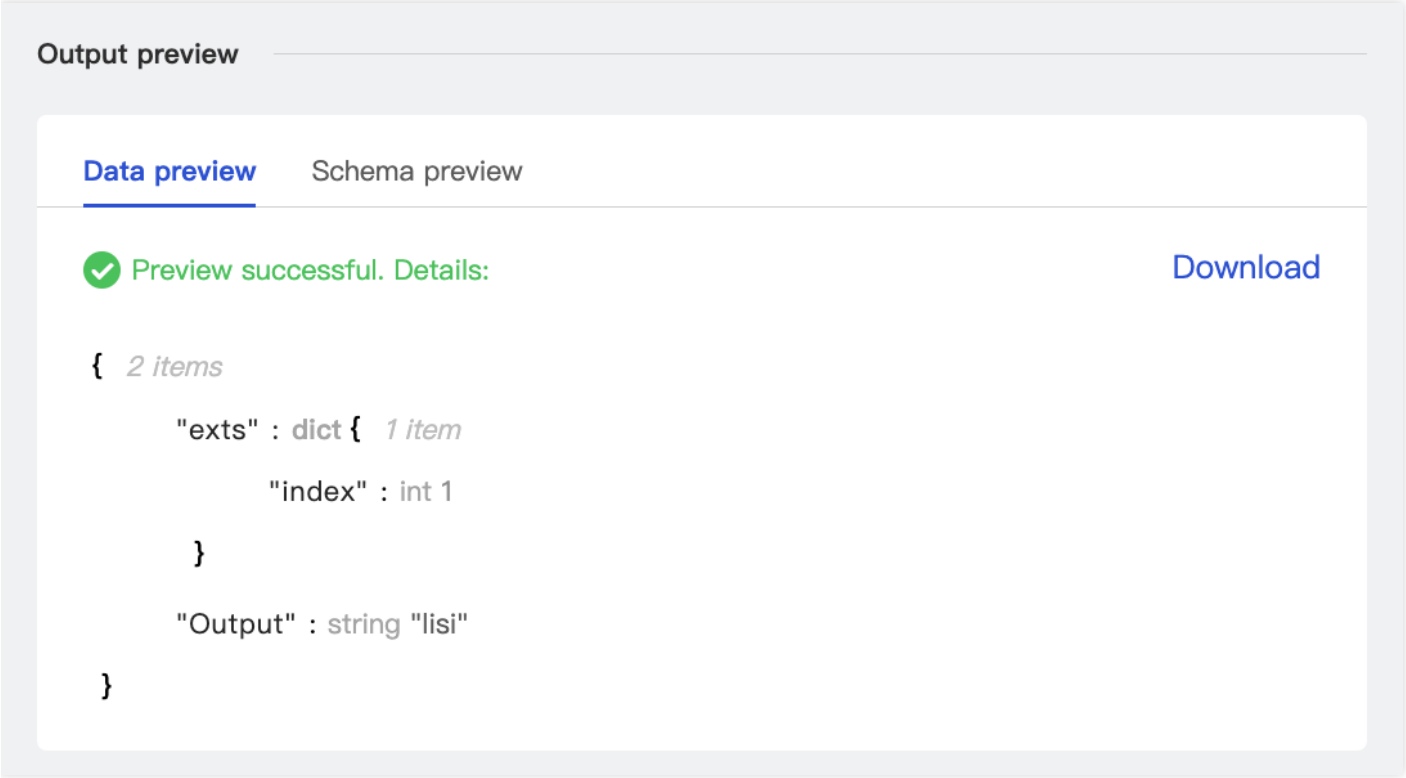
Configuration page



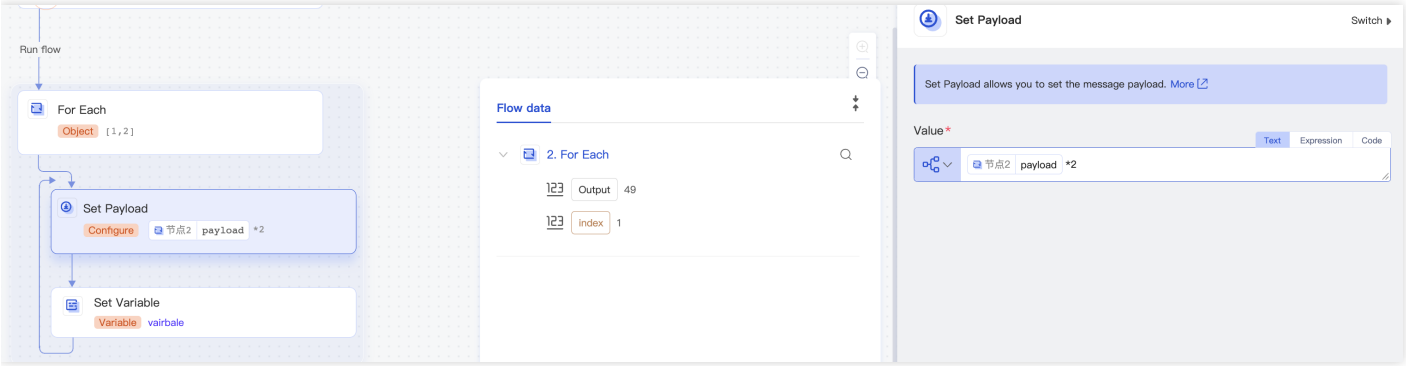
Data preview

Preview Field	Data Type	Description

Preview Field	Data Type	Description
payload	any	Input value in each traversal, which is also an element in the data set.
index	int	Position in each traversal. This field represents the subscript position of the current input value in the data set, which starts from `0` .



The data preview content is visible only in the subflow. Components in the subflow can directly use `payload` and `index` in the For Each component.



`message` input to the subflow

`message` Attribute	Value
payload	An element in the data set. For example, if the data set to be iterated is `[1,2,3]`: In the first loop, `payload` in the subflow will be `1`. In the second loop, `payload` will be `2`. If the data set to be iterated is `{"key":"key1", "value":"value1"}` of `dict` type: In the first loop, `payload` in the subflow will be `value1`. In the second loop, `payload` will be `value2`.
error	Null.
attribute	Null.
variable	This attribute inherits the `variable` of the main flow and has two new variables: counter and root message. If you use the default values of the two new variables, you can use expressions `msg.vars.get('counter')` and `msg.vars.get('rootMessage')` to access them. If Set Variable is used in For Each, the new variables will be added to `variable` during subflow execution.

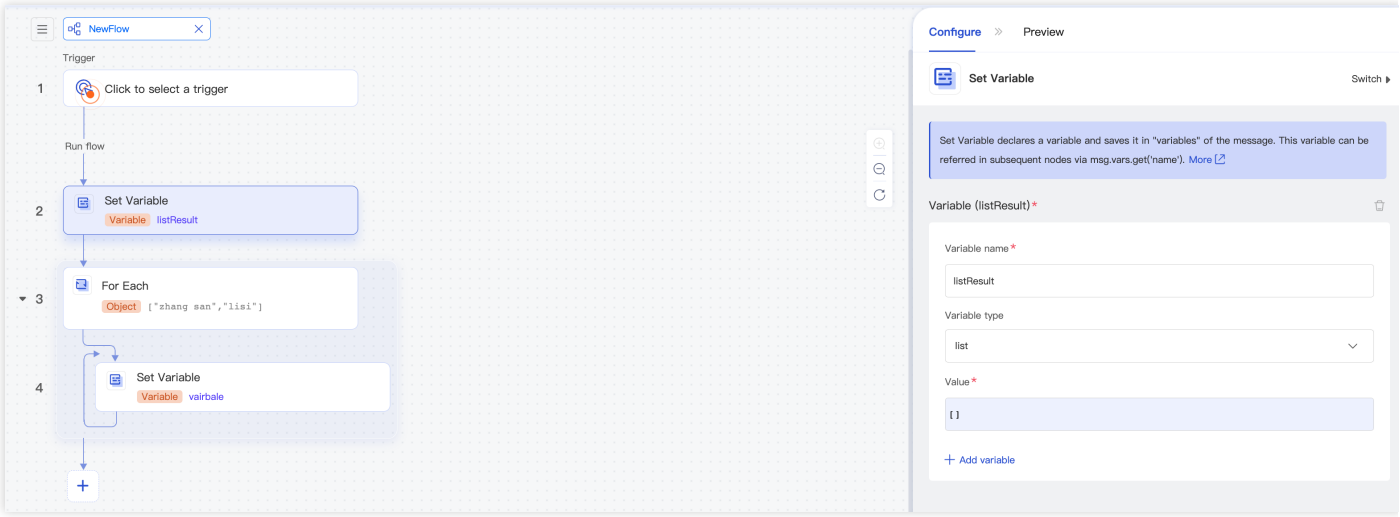
Output

The For Each component doesn't change the content of `message`, and subsequent nodes only notice the changes in variables.

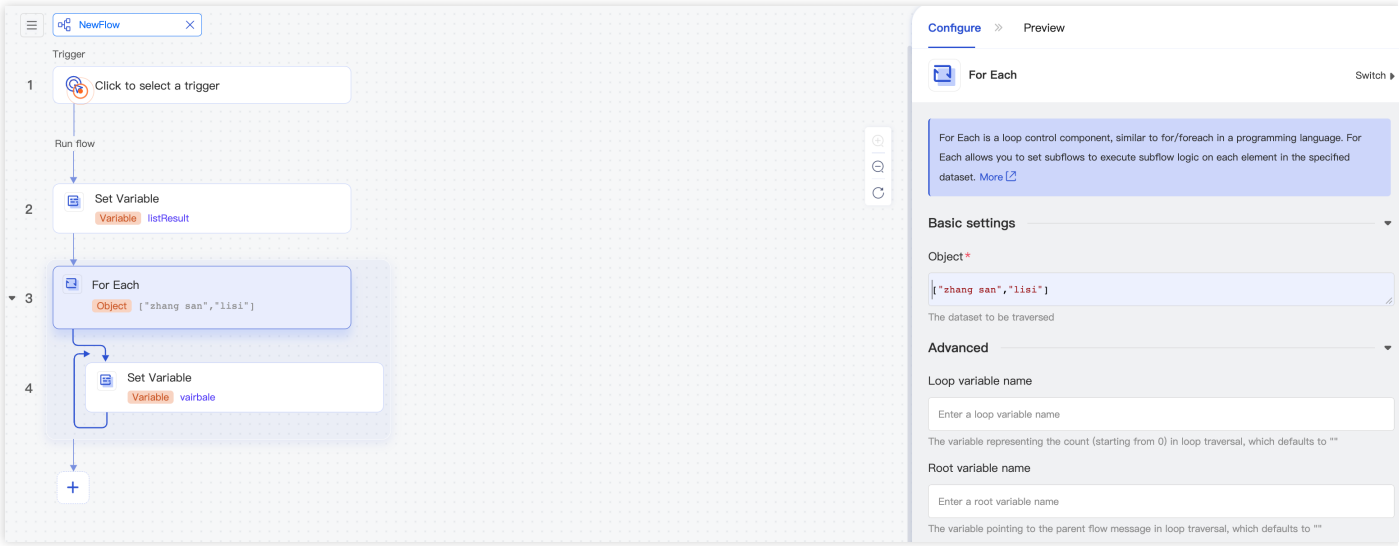
Example

The following describes how to use the For Each component to traverse the list and add numbers and prefixes to elements in the list.

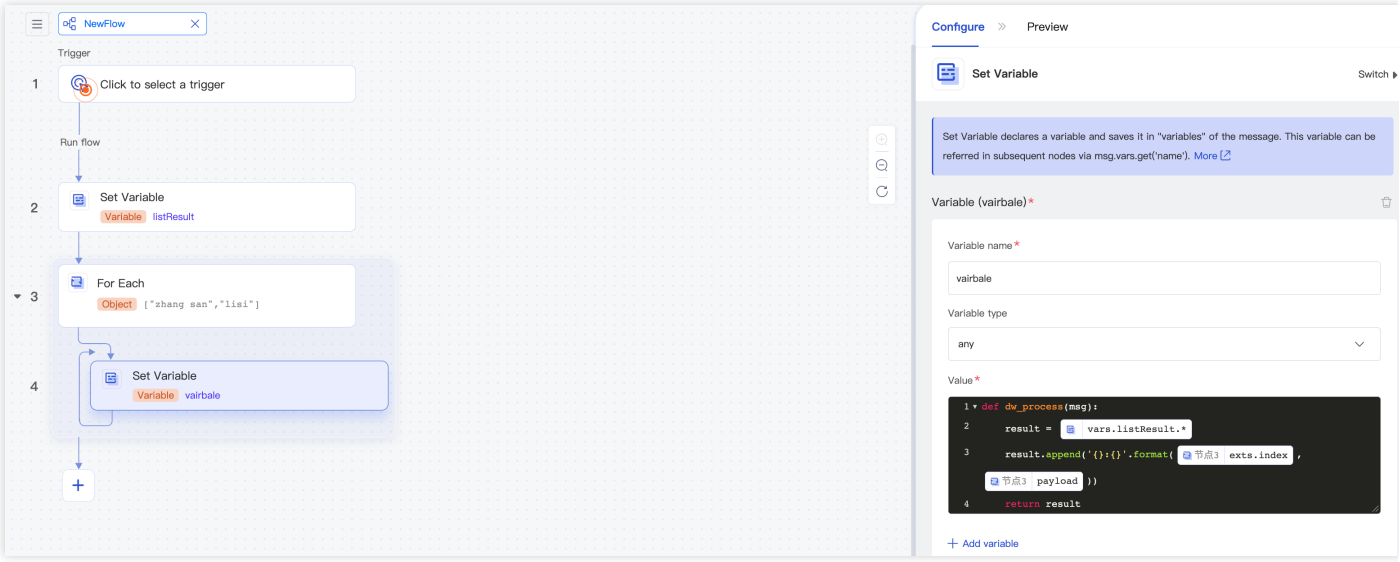
1. Initialize the variable `listResult` .



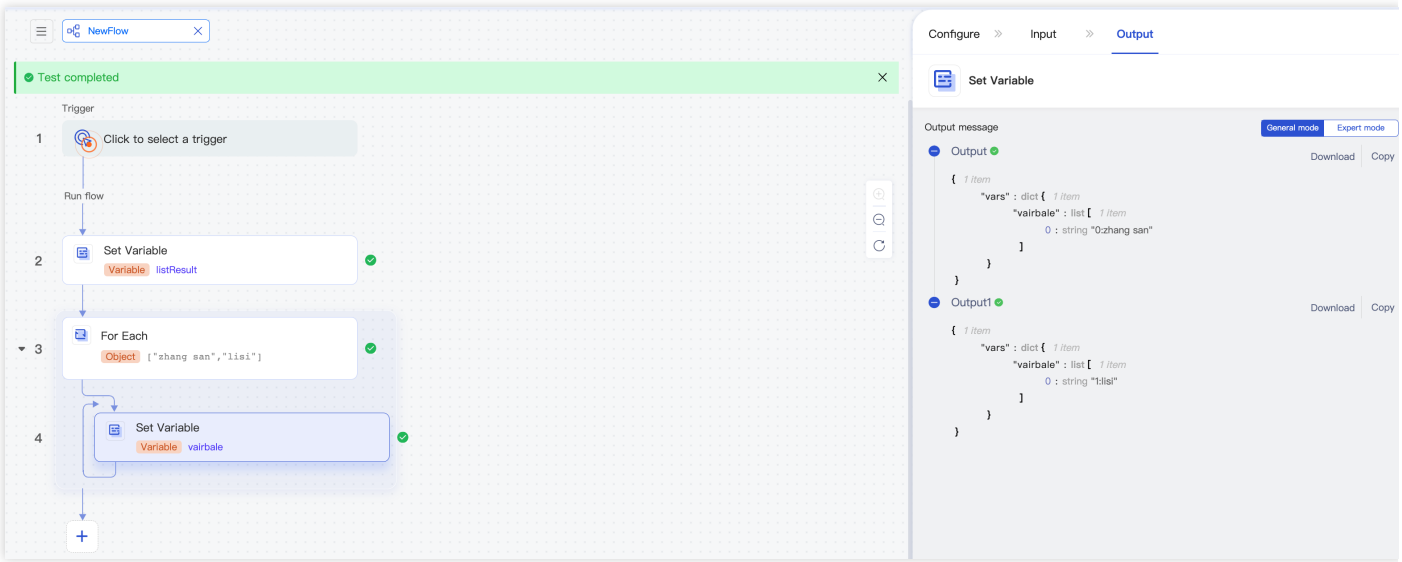
2. Add a For Each component and configure it.



3. Set the variable in the For Each subflow. Specifically, add a prefix to each element (the subscript position of the element) and add it to the `listResult` variable.



4. Perform a unit test to check the output effect.



Set Payload

Last updated：2023-08-03 17:12:12

Overview

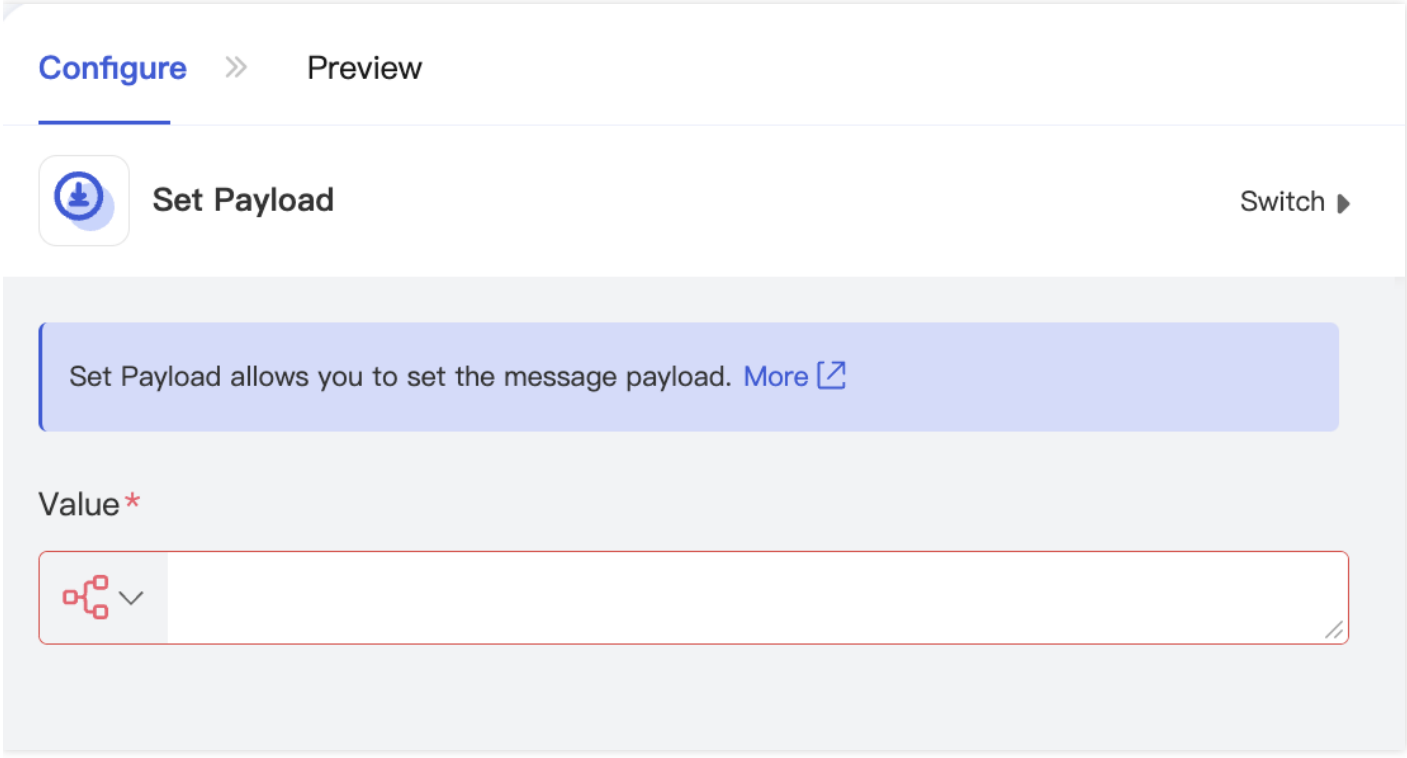
Set Payload is used to set the `payload` attribute in `message`. It supports expressions and literals. To enter a literal, select the data type and enter the literal in the input box. To enter an expression, select the `any` type and write an expression.

Operation Configuration

Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Value	any	The data to be saved in `payload`.	Yes	None

Configuration page

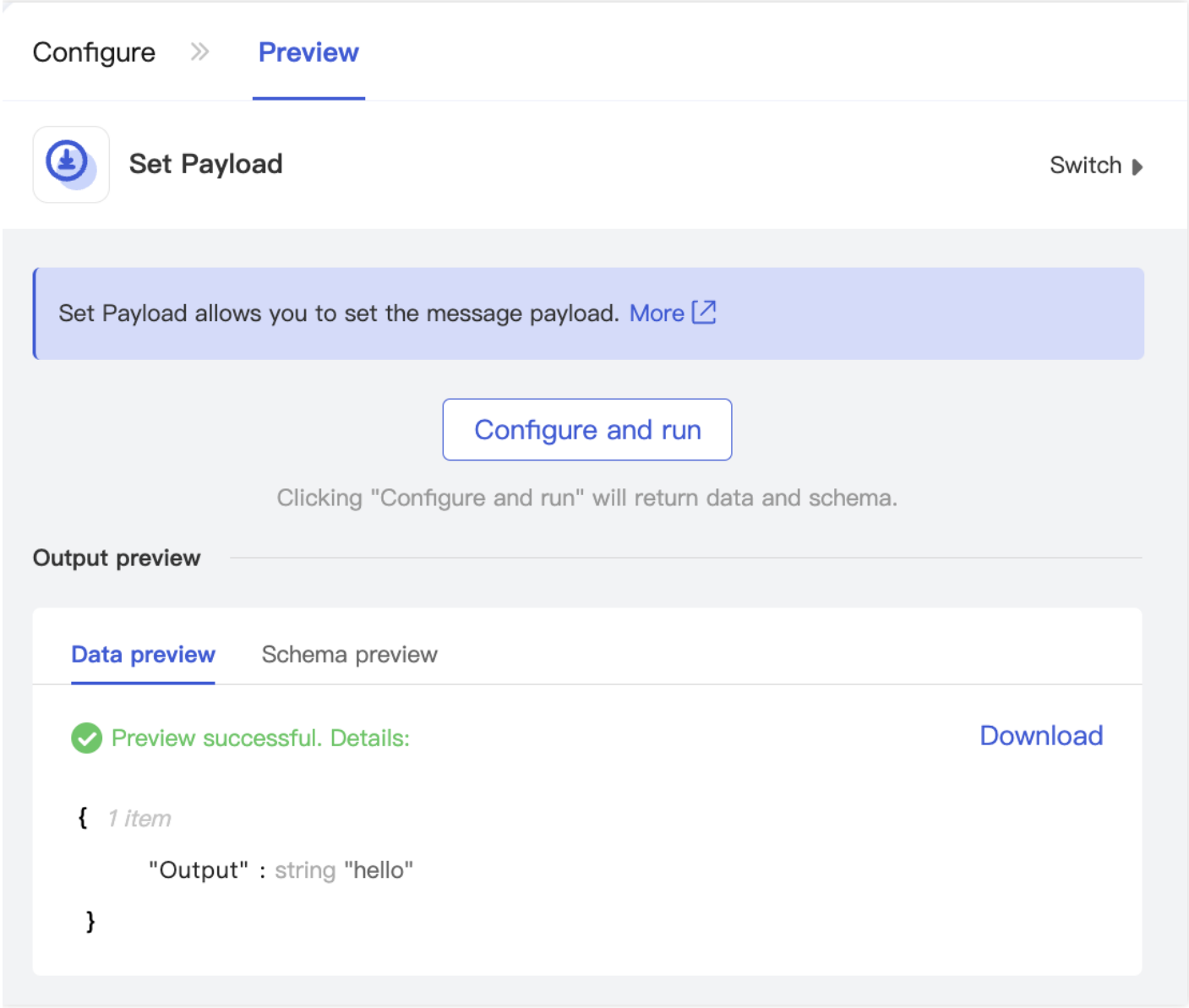


Output

The `message` output by the component is as detailed below:

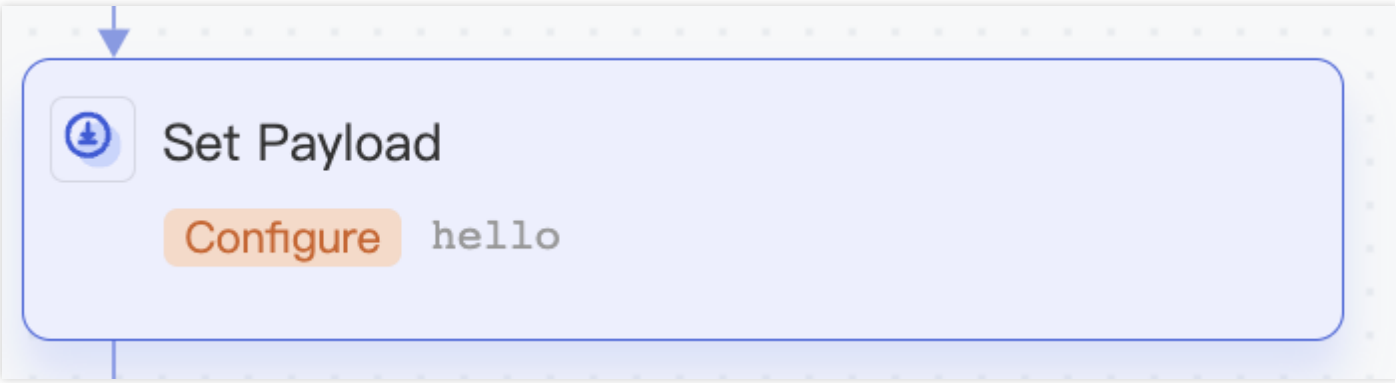
`message` Attribute	Value
payload	The data entered by the user.
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute inherits the `attribute` of the previous component.
variable	This attribute inherits the `variable` of the previous component.

Data preview

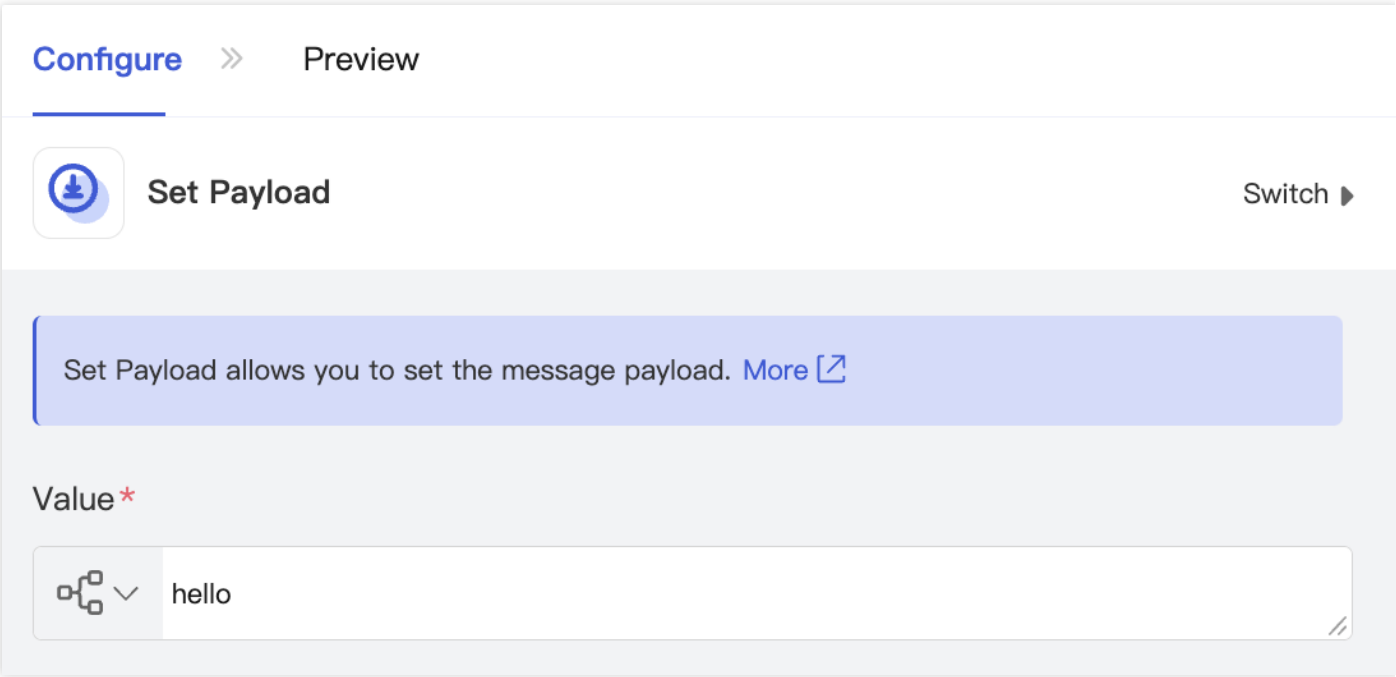


Example

1. Add a Set Payload component.



2. Enter the data to be configured.



Set Variable

Last updated：2023-08-03 17:12:12

Overview

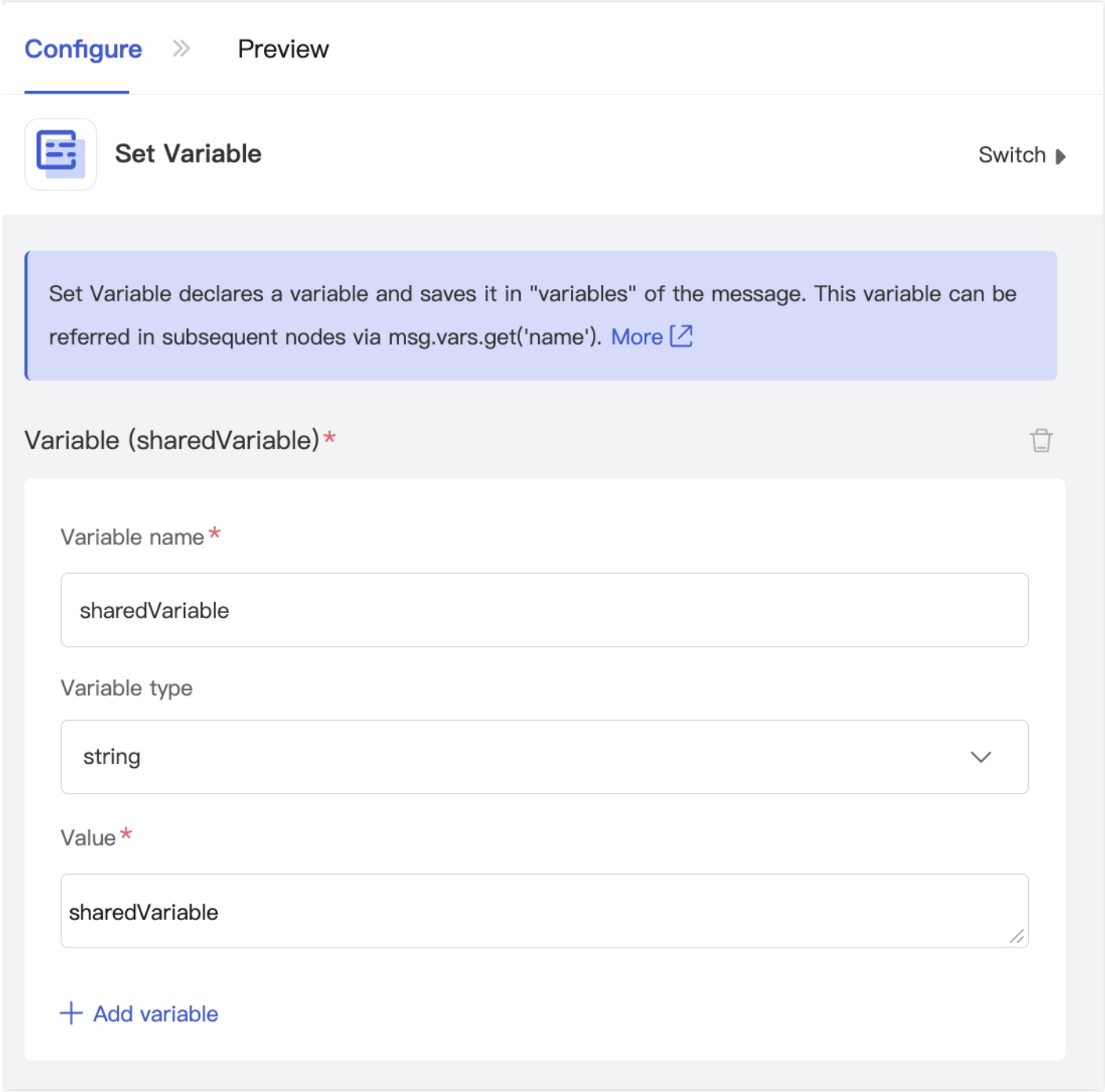
Set Variable is used to declare a variable and save it in `variables` in `message`, so that subsequent nodes can reference it in the form of `msg.vars.get('name')`.

Method

Parameter configuration

Parameter	Data Type	Description	Required	Default Value	Remarks
Variable name	string	Variable name.	Yes	None	-
Value	any	Specific value of the variable.	Yes	None	-
Type	string	Data type of the variable	Yes	string	-
Operation	string	Variable operation	No	None	This parameter is available only if a variable of `list` or `dict` type exists.

Configuration page



Output

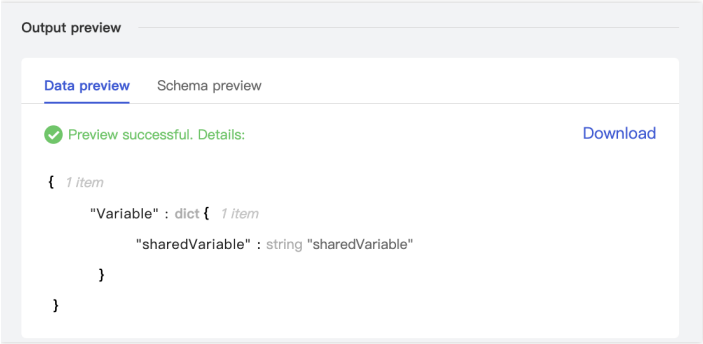
To reference `variables`, you need to use the `msg.vars.get('company')` expression.

The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	This attribute inherits the `payload` of the previous component.

`message` Attribute	Value
error	• `error` will be empty if the flow is executed successfully. • `error` will be of `dict` type and contain the `Code` and `Description` fields if the flow fails to be executed. The `Code` field indicates the error type, and the `Description` field indicates the error details.
attribute	This attribute inherits the `attribute` of the previous component.
variable	`variable` of the previous component and the variables added in the current component.

Data preview



Try-Catch

Last updated：2023-08-03 17:12:12

Overview

The Try-Catch component consists of an **execution** subflow and one or multiple **error capture** subflows. You can configure it to capture errors thrown when the **execution** subflow runs as well as system errors. It can also be used together with the Raise Error component to capture custom errors. If the **execution** subflow throws an error, the first matched **error capture** subflow will be executed. If no subflows are matched, the error will be thrown to the outer layer.

Operation Configuration

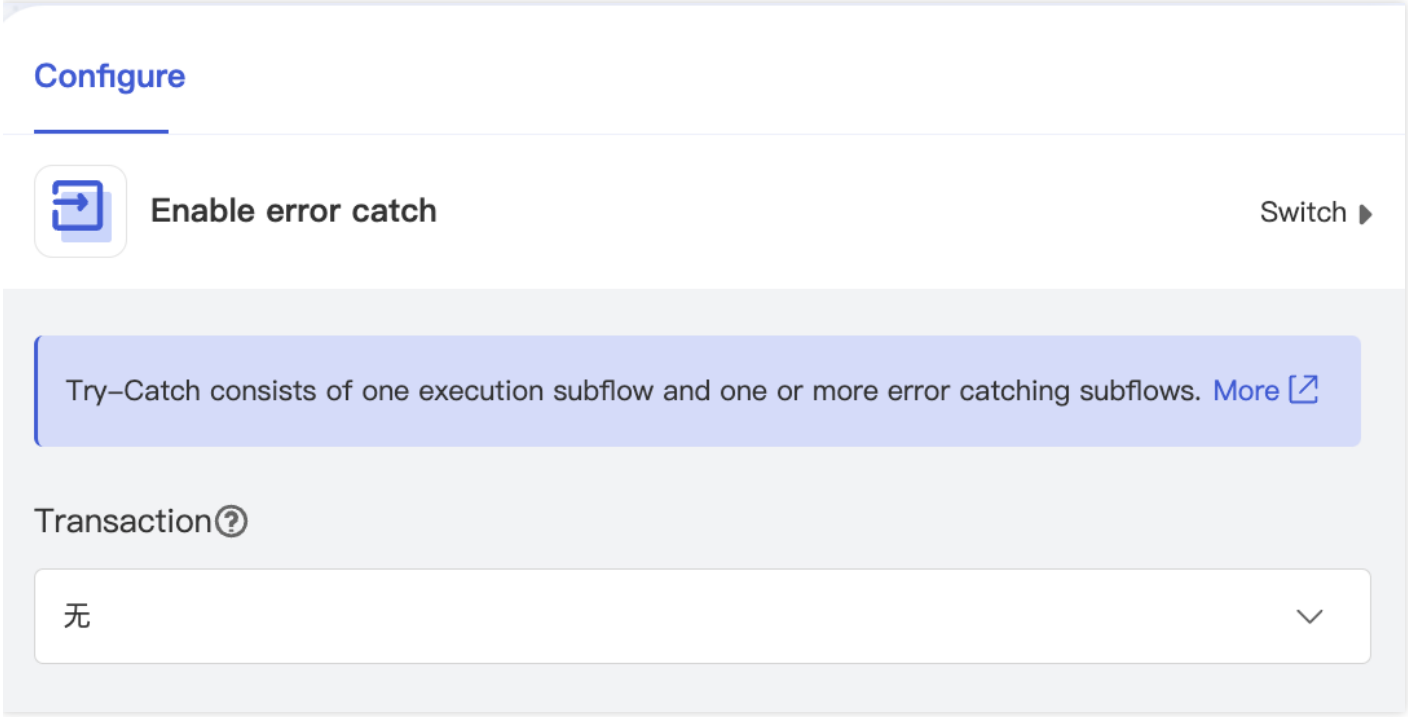
Connection description

None.

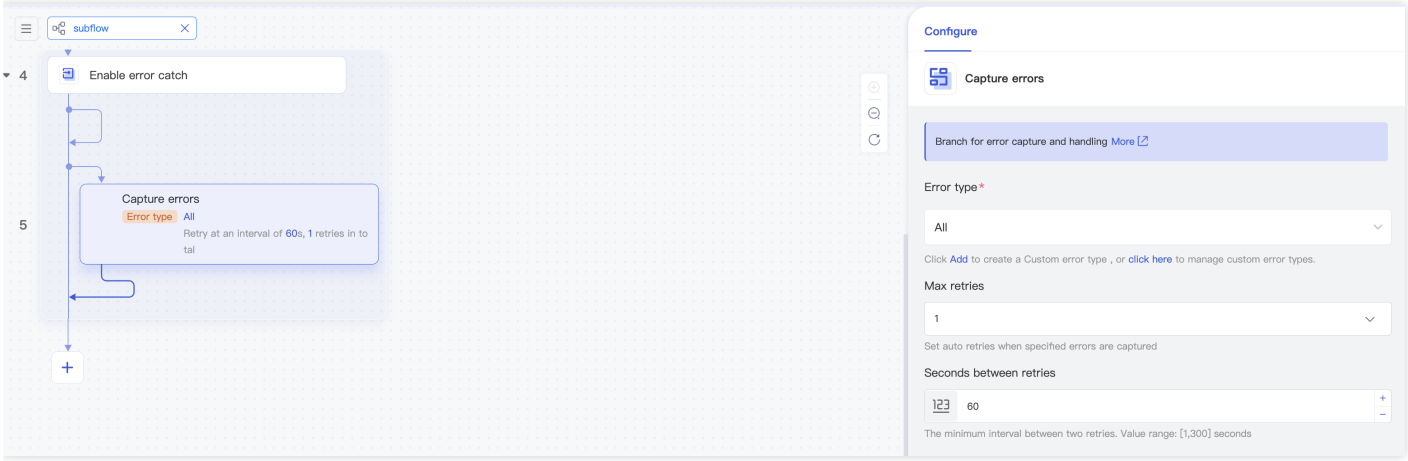
Parameter configuration

Parameter	Data Type	Description	Required	Default Value
Transaction	Enumeration	This parameter is used together with the `Database` component and is used to roll back the database operation when execution fails.	Yes	None
Error type	string	Type of the error thrown during flow execution, which is selected in the drop-down list. You can select one, multiple, or all types.	Yes	None
Number of retries	int	The maximum number of retries of the **execution** subflow when an error of the specified type occurs. Value range: 0–5. The number of retries for each **error capture** subflow is counted separately.	Yes	No retry
Retry interval	int	Interval between two retries. Value range: 1–300s.	No	60

The transaction is configured on the **Start Error Capture** node.



The error type and the maximum number of retries are configured on the **Try-Catch** node.



Data preview

None.

`message` input to the subflow

`message` Attribute	Value
payload	This attribute inherits the `payload` of the component previous to **Try-Catch**.

`message` Attribute	Value
error	This attribute inherits the `error` of the component previous to **Try-Catch**.
attribute	This attribute inherits the `attribute` of the component previous to **Try-Catch**.
variable	This attribute inherits the `variable` of the component previous to **Try-Catch**.

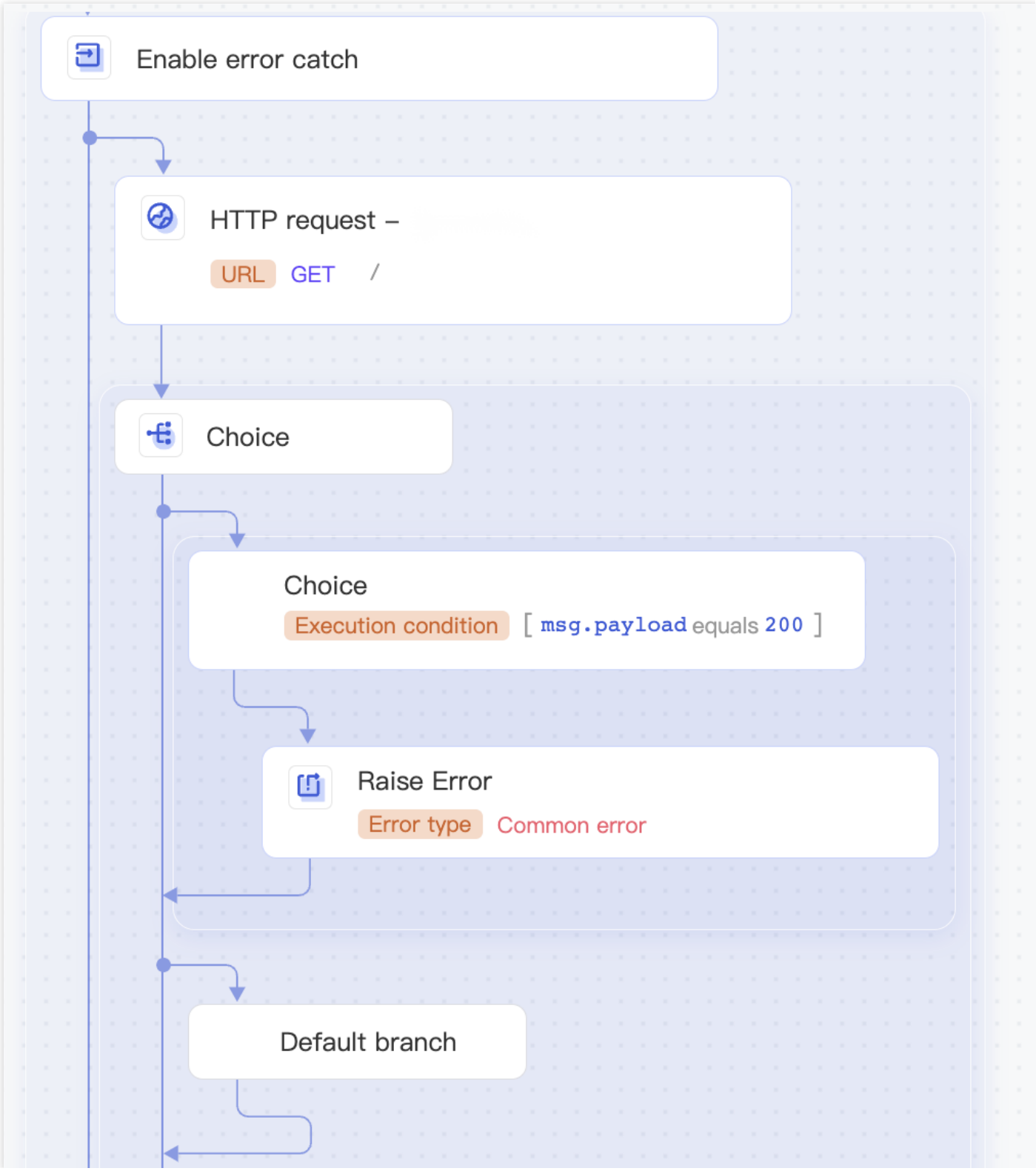
Output

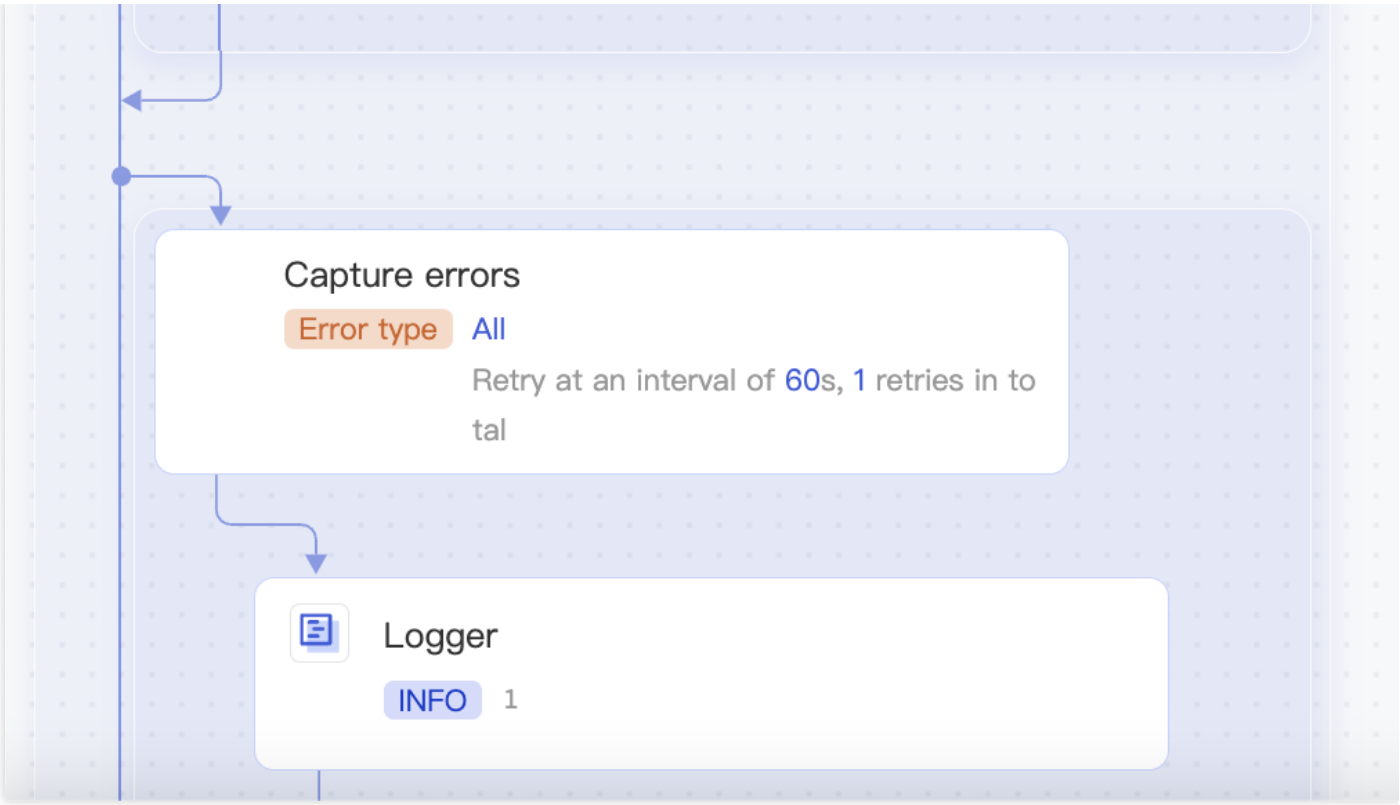
The `message` output by the component is as detailed below:

`message` Attribute	Value
payload	• If the **execution** subflow runs normally, the value of `payload` will be the `payload` output by the **execution** subflow. • If an error is thrown and captured, the value of `payload` will be the `payload` output by an **error capture** subflow. • If a thrown error is not captured, the flow will stop running.
error	• If the **execution** subflow throws an error and the error is not captured, or if an **error capture** subflow throws an error, `error` will store the error message of `dict` type and contain the `Code` and `Description` fields. The `Code` field indicates the error type, and the `Description` field indicates the error description. • Otherwise, `error` will be empty.
attribute	• If the **execution** subflow runs normally, the value of `attribute` will be the `attribute` output by the **execution** subflow. • If an error is thrown and captured, the value of `attribute` will be the `attribute` output by an **error capture** subflow. • If a thrown error is not captured, the flow will stop running.
variable	• If the **execution** subflow runs normally, the value of `variable` will be the `variable` output by the **execution** subflow. • If an error is thrown and captured, the value of `variable` will be the `variable` output by an **error capture** subflow. • If a thrown error is not captured, the flow will stop running.

Example

1. Add the custom error type **Request failure**.
2. When a request fails, an error of this type will be thrown.
3. Capture the error and execute the retry policy. The HTTP request in the **execution** subflow will be executed again, and the Choice component will be executed. If all retries fail, the error will be logged.





Choice

Last updated：2023-08-03 17:12:12

Overview

Choice is a branch selection statement. Similar to if-else, it is used to execute actions by condition.

Choice contains two types of branches: conditional branch and default branch. In Choice, you can add multiple conditional branch nodes, each of which contains a Boolean expression. Choice will evaluate the conditional branch nodes one by one until the first Boolean expression meets the condition. Then, it will execute the subflow configured on the conditional branch node. If all When conditions cannot be matched, the action of the default branch will be executed.

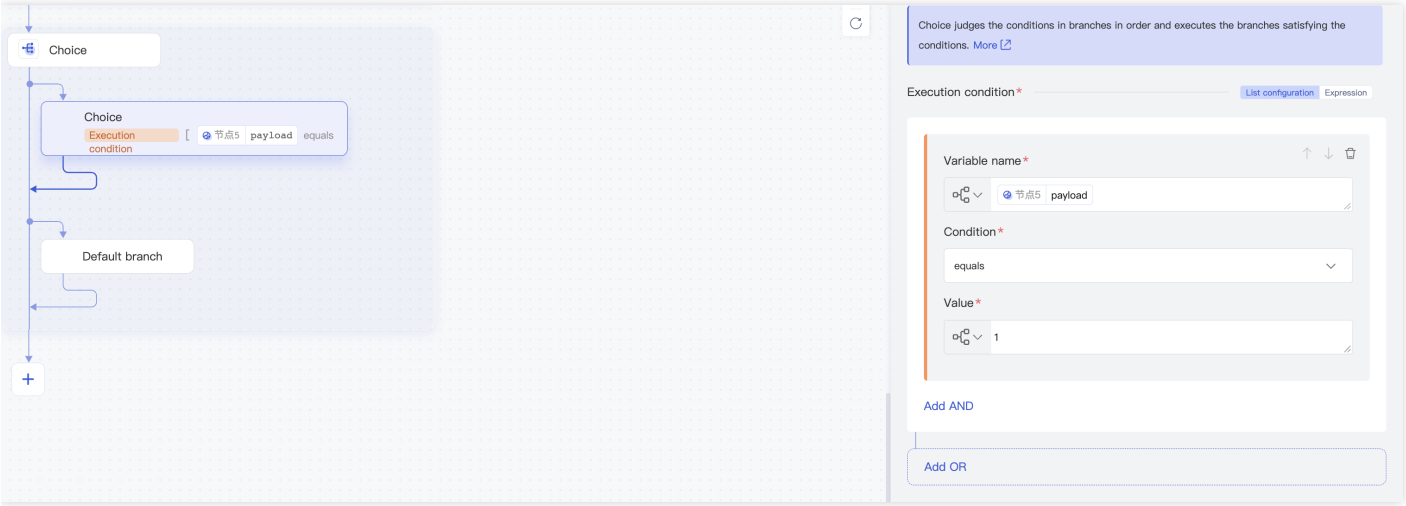
Operation Configuration

Parameter configuration in expression mode

On the When node, you can configure a conditional statement to control branch selection.

Parameter	Data Type	Description	Required	Default Value
Execution condition	bool	Condition. If the condition is met, the corresponding subflow will be executed.	Yes	None

Configuration page in expression mode

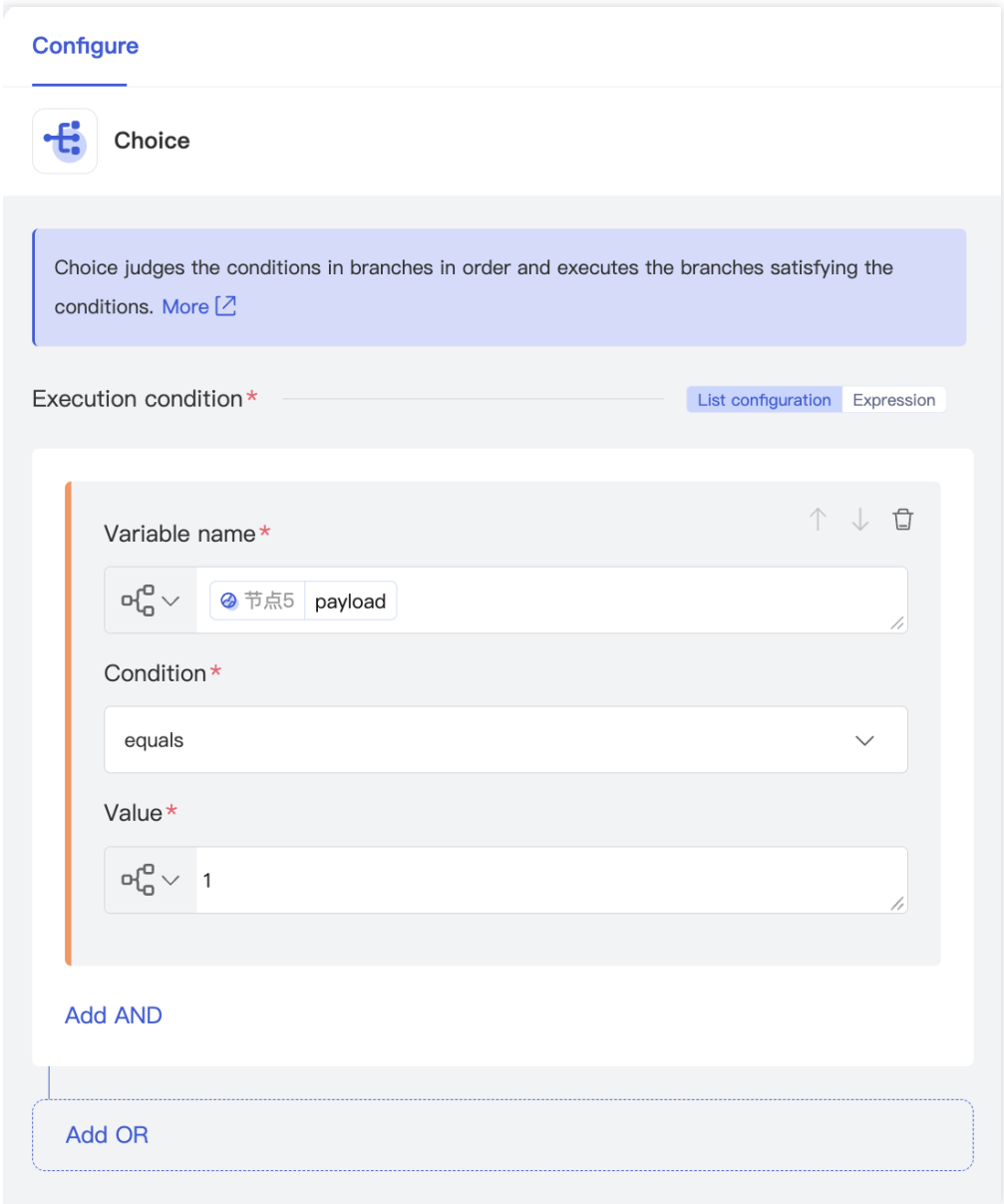


Parameter configuration in list mode

When configuring multiple comparison conditions on the GUI, you can use the logical operator `OR` or `AND` to connect them.

Parameter	Data Type	Description	Required	Default Value
Value	any	Value.	Yes	None
Condition	Enumeration	Condition, i.e., comparison operator.	Yes	None

Configuration page in list mode



Data preview

None.

`message` input to the subflow

The entire `message` of the main flow is inherited.

Output

The entire `message` eventually output by the subflow is output, including the error.

API Management

Last updated：2023-08-03 17:20:54

Overview

With many new APIs launched every day, and more and more enterprises starting to open up their web APIs, API use cases are increasing. Nowadays, the number of daily API calls is surging, and how to manage these APIs securely and efficiently has become a challenge to enterprises.

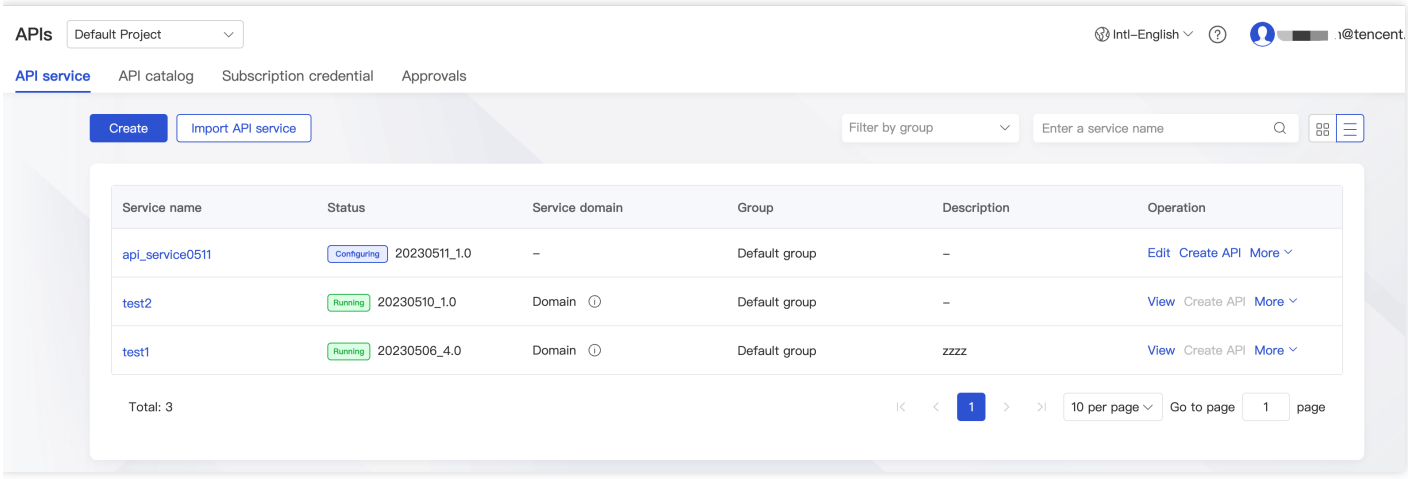
iPaaS offers the API publishing feature, which allows you to quickly package published apps to generate APIs for users to manage and call, and provides API management capabilities to control access permissions and traffic scheduling for APIs.

Directions

API Management Page

Log in to the iPaaS console and click **Integration development** > **APIs** on the left sidebar.

On the **APIs** page, you can create or view API services, view API catalogs, manage API subscription credentials, and manage the approvals.



There are three API service status: configuring, running, and stopped. You can hover to see service domain name to view the publishing environment and domain name of the API service.

The operations supported by the API service include: view, create new API, lauch, remove, delete, view description files, and view release history.

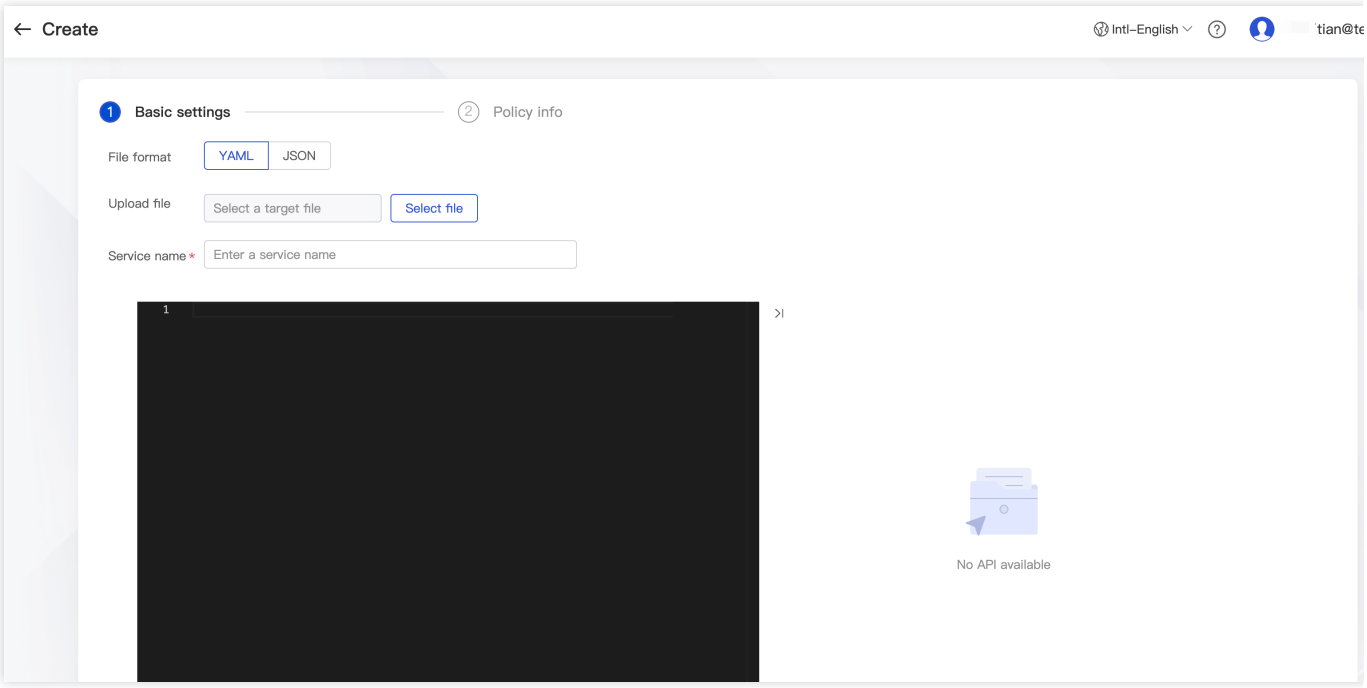
Creating an Service

The API management feature supports OpenAPI Specification v3.0.0. For the object definitions of OpenAPI Specification v3.0.0, see OpenAPI Specification. You can click **Create** to enter the API creation page.

There are two ways to create an API service: including manually creating and importing service.

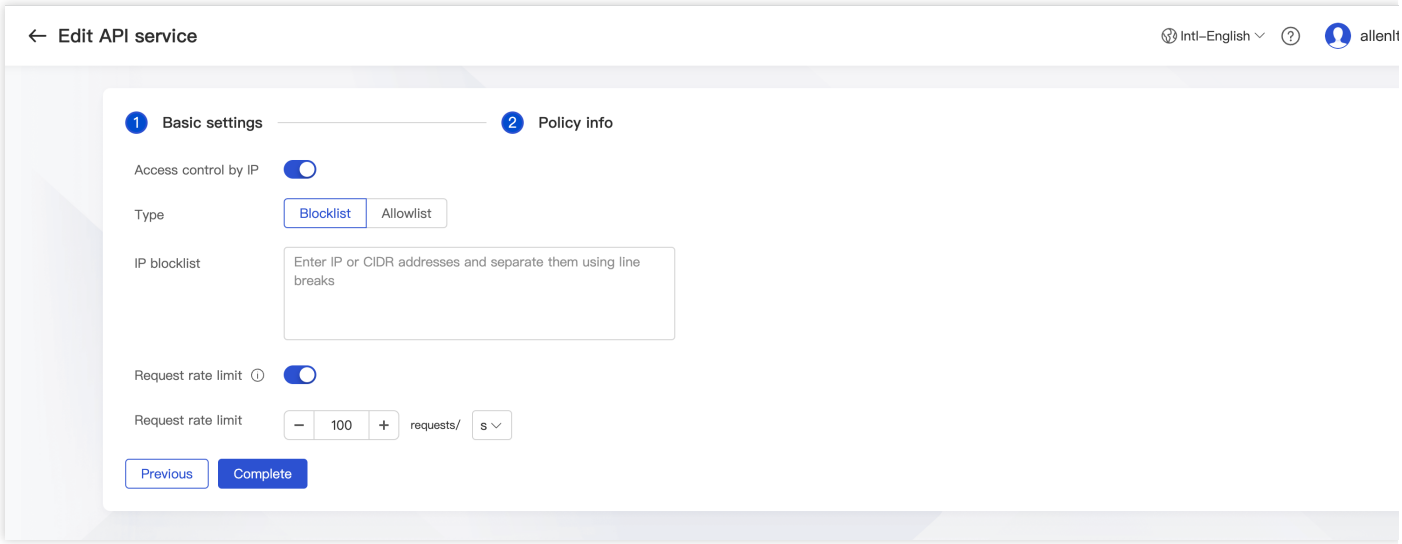
- Creating an API service by importing a description file
- Creating an API service manually

1. On the [APIs] page, click **Import API service**. On the **Basic settings** page, configure the following information and click **Next**.
   - Upload description file: Upload a YAML or JSON file up to 100 KB in size.
   - File format: Select **YAML** or **JSON**.



2. On the **Policy info** page, configure the following information and click **Done** to create the API service.

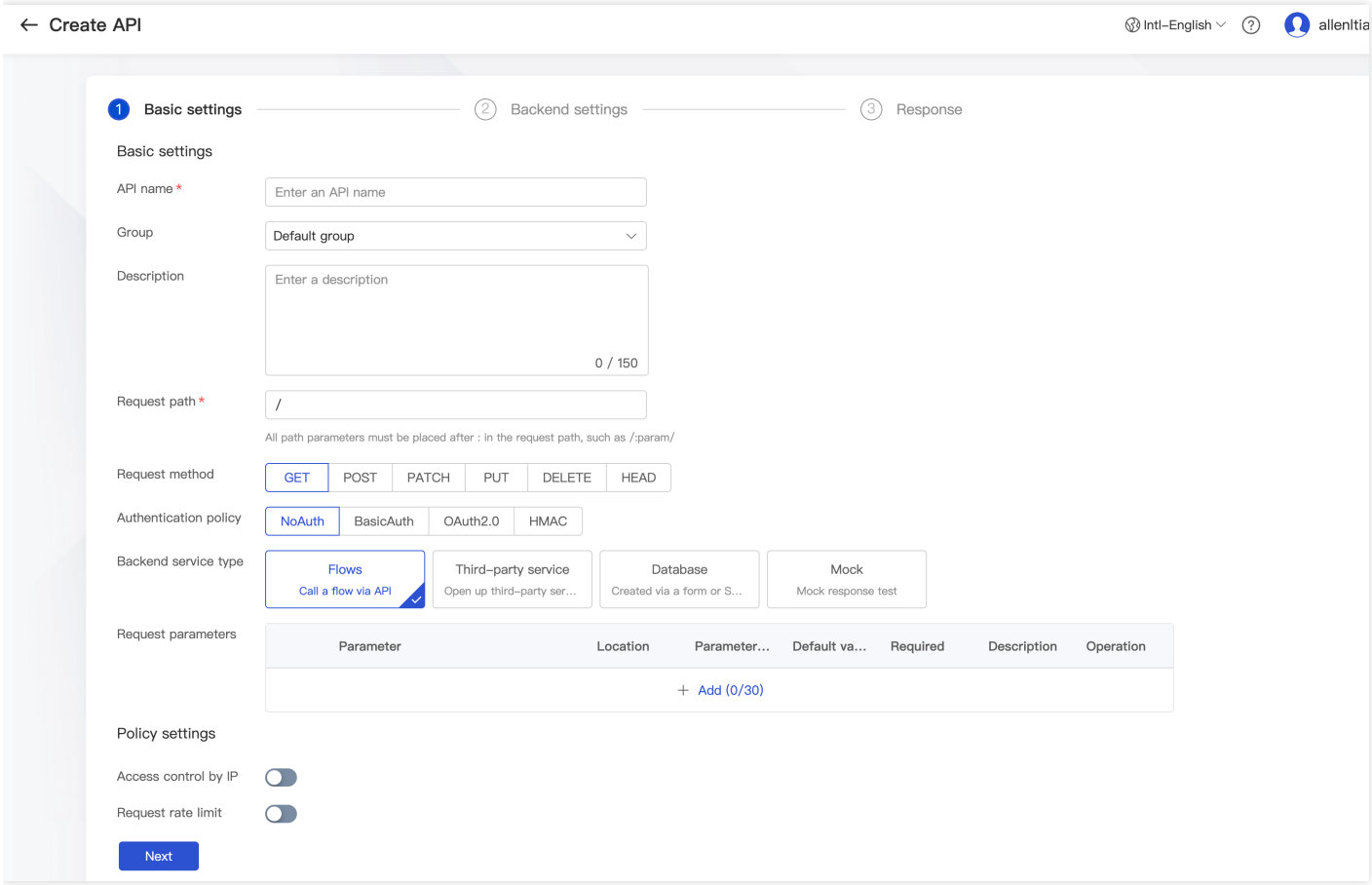
- Access control by IP: You can enable this as needed. After it is enabled, you can enter multiple IPs to restrict access based on the allowlist/blocklist.
- Request rate limit: The maximum number of allowed access requests per unit of time from the configuration time. Value range: 1–1000.



Creating an API

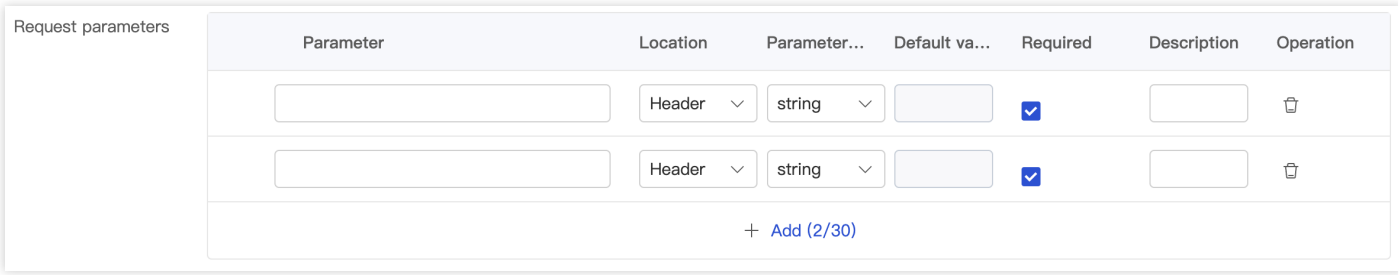
After we have created an API service, we can start editing its specific API. Including API request path, request method, authentication policy, request parameters, policy settings, backend service type and other operations. There are 3 steps to create a new API (The appendix uses postman as an example to introduce how to call the API from the user side).

Step 1: Basic configuration

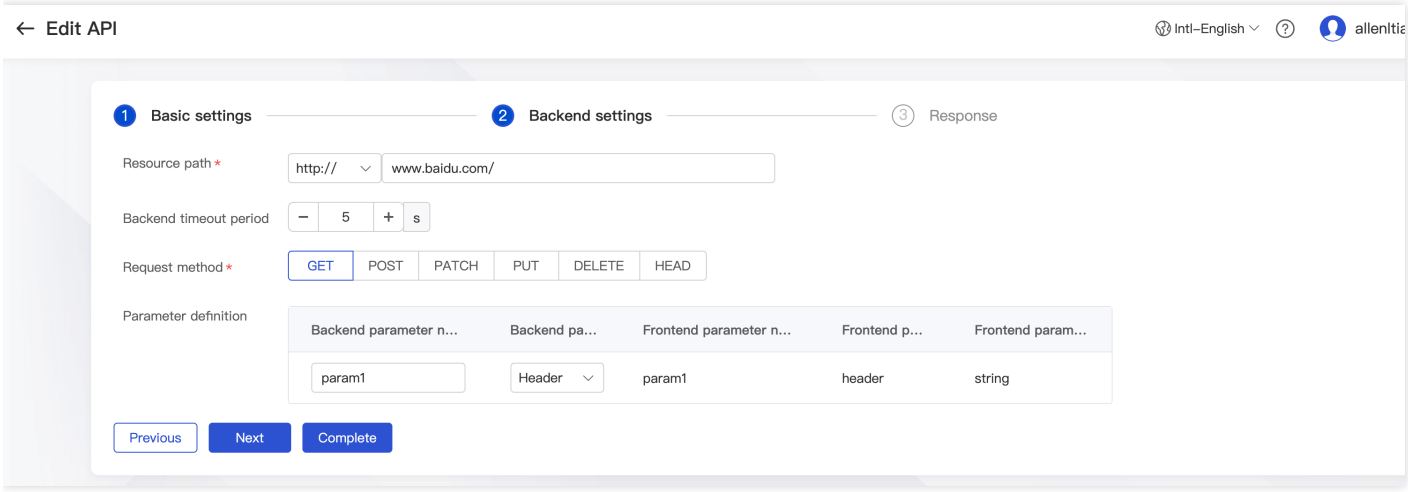


API name and description support customization. For grouping, you can choose the default grouping or create a new grouping.

- Request method: GET, POST, PATCH, PUT, DELETE, HEAD.
- Authentication strategy: NoAuth, BasicAuth, OAuth2.0, HMAC.
- Backbounce service type: integration stream, third-party service, database, Mock. The request parameters of integration flow, third-party service, and Mock can be added by yourself, up to 30.



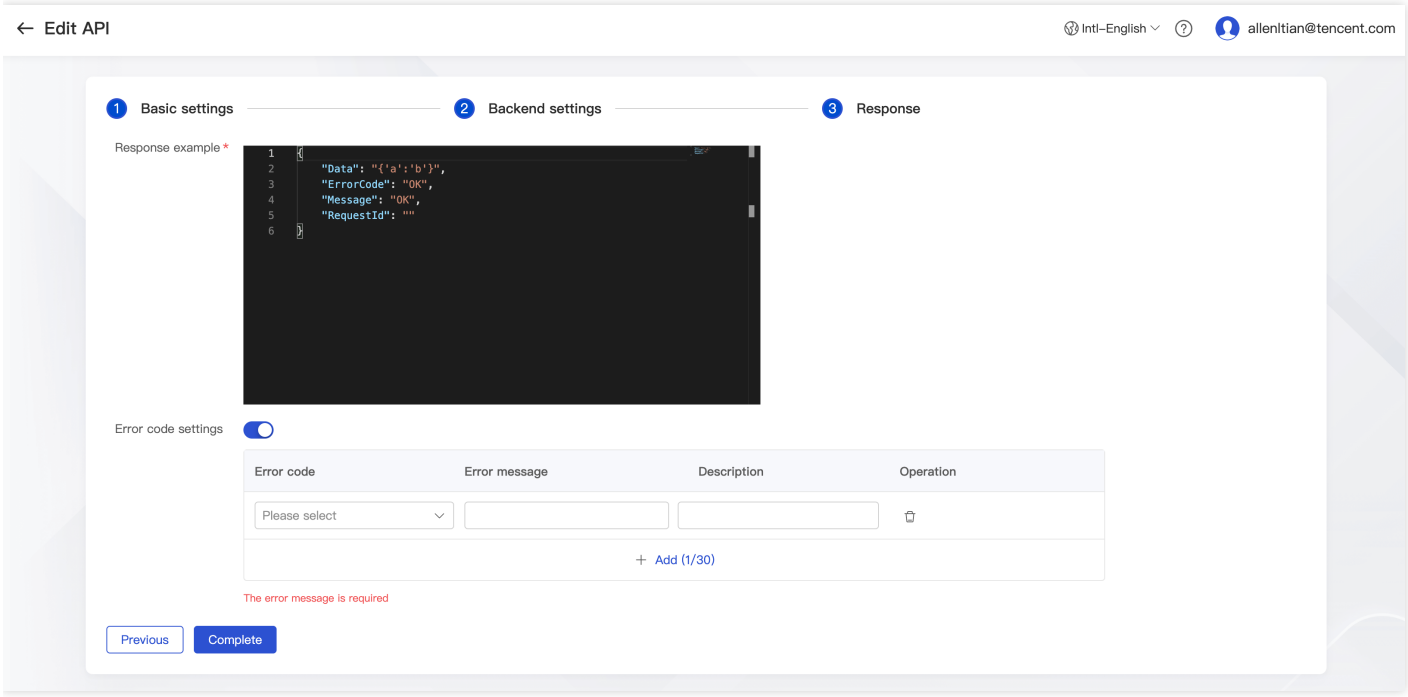
Step 2: Backend configuration



- Backend resource path: Take the integration flow backend service as an example, here you need to select the integration flow triggered by webhook.
- Backend timeout period: It can be preset by default or customized.
- Request method: choose according to user needs. Support: GET, POST, PATCH, PUT, DELETE, HEAD.
- Parameter definition: Support configuration of front-end and back-end parameter mapping.

Step 3: Response configuration

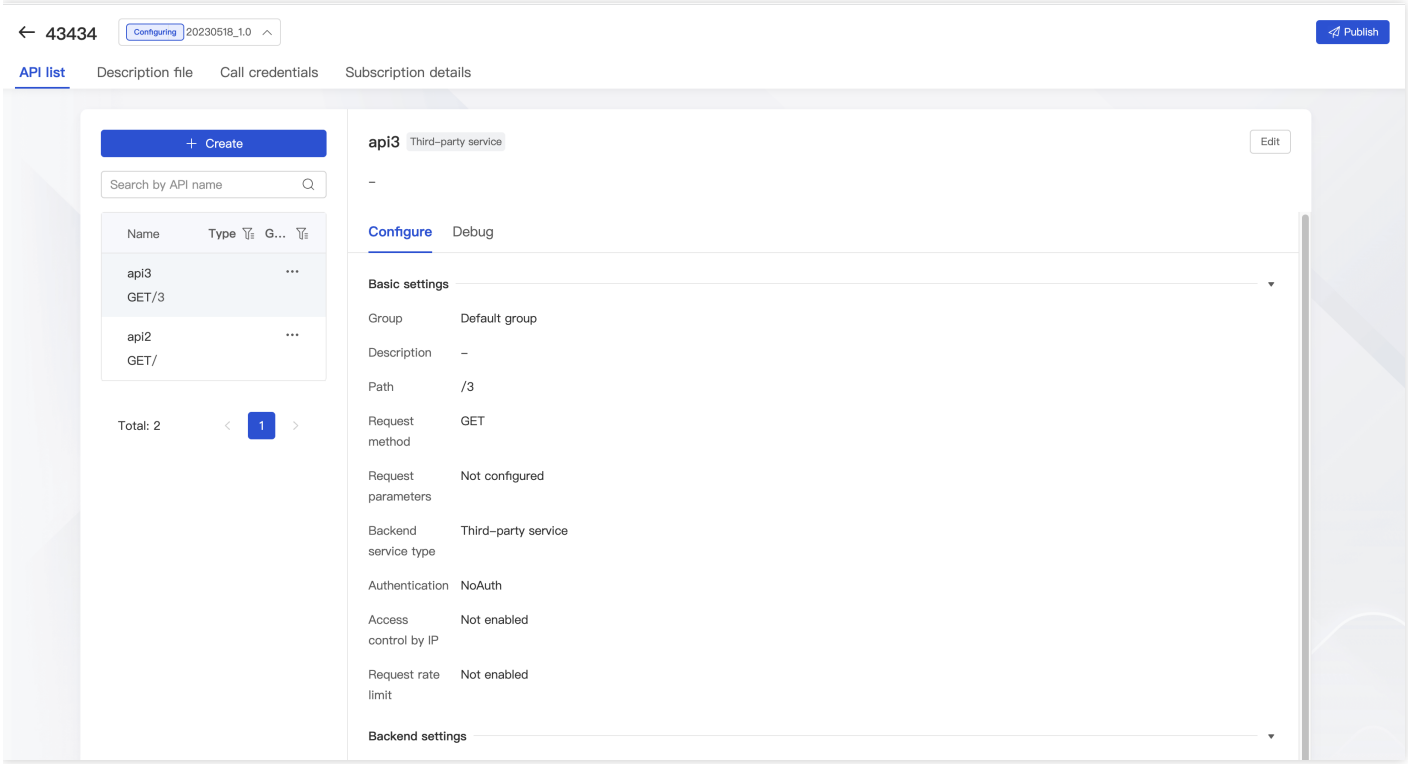
Support configuration response example and error code configuration.



When all the above configurations are completed, click **Finish**, and the API list will be returned, and the created API information will be displayed here.

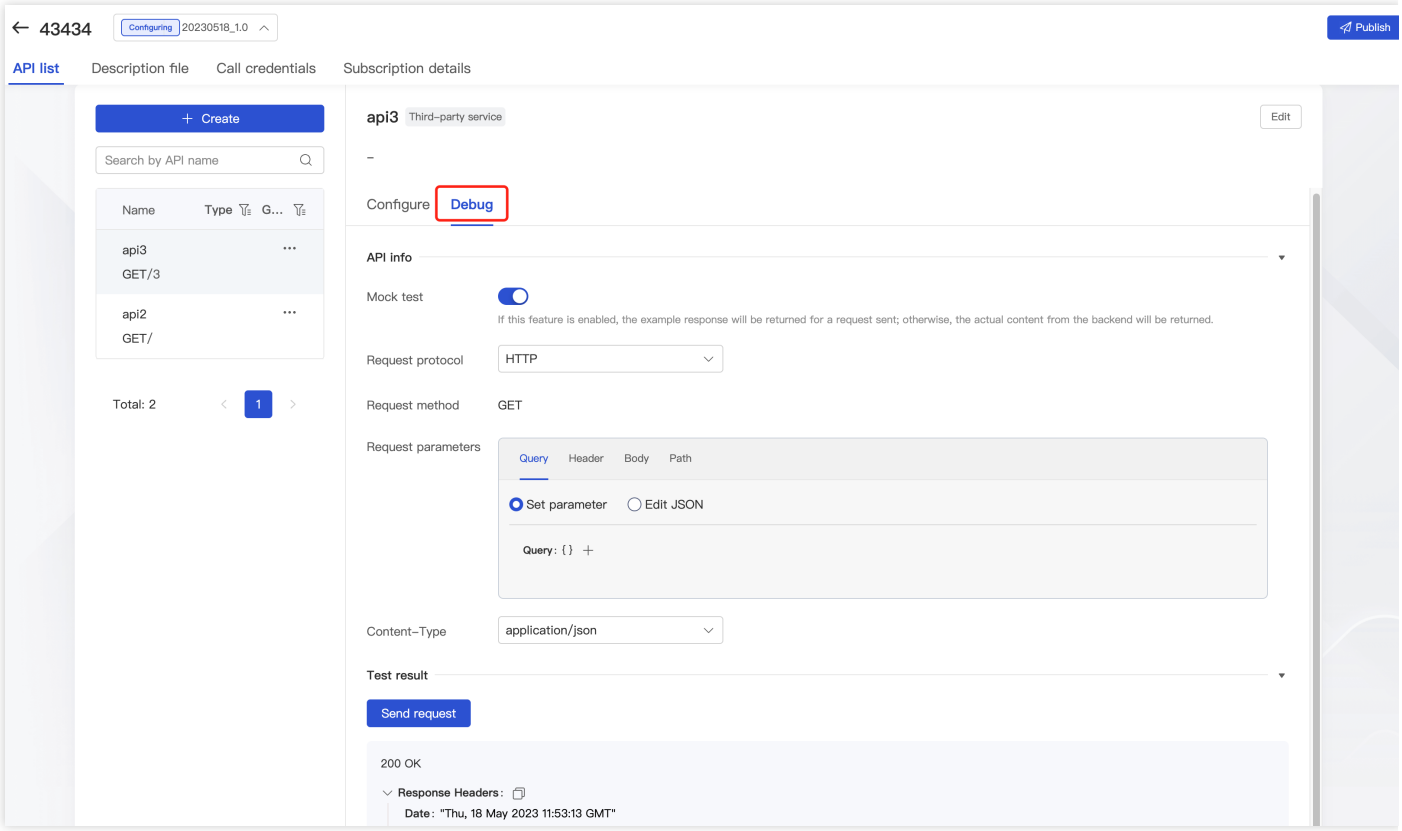
API list

The created API will be displayed in the API list, and you can create, view and edit APIs on this page. You can publish APIs, set and view API description files and call credentials, and view API subscription details.

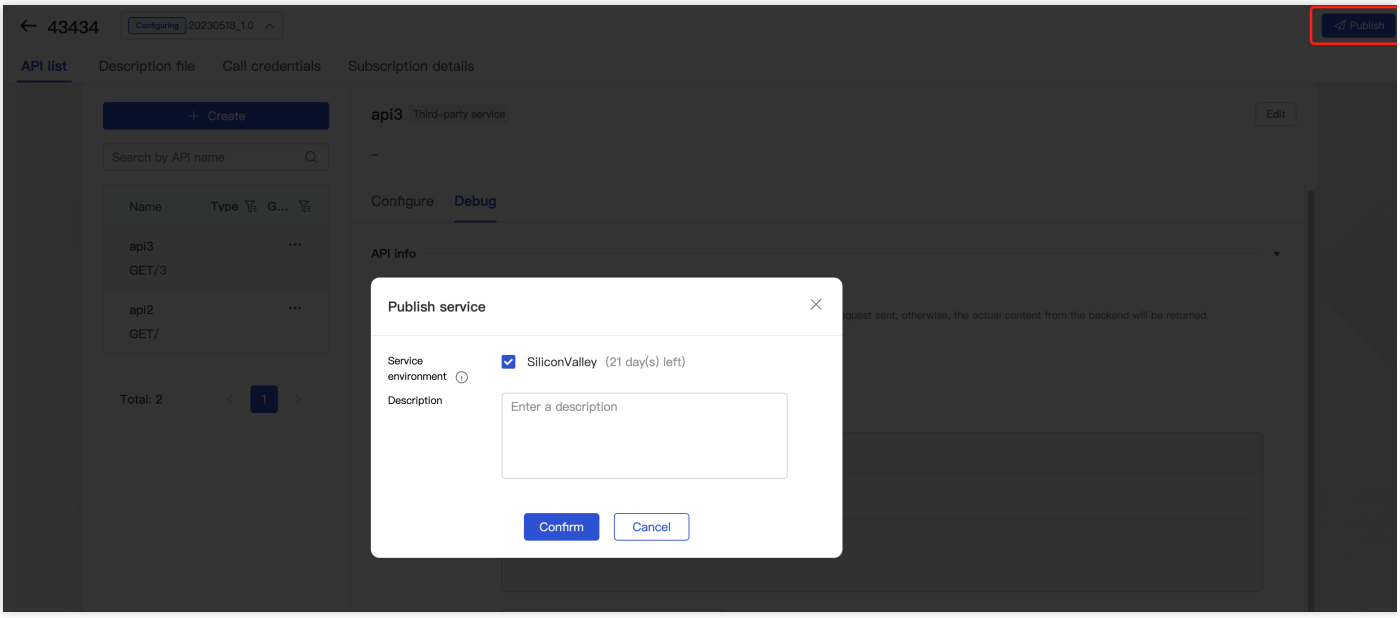


The left side is the API list, and the default tab page on the right side is the detailed configuration information of the API: API access path, request method, parameters, backend service type, etc. can be viewed here.

1. Click the **Debug** tab page to enter the API debugging page. On the API debugging page, you can configure the request Header and Body content of this API Endpoint, and click **Send Request**.



2. The test results are then available. We will return the Response status code and result returned by the backend service to the user for further debugging.
3. Click **Publish** in the upper right corner to publish this API to the corresponding environment.

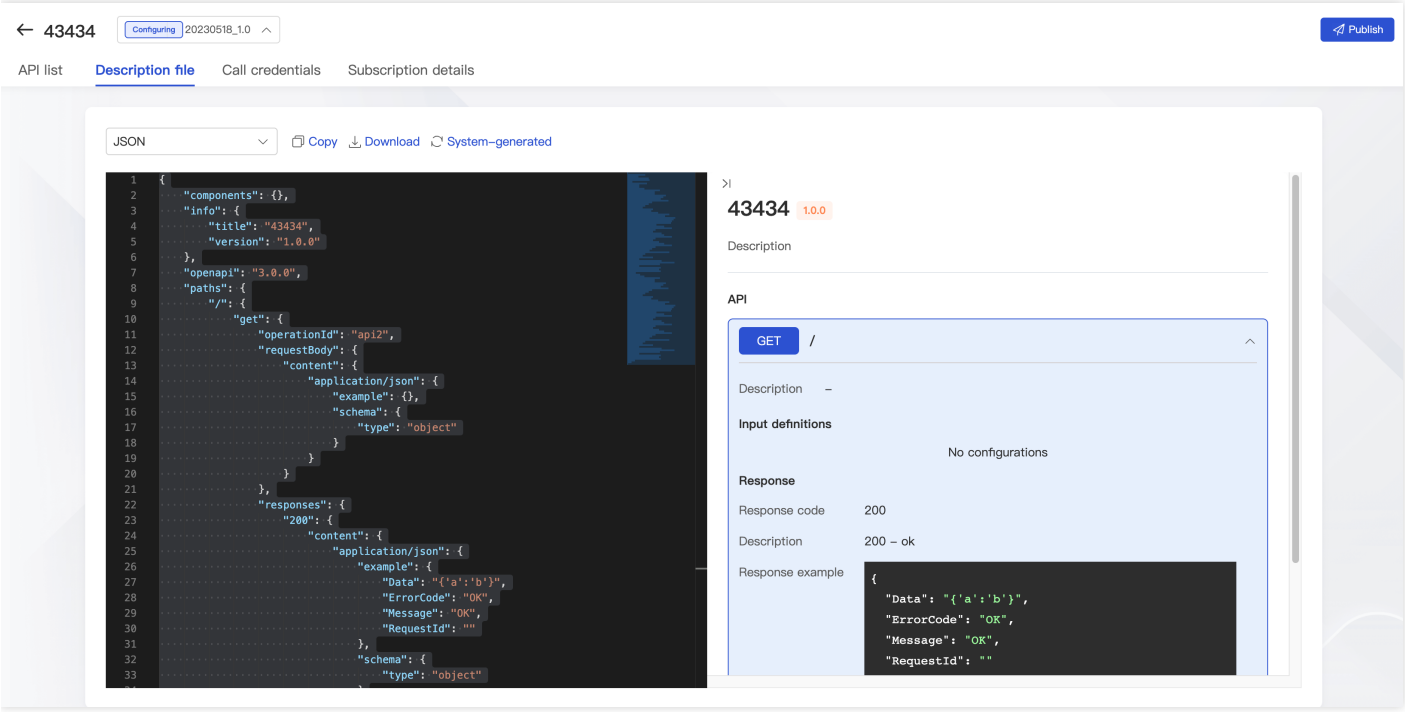


4. After release, the API status after release is **Running**.
5. **Copy** in the upper right corner can overwrite the current version to the version in the configuration. After replication, the API service can be published again. An API service can be published to multiple environments.

6. You can view its logs and monitoring on the API details page. For detailed information on logs and monitoring, see Operation and Maintenance Center-Monitoring Management and Operation and Maintenance Center-Running Log.
7. After the API service is released, the service can be stopped.

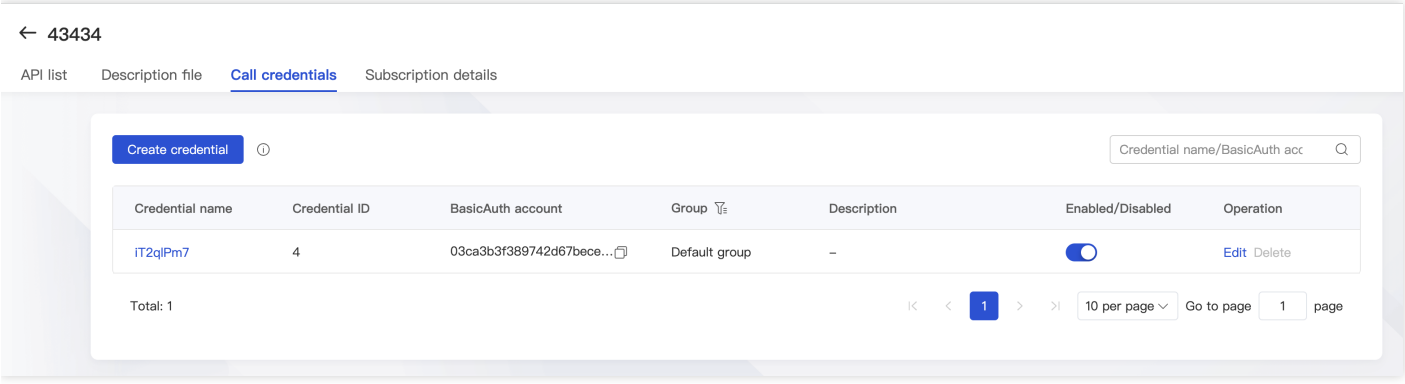
Description file

The description file is a description for the current API service. The YAML/JSON format file is displayed on the left, and the Swagger visualization content is displayed on the right.



Call Credentials

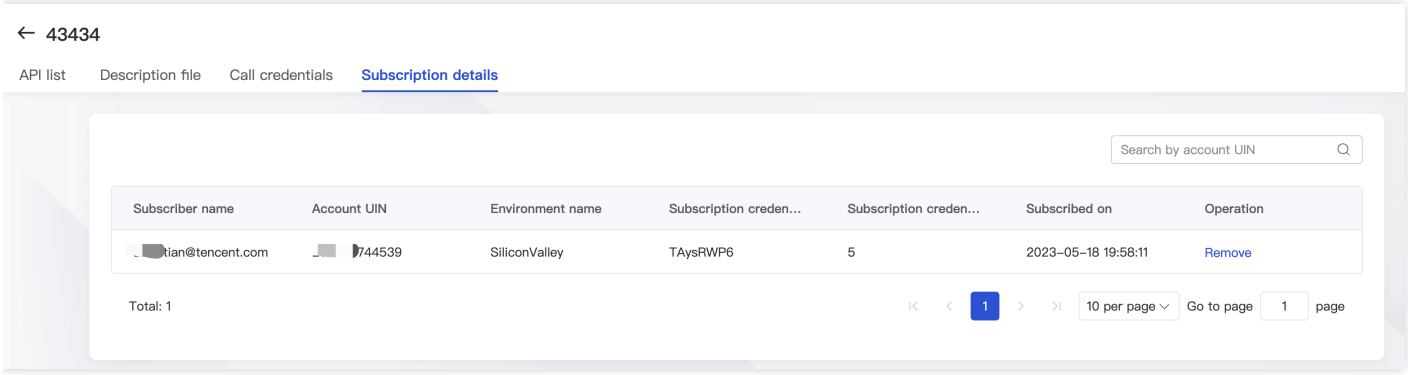
When creating an API service, if the selected authentication strategy is NoAuth, this option can be ignored. Conversely, if the API service requires authentication, you need to configure the calling credentials on this page. With any calling credentials under the current service, you can call any API under the service.



The above picture is the certificate list page, the created certificate will be displayed here, select **New Credential** to create a new certificate. Customize the credential information and save it.

Subscription Details

The listed API service can be subscribed and invoked by all sub-UINs under the business owner's UIN. This menu allows you to view the status of the current API service being subscribed.

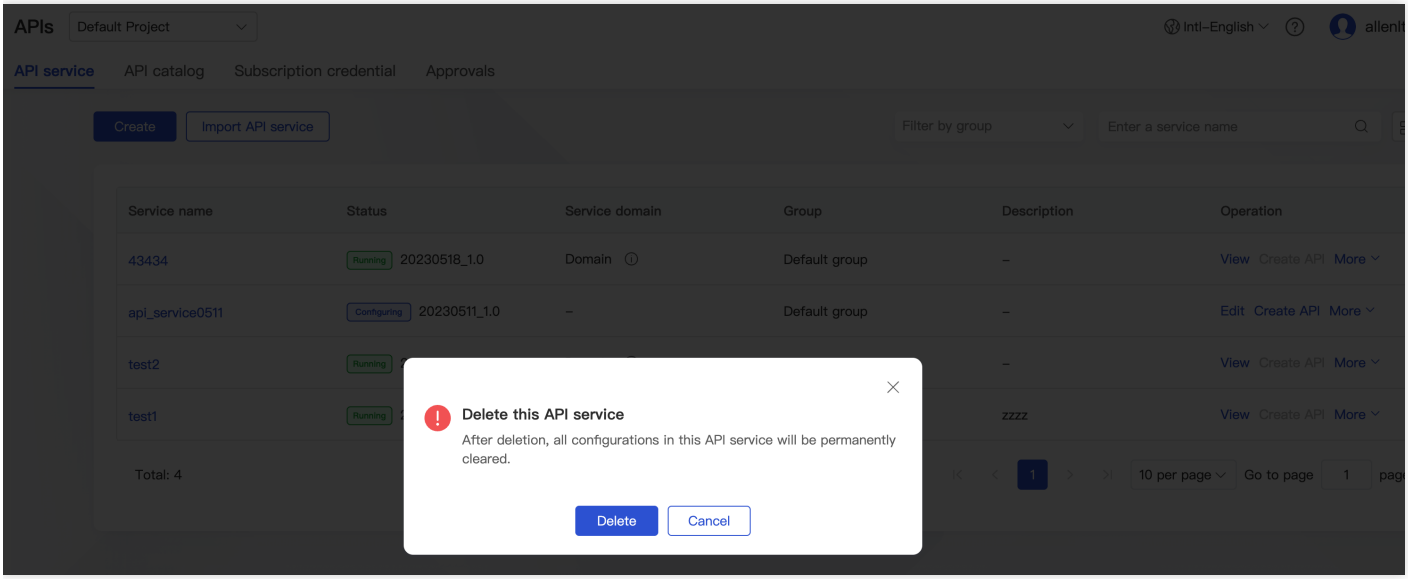


Here you can see a list of all users who have subscribed to the API, and at the same time, you can remove a user's subscription.

operation

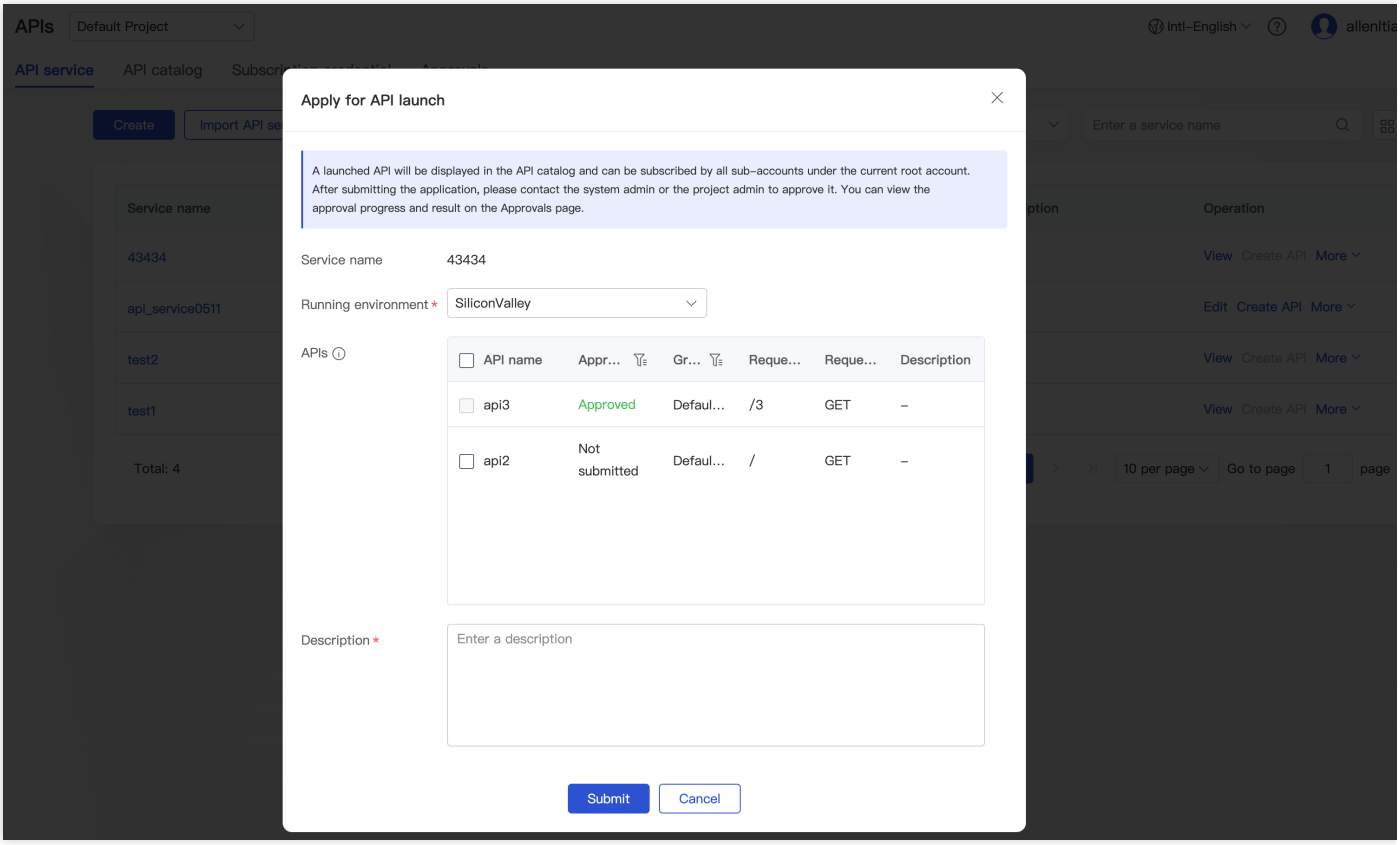
Delete API service

Click **Delete** to delete the current API service. After deletion, all configurations under the API service will be cleared and cannot be restored. The running service cannot be deleted directly, it needs to be stopped first and then deleted.



API Launch and Remove

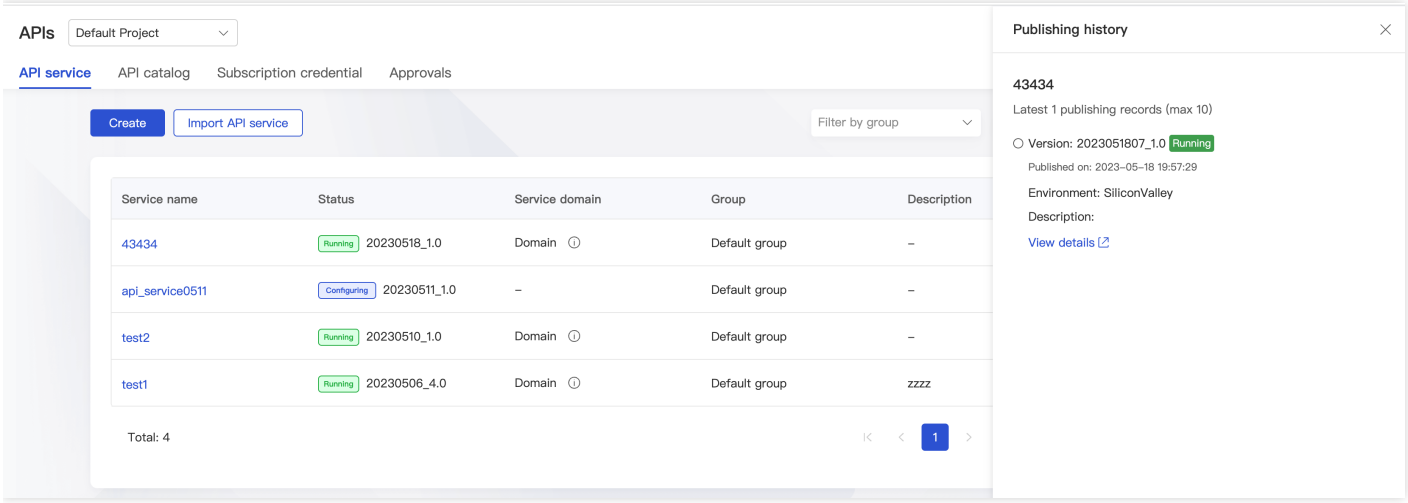
- Launch: The running API service can be shared with other employees of the enterprise through the **shelf** function. Submit the listing application at the **API Service** > **Operation** > **More** > **Launch** path. The submitted API service will be reviewed by the enterprise administrator. After the review is passed, it can be displayed on the API In the directory, it is supported to be subscribed by all sub-accounts under the current main account.
- Remove: After the API service is put on the shelf, if you do not want to continue to be subscribed by other employees, you can complete it by taking it off the shelf. Submit an application for removal from **API Service** > **Operation** > **More** > **Removal**. After submitting the application, you need to contact the system administrator or the project administrator of the project for review , it can be taken off the shelf after the review is passed, and the API cannot be subscribed after it is taken off the shelf. The delisted API service will be moved out of the API directory.



View release history

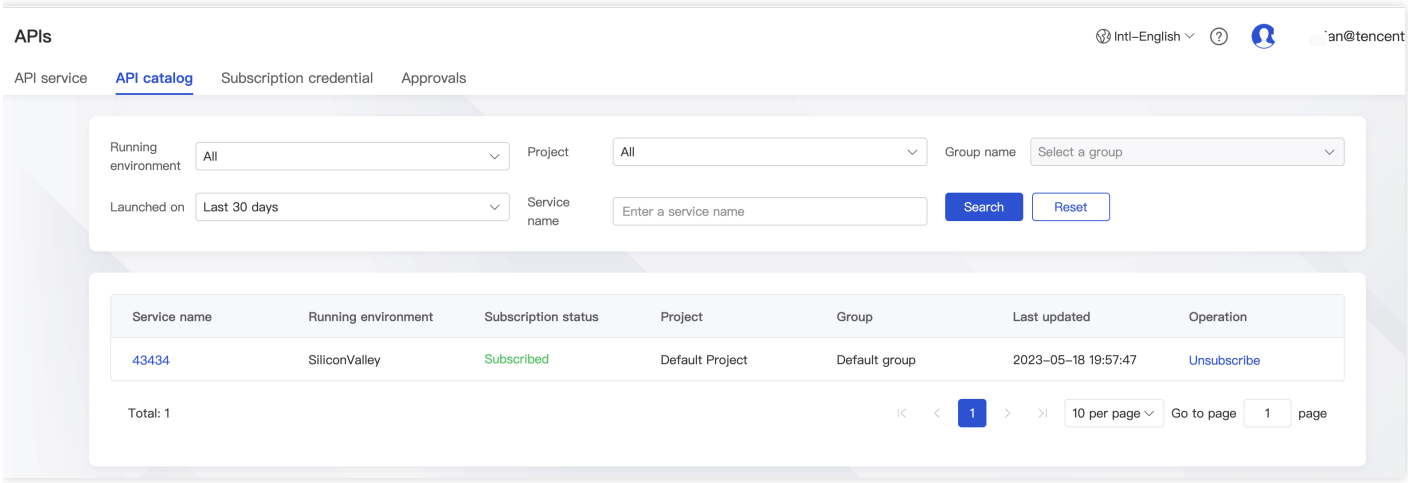
Published API services can change state, environment, etc. Go to **API Service** > **Operations** > **More** > **View Release History** path. This function can view the historical situation after the release of the API service (up to 10

items can be displayed).



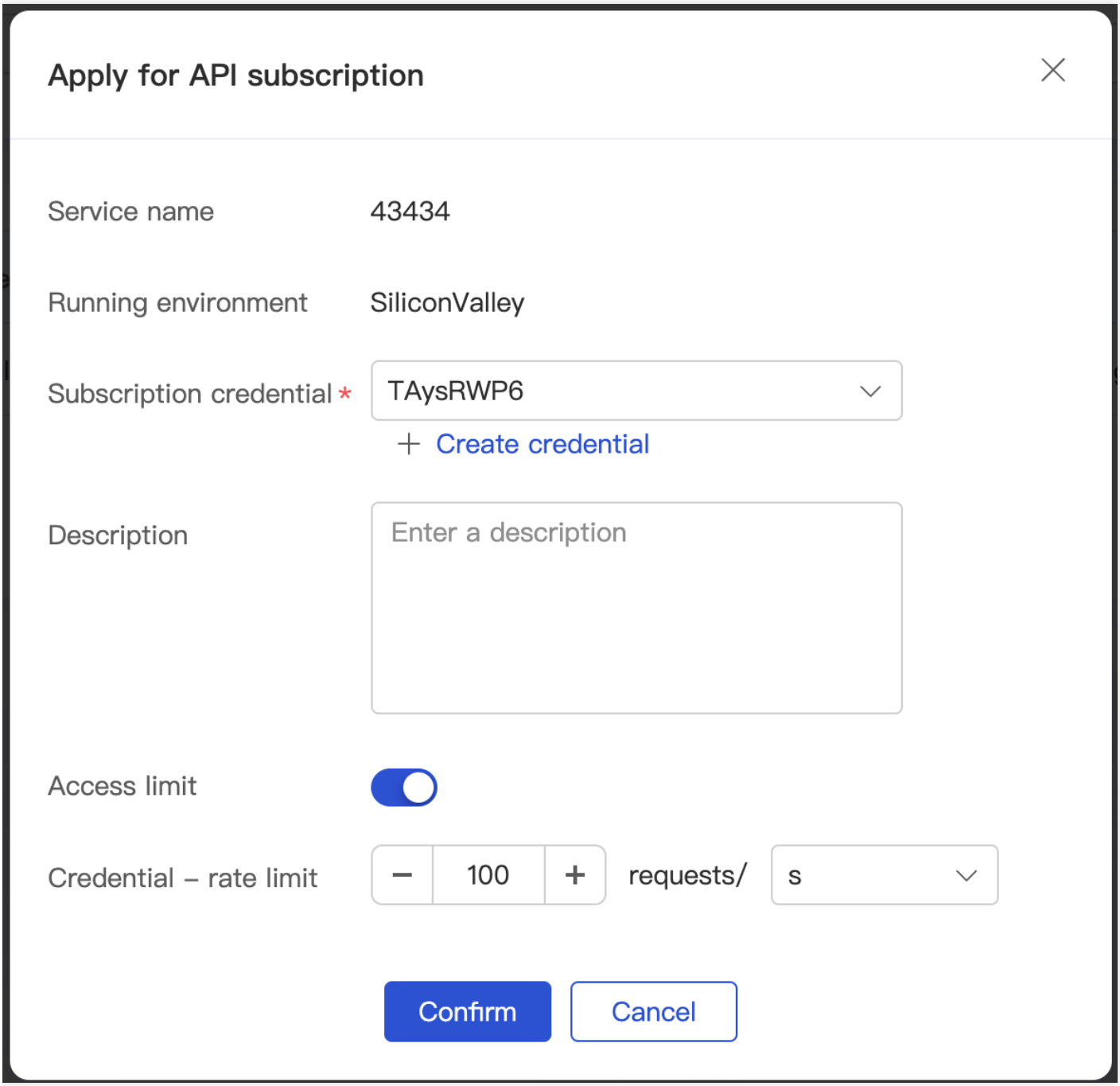
API catalog

The API catalog displays listed API services. Similar to an API service market, after the service is put on the shelf, it is not limited to the project dimension, and can be viewed, subscribed to and called by all sub-accounts under the current admin account. This page provides a quick search for services by their properties. At the same time, you can apply for subscription or unsubscribe API service.



Subscribe

When applying for API service subscription, you need to select or create a new subscription certificate. Associate the credentials with the API service. After being approved by the system administrator, you can successfully subscribe.



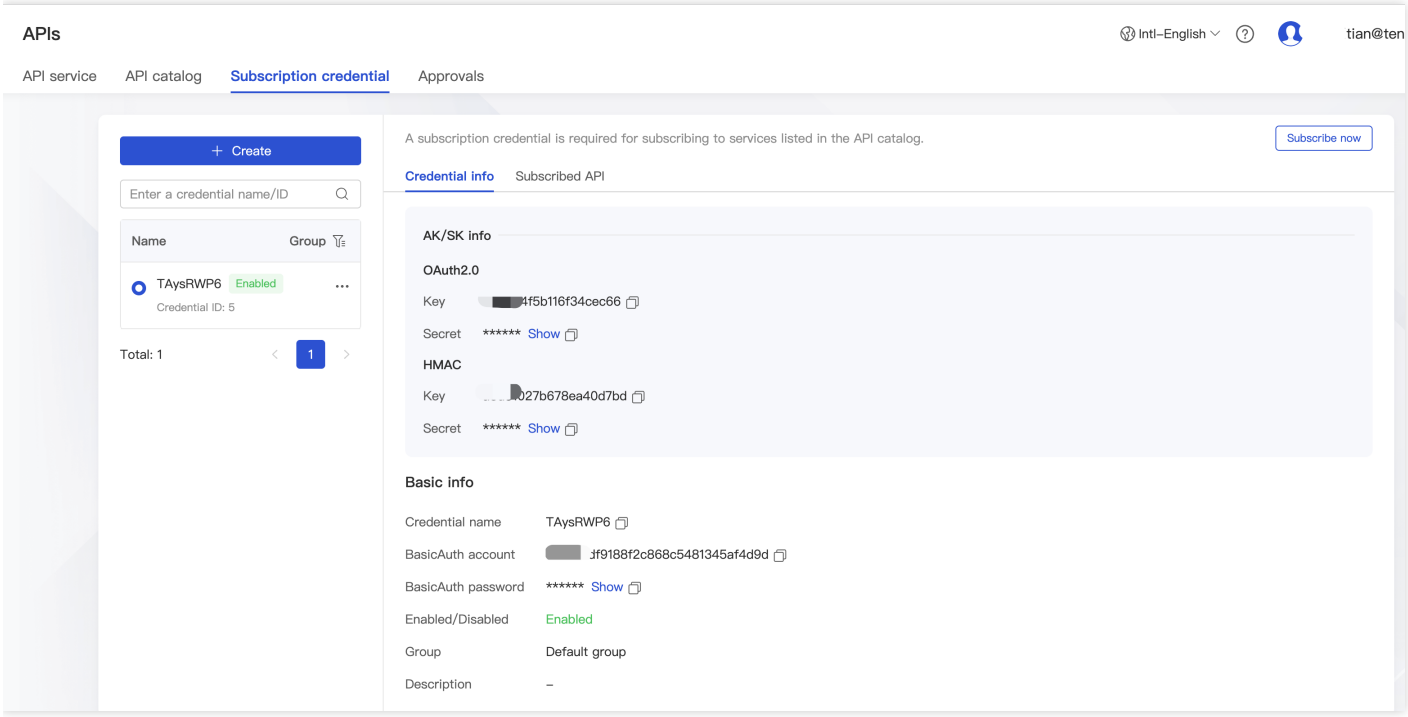
Unsubscribe

After canceling the subscription, the API service cannot be called, and this operation does not need to be reviewed by the system administrator.

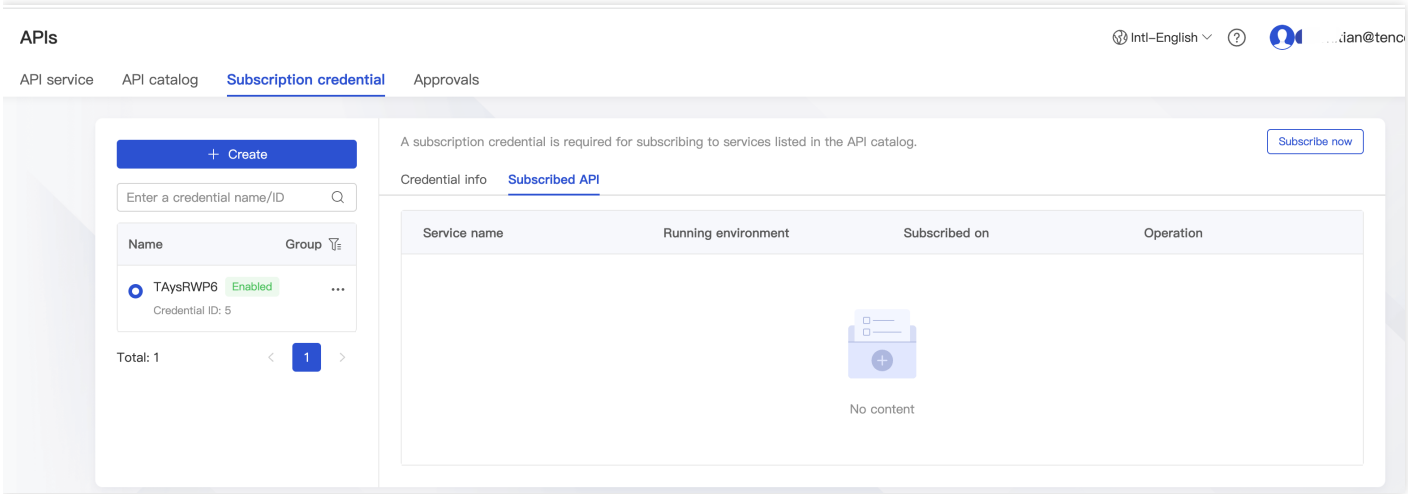
Subscription Credentials

This list can display or search for all subscription certificates, and at the same time, new certificates can be created. Subscription credentials are used to subscribe to services in the API catalog. Credentials are keys to an API service. When applying to subscribe to the API service, associate the credential with the API service, fill in the credential when

calling, and the service can be called successfully. At the same time, you can see the Key and Secret of various authentication types of the credential, which can be directly copied when calling.



One credential supports association with multiple API services. All API services associated with this credential can be viewed on the **subscribed API** tab page.

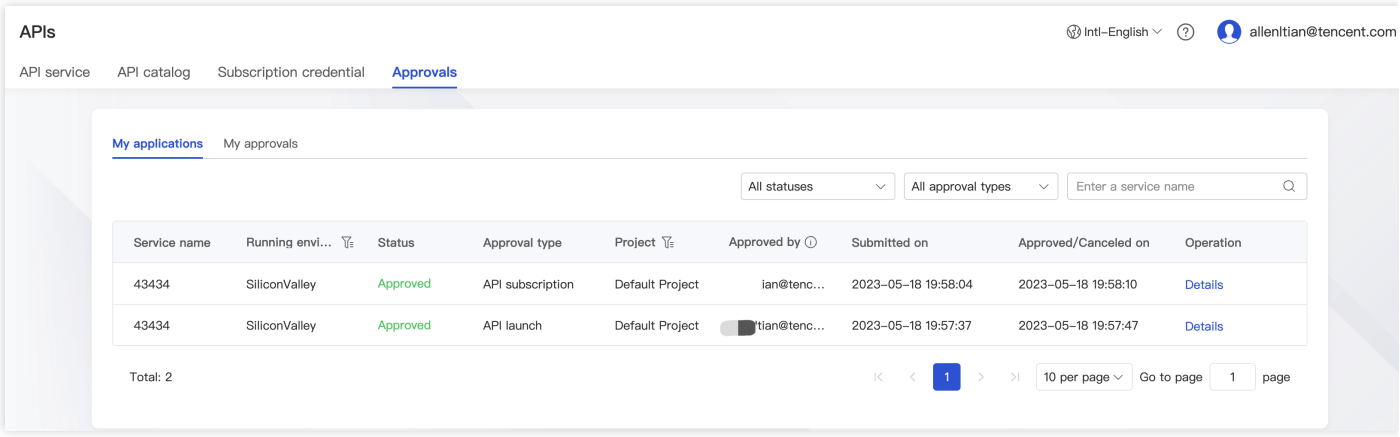


When creating a new credential, you can customize the relevant attributes.

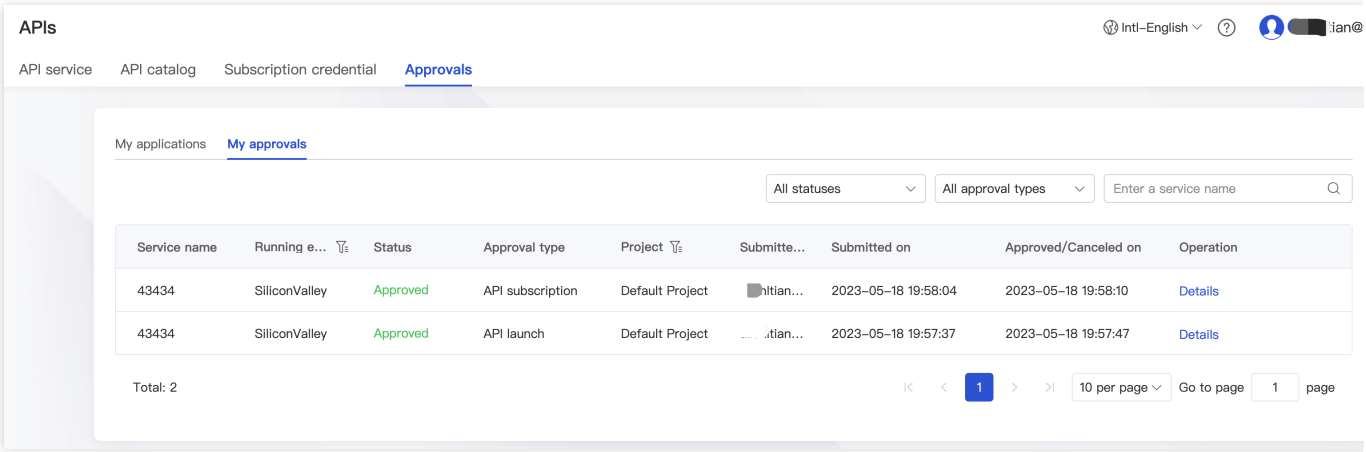
Approvals Management

Approvals management is divided into two functions: my applications and my approvals. Matters related to approvals are handled on this function page.

- Submitted by me: Displays all review information submitted by an individual. All characters are visible.



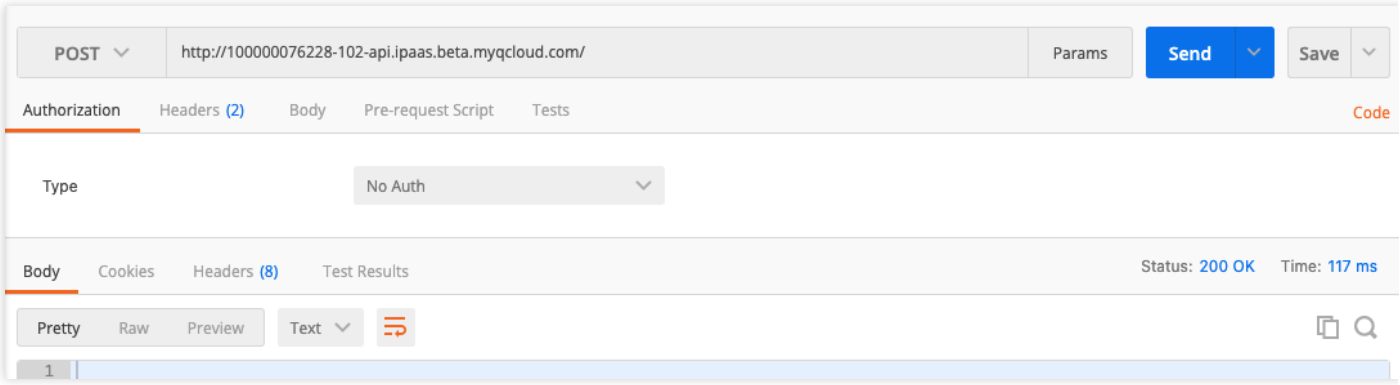
- I Reviewed: This page shows the information that needs to be reviewed. Visible only to System Admin and Project Admin roles. Other role access page data is empty.
  - The system administrator approves all requests for project API service delisting or API service subscription.
  - The project administrator approves the request for API service release or API service subscription within the project.



API call examples

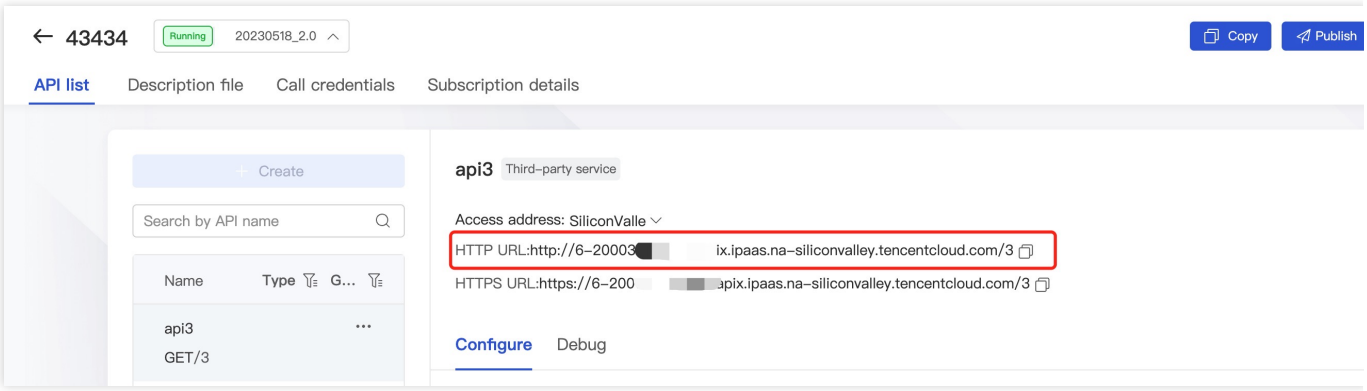
Call the API from the user side (take postman as an example)

- NoAuth:

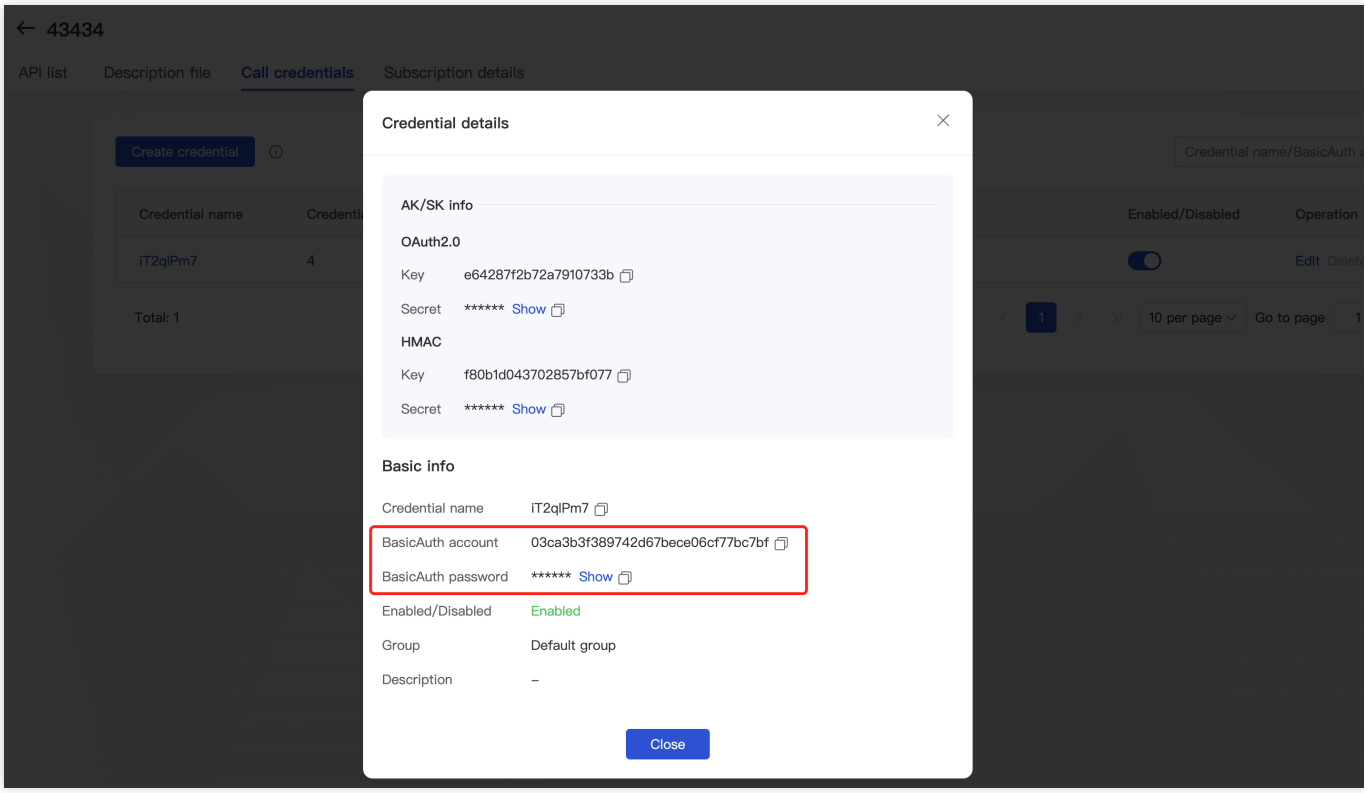


- BasicAuth:

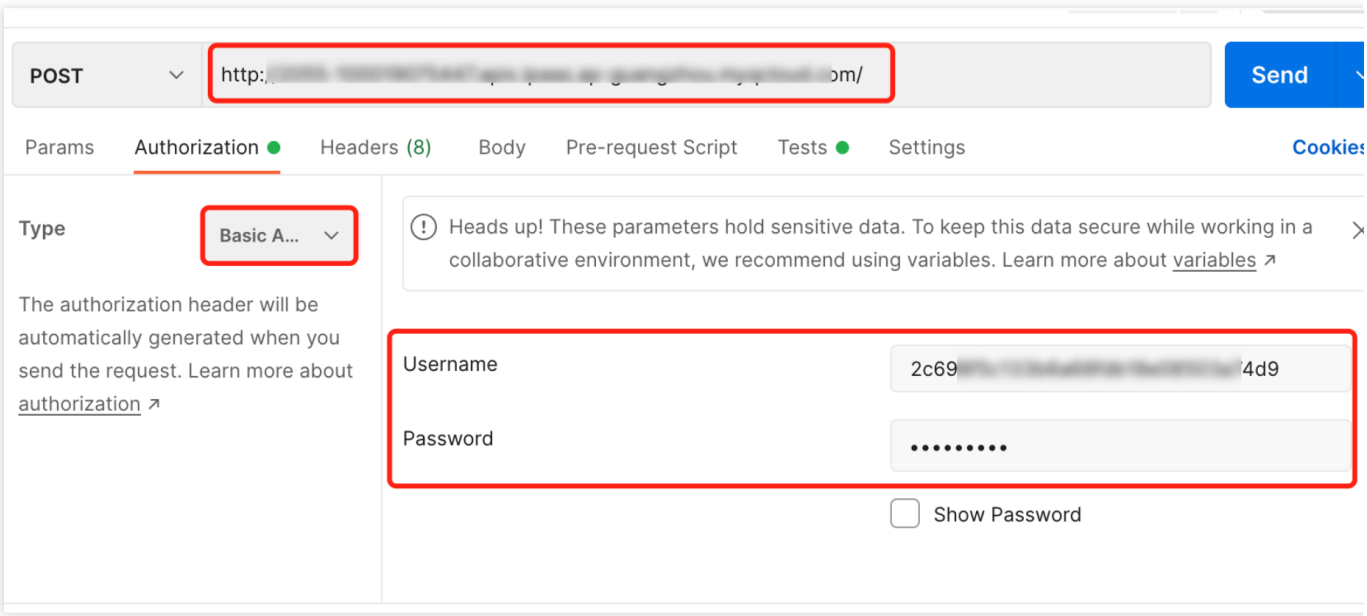
  i. Copy the calling address of the API (the API service must be successfully published first):



  ii. Enter the API list page, create or open the existing "Call credentials" to view the AK/SK information used for BasicAuth:



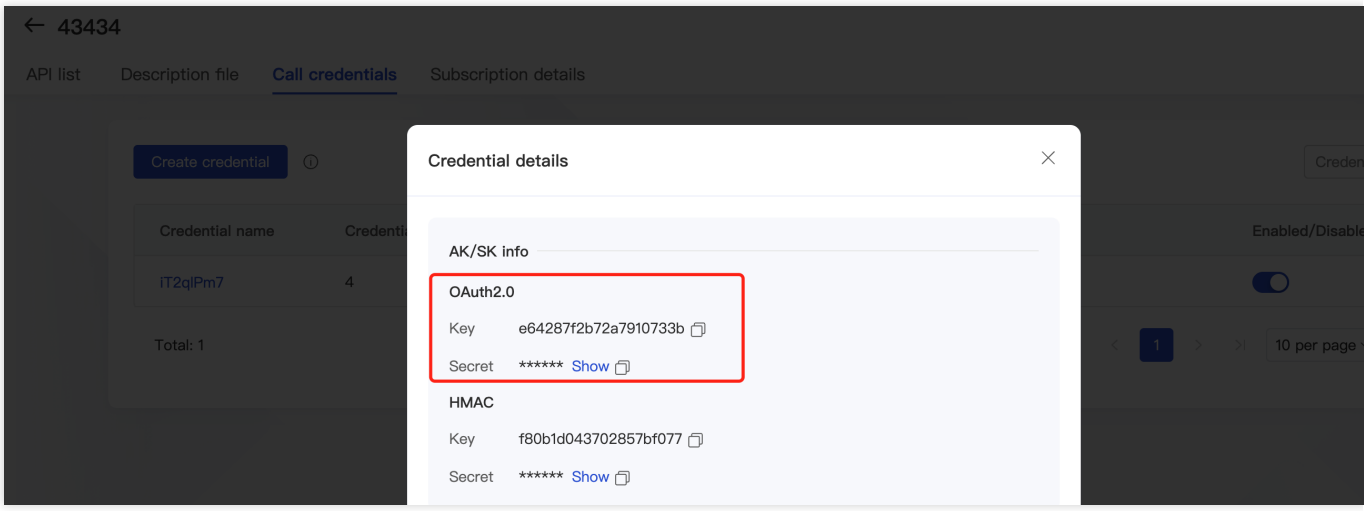
iii. Open postman, and fill in the API call address obtained above and the AK/SK information used for BasicAuth respectively:



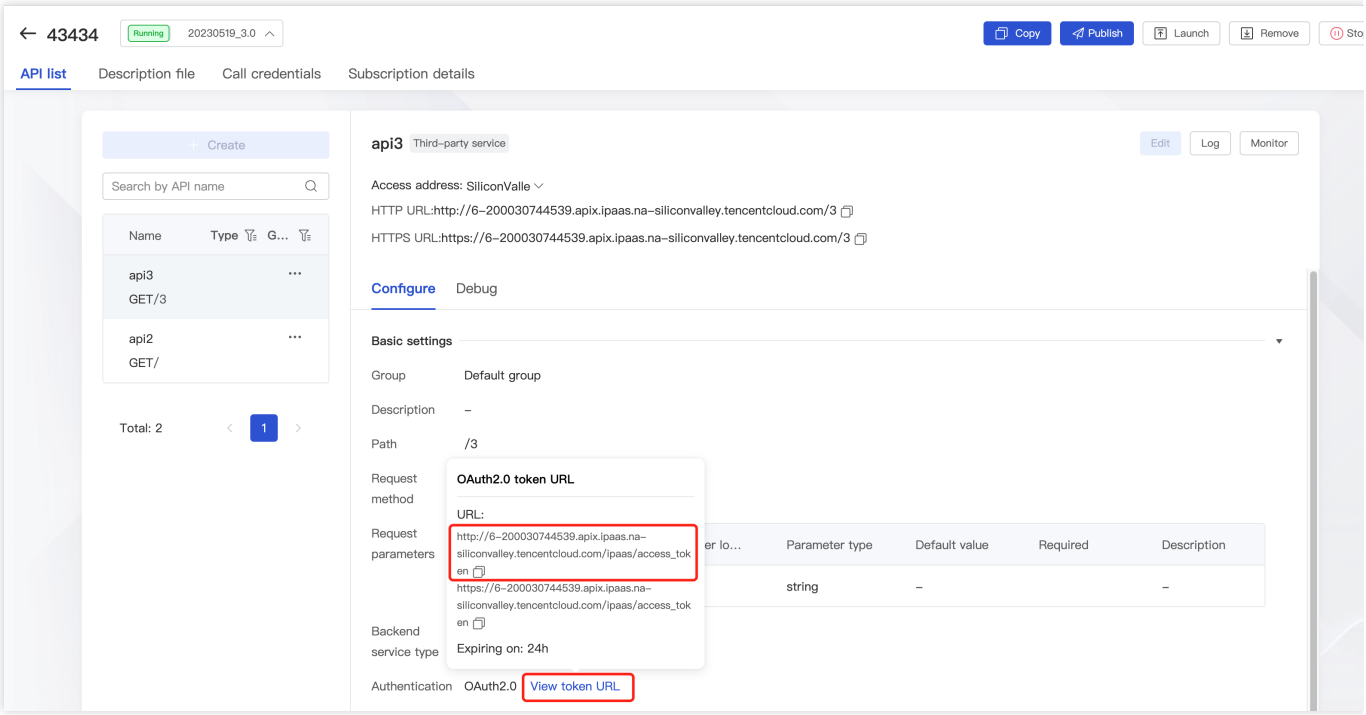
- OAuth2.0:

i. Copy the call address of the API (the API service needs to be successfully published first), the method is the same as above.

ii. Enter the API list page, create or open the existing "Credentials" to view the AK/SK information for OAuth2.0:



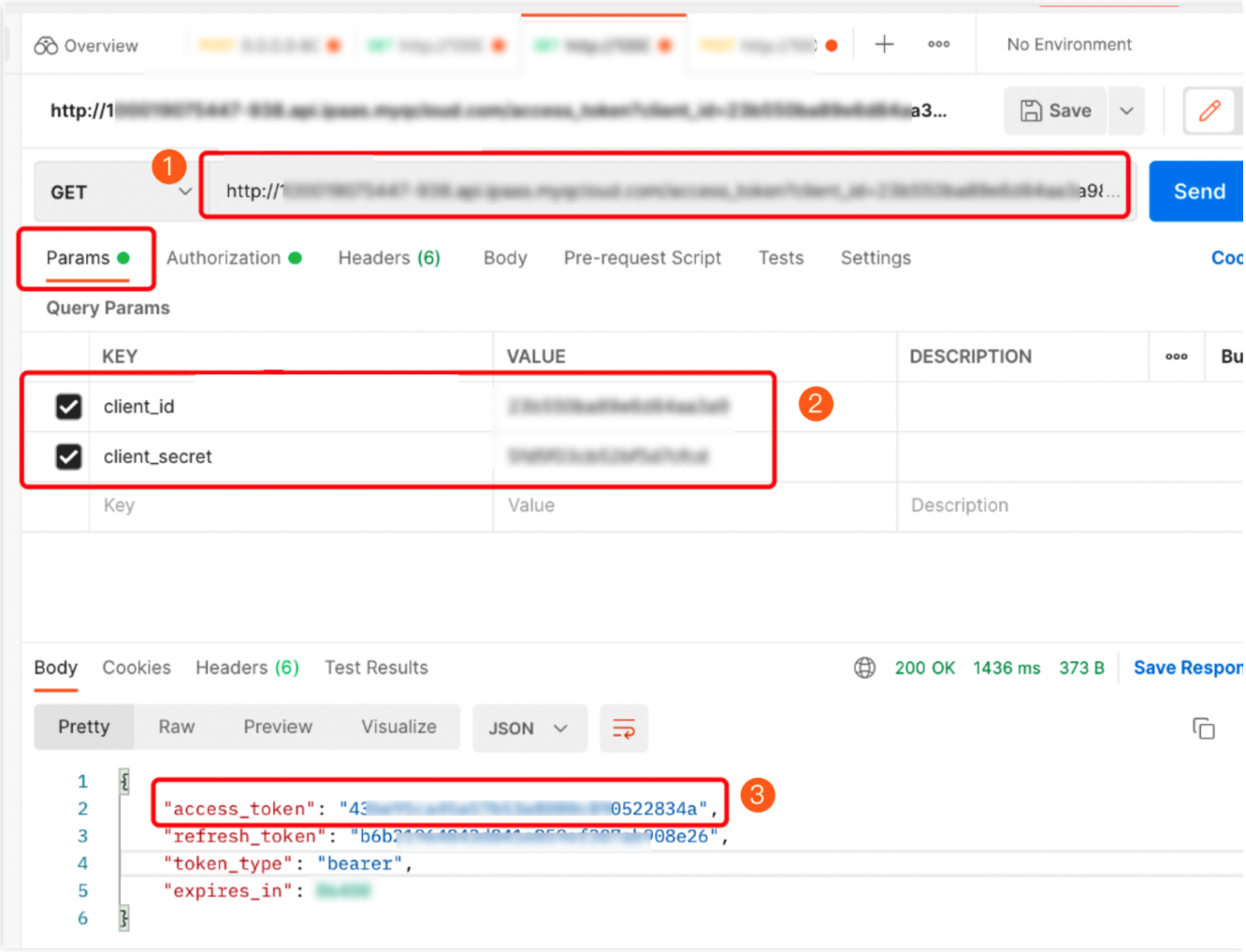
iii. Enter the API list page, click and copy the "View token URL" corresponding to the API.



iv. Create a new request in postman to get accsee_token.

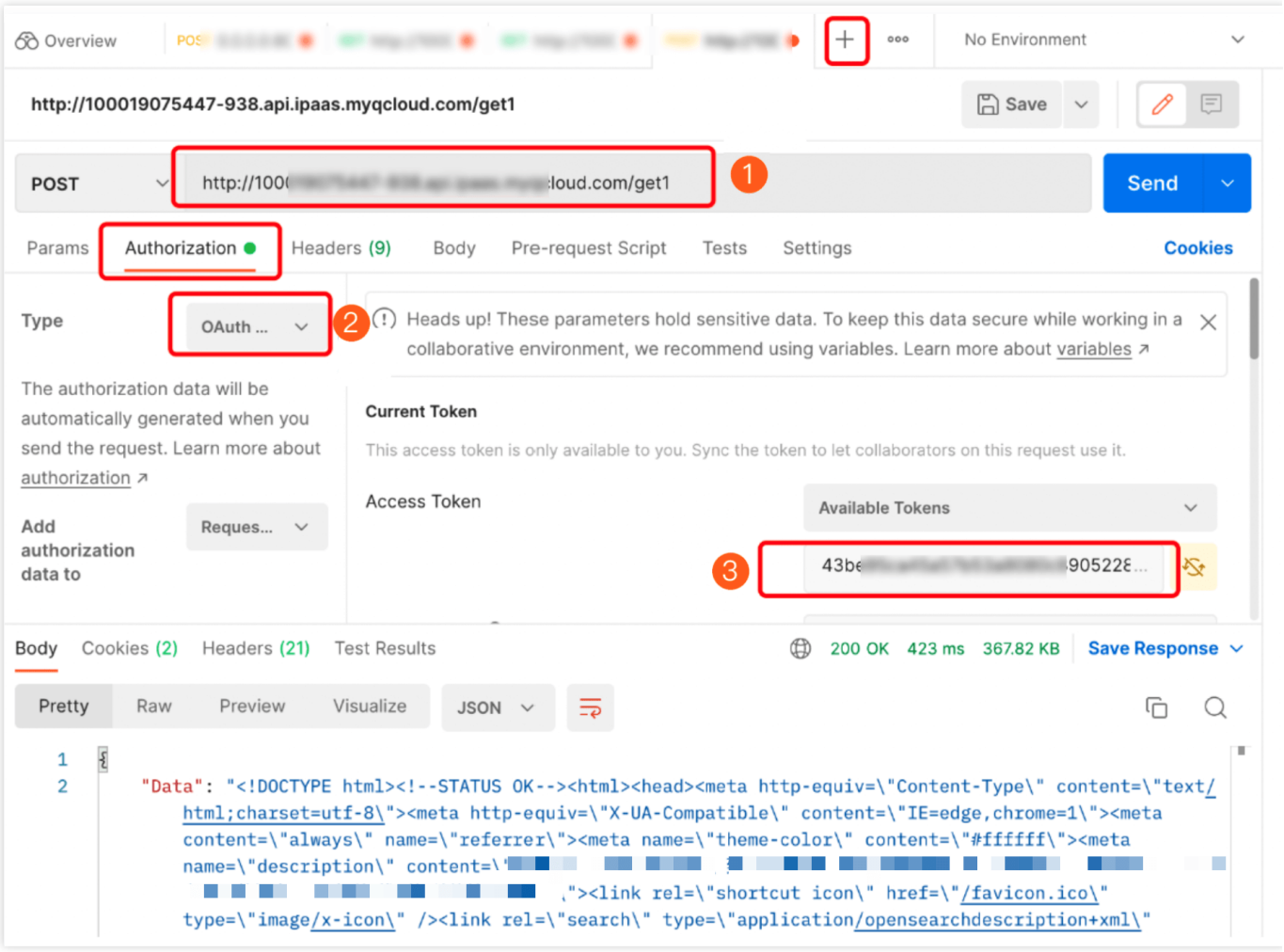
- First, enter the "Token acquisition link" obtained in step 3 in the input field, and select the GET request method.
- Next, select the Params tab, and enter the AK/SK information for OAuth2.0 obtained in step 2 (see the figure below for the format).

- Finally, click the **send** button to generate the "access_token" content in the figure below and copy it.



v. Re-open a request interface in postman, fill in the API call address obtained in step 1, and select OAuth2.0 for Type. And fill in the accsee_token obtained in step 4 on the right, and click **send** to see the access result (if you

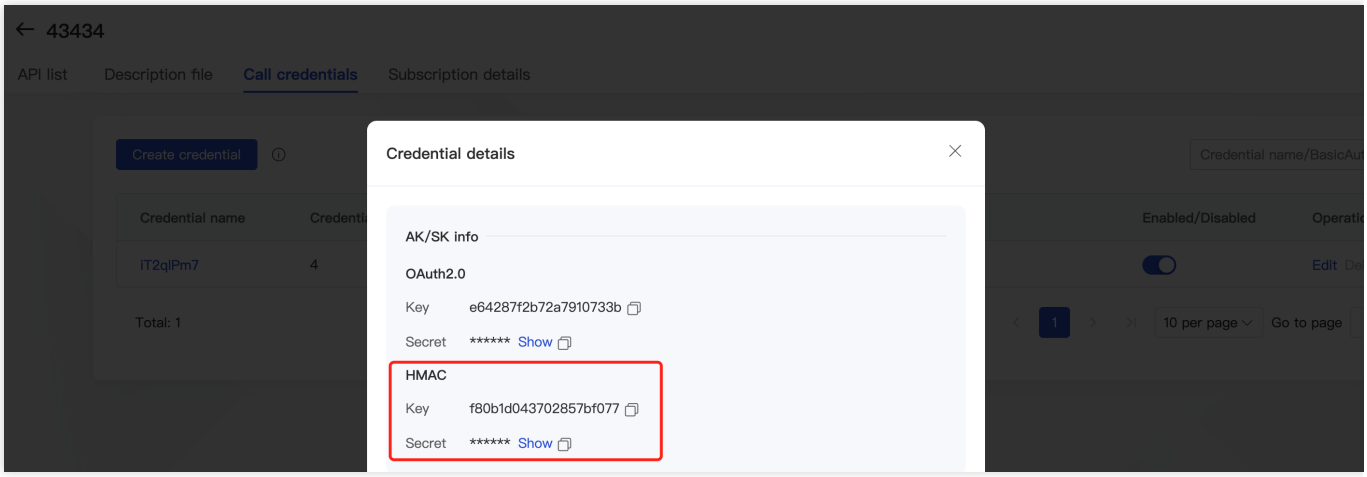
also set request parameters when configuring the API, you must also enter it here by location).



- HMAC:

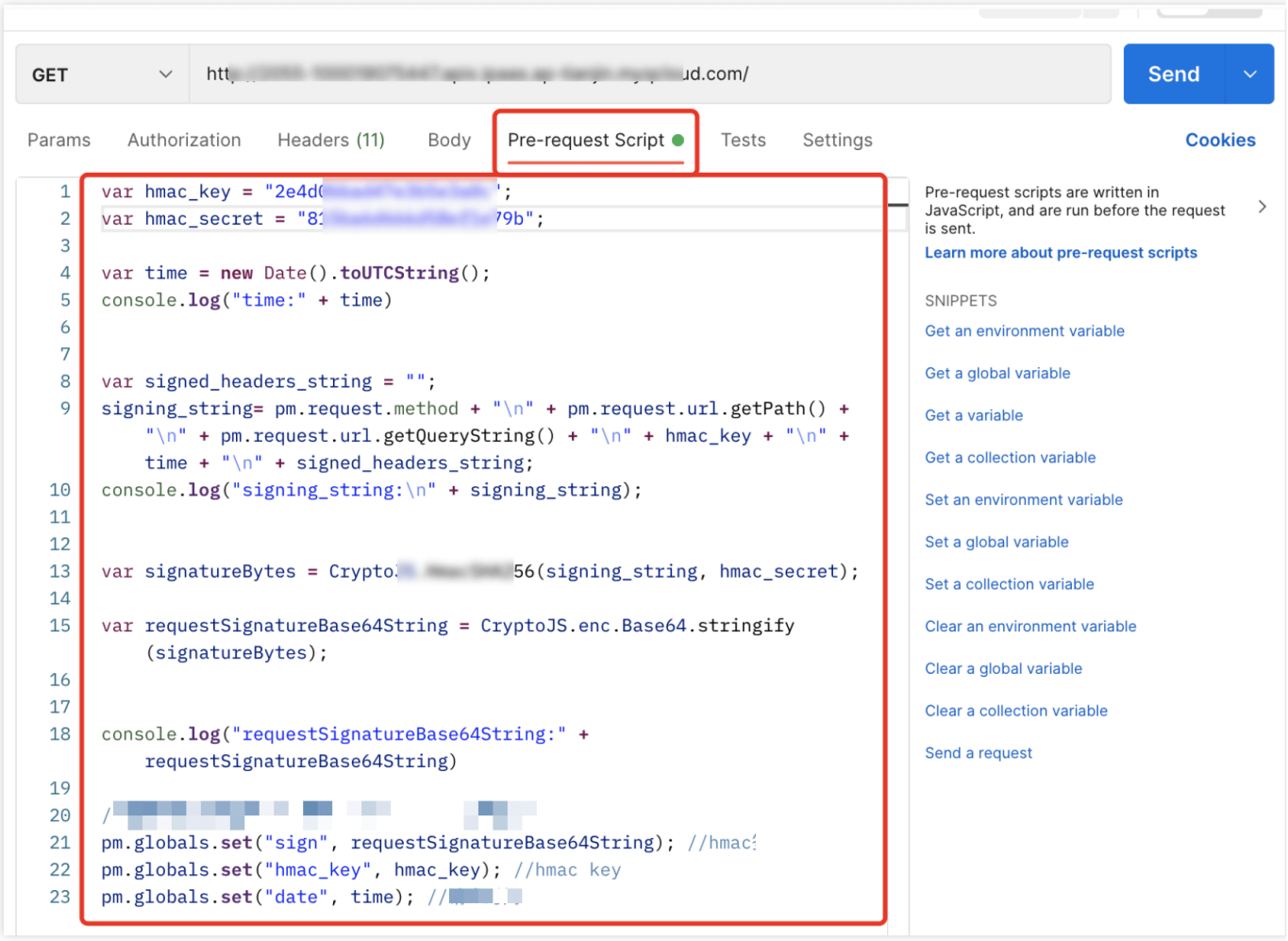
  i. Copy the call address of the API (the API service needs to be successfully published first), the method is the same as above, no more drawings.

  ii. Enter the API service details page, create or open the existing "Call Credentials", and you can view the AK/SK information used for HMAC:



iii. Open postman, and fill in the API call address obtained in step 1 in the input box.

iv. Switch postman to the Pre-request Script tab, and paste the following code segment (note that the HMAC Key and Secret obtained in step 2 are used to replace the hmac_key and hmac_secret variable values in the code segment).

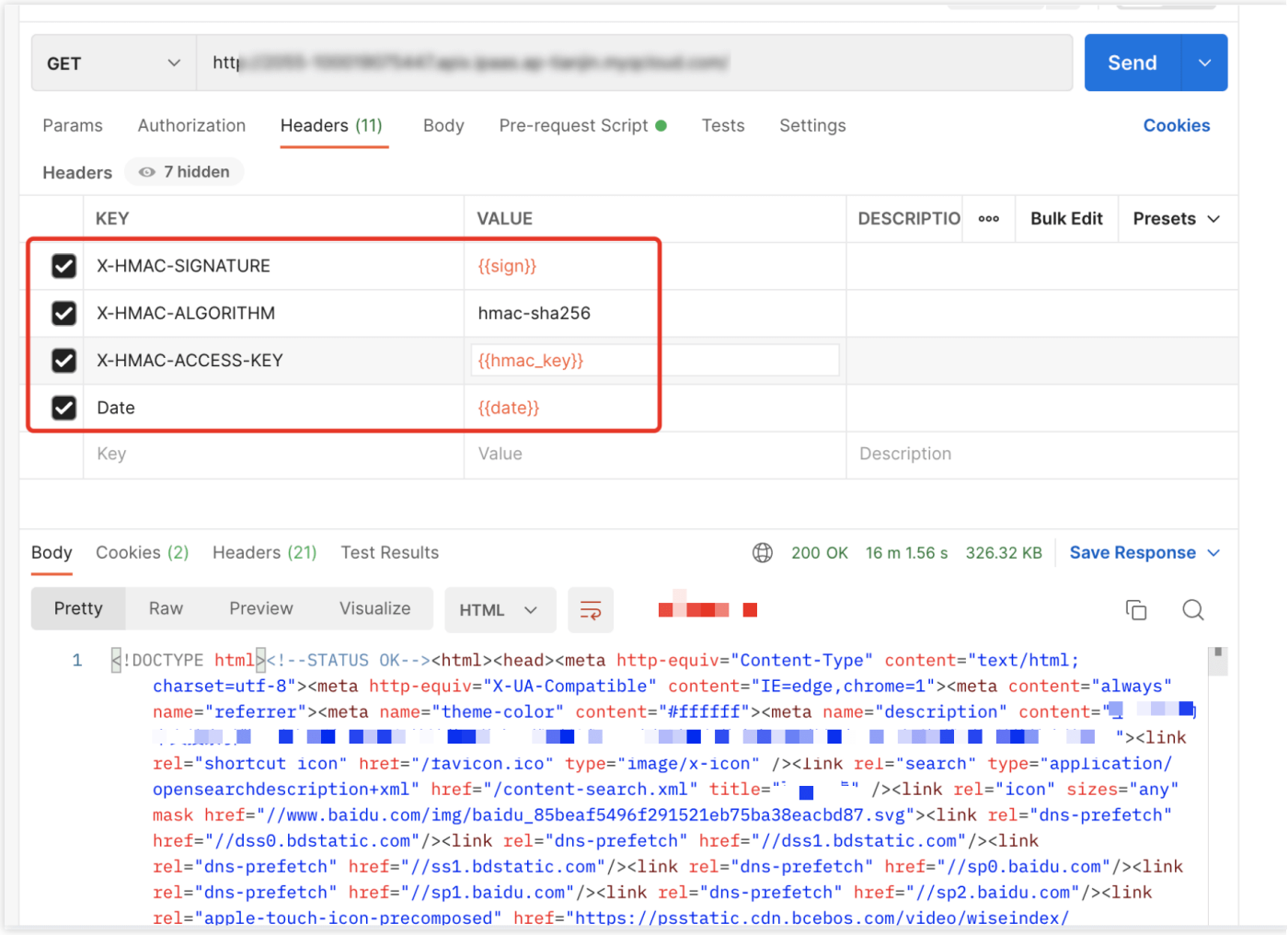


```
var hmac_key = "2e4d0bbad47e3b5e3a0c";
var hmac_secret = "815ba6d666d58ef1e79b";
var time = new Date().toUTCString();
console.log("time:" + time)
var signed_headers_string = "";
signing_string= pm.request.method + "\n" + pm.request.url.getPath() + "\n" + pm.r
equest.url.getQueryString() + "\n" + hmac_key + "\n" + time + "\n" + signed_heade
rs_string;
console.log("signing_string:\n" + signing_string);

var signatureBytes = CryptoJS.HmacSHA256(signing_string, hmac_secret);
var requestSignatureBase64String = CryptoJS.enc.Base64.stringify(signatureBytes);
console.log("requestSignatureBase64String:" + requestSignatureBase64String)

//used in Header
pm.globals.set("sign", requestSignatureBase64String); //hmac signature
pm.globals.set("hmac_key", hmac_key); //hmac key
pm.globals.set("date", time); //request time
```

5. Switch postman to the Headers tab, and enter the 4 KEY-VALUE pairs in the figure below. Finally, click the **send** button to see the return result of calling the API.



Ops Center
Monitoring Management

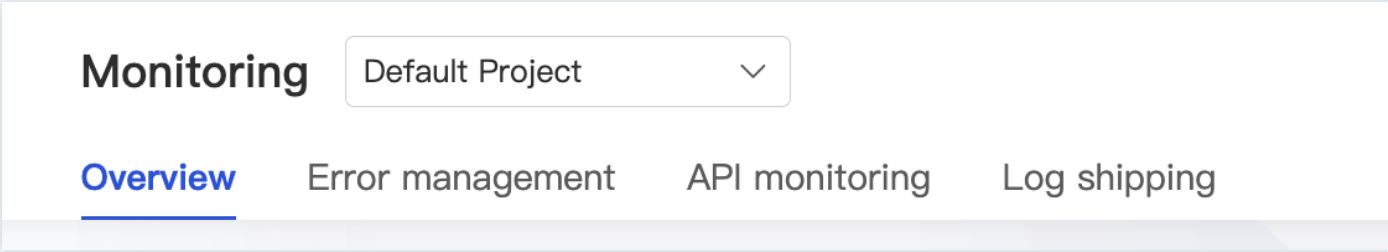
Last updated：2023-08-03 17:24:02

Overview

The monitoring management page displays the basic operation overview and reported error statistics for integration apps, flows, connectors, and components. On this page, you can view the real-time and historical monitoring data of a flow in a certain period of time.

For the descriptions of monitoring metric parameters, see Monitoring Metric Description.

Note：
Currently, iPaaS monitoring management is categorized by project. To view the monitoring data, select a project first.



Directions

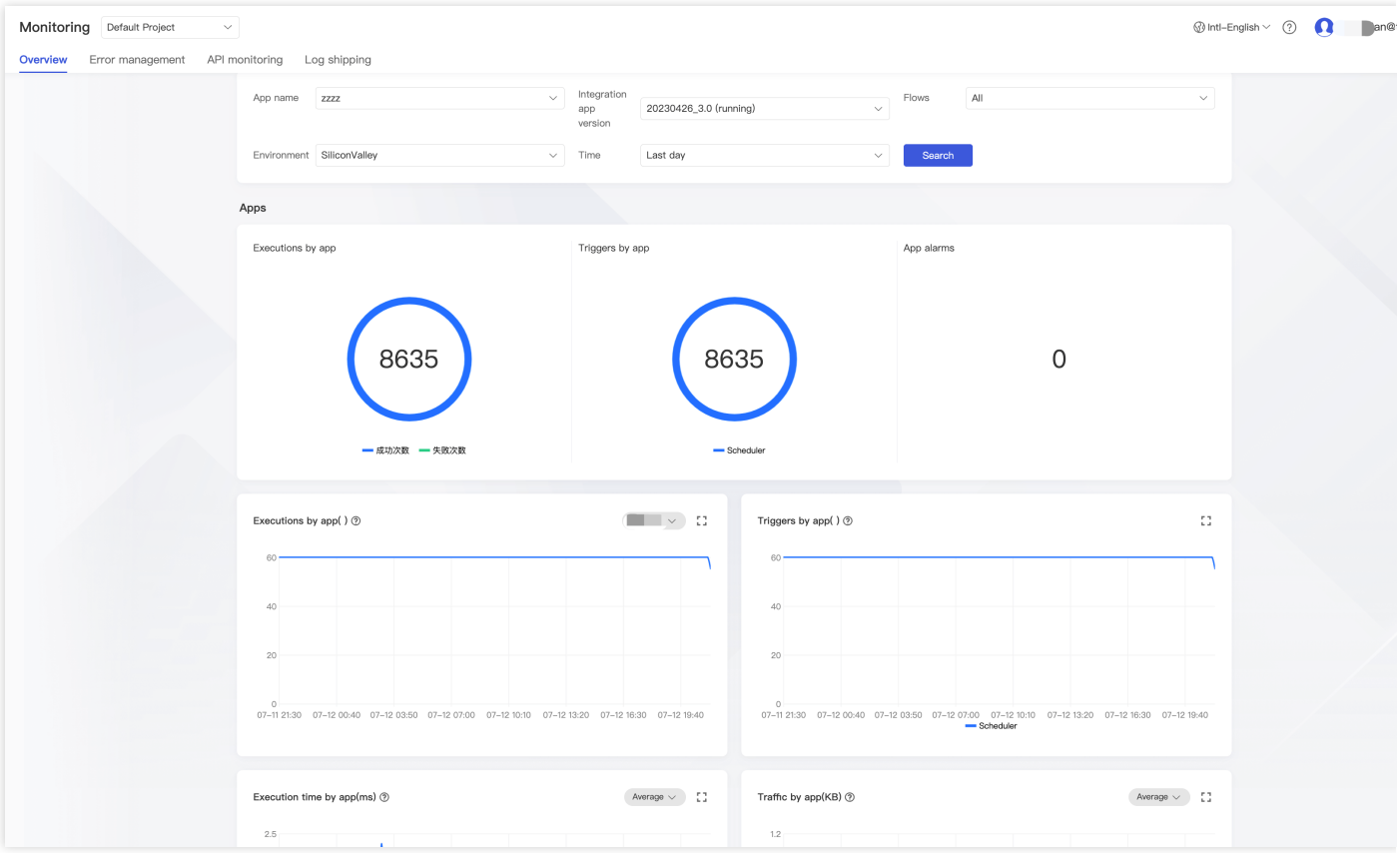
Monitoring management

You can view the real-time and historical monitoring data and analyze the flow execution based on the data. The monitoring overview of the integration app in the last 24 hours is displayed by default. To view the historical data, you can customize the time period, select an app or flow, and click **Search**.

Note：
Currently, you can search for the monitoring data in the last seven days in the console. To search for earlier data, please submit a ticket for assistance.

Querying integration app overview data

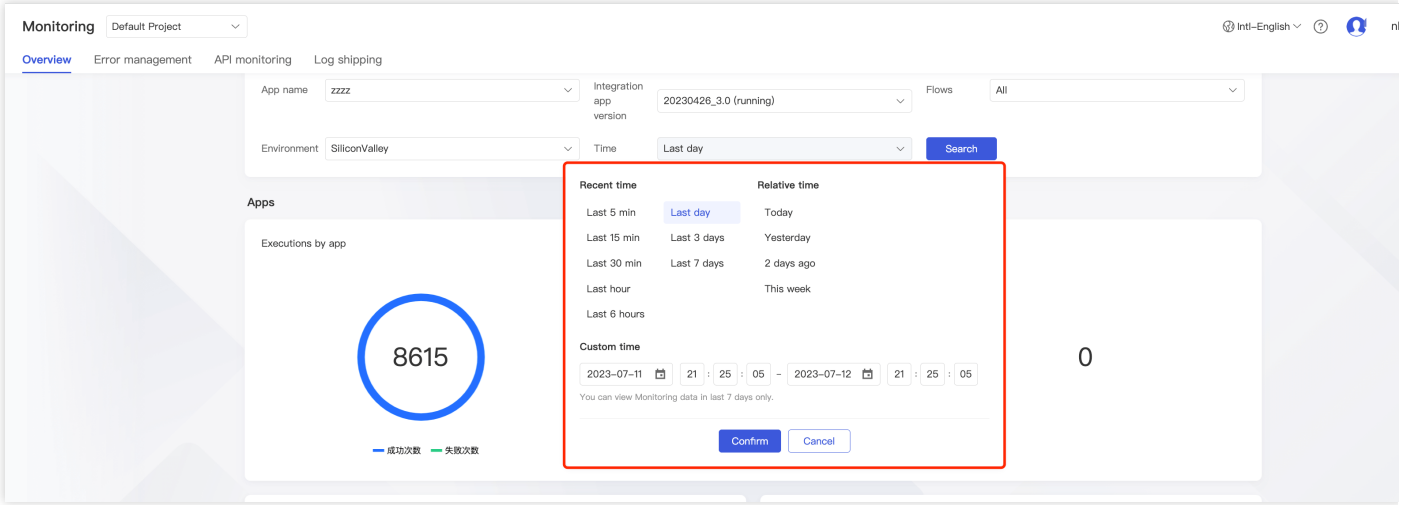
1. Log in to the iPaaS console and select Monitoring > **Overview**.
2. On the **Overview** tab, **All flows** is selected by default, that is, the app overview data is displayed by default. You can view metrics such as the number of app triggers, executions, and alarms, app execution duration in milliseconds, and app traffic usage in KB.



Description of app monitoring data search: You can search by app name, version, flow (**All flows** must be selected), runtime environment, and time.

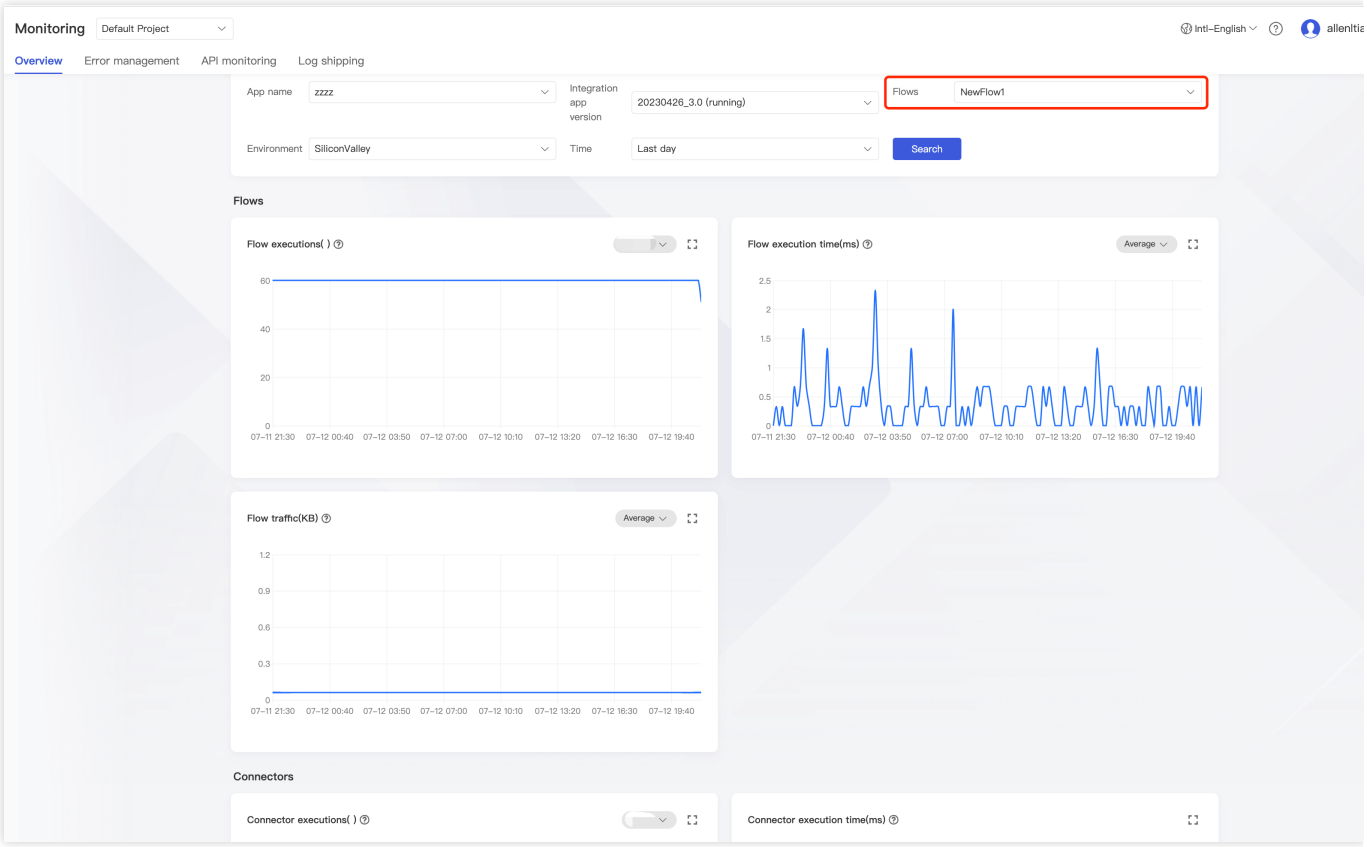
Click the time drop-down list, and the panel for setting the time period for search will be displayed. You can select the last 5 minutes, last 15 minutes, last hour, last 6 hours, last day, last 3 days, last 7 days, today, yesterday, 2 days ago, this week, or a custom time period. You can preset a frequently used time period to quickly filter the

monitoring data within it.



Querying flow data

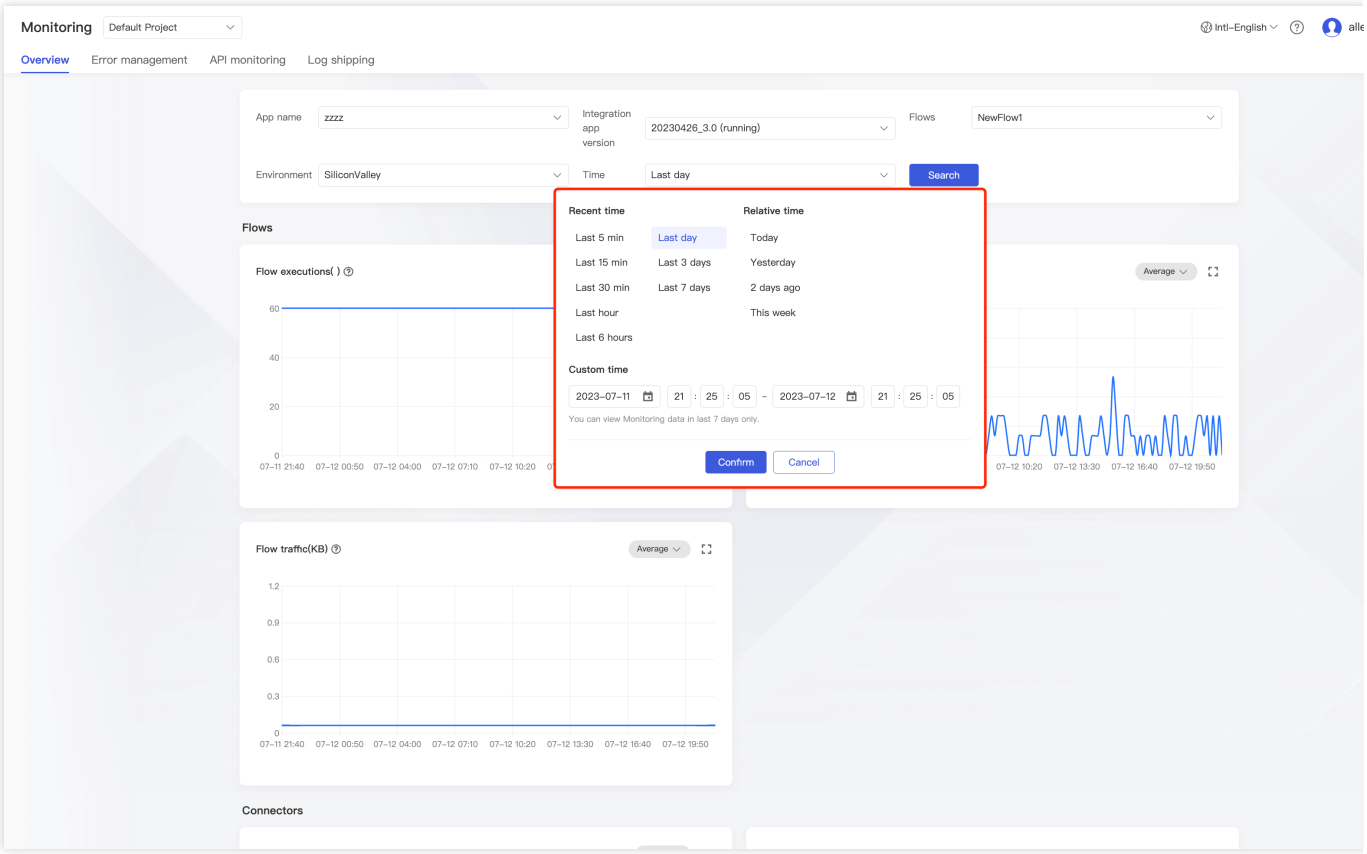
1. Log in to the iPaaS console and select Monitoring > **Overview**.
2. On the **Overview** tab, select a flow of the integration app to query the flow and connector overview data.
   - Flow overview: You can view flow metrics such as numbers of flow executions (including the total number of executions, number of successful executions, and number of failed executions), flow execution duration in milliseconds, and flow traffic usage in KB.
   - Connector overview: You can view connector metrics of a flow such as numbers of connector executions (including the total number of executions, number of successful executions, and number of failed executions), connector execution duration in milliseconds, and connector traffic in KB.



Description of flow monitoring data search: You can search by project, app, version, flow (required), region, and time to view the flow and connector overview of the specified flow.

Click the time drop-down list, and the panel for setting the time period for search will be displayed. You can select the last 5 minutes, last 15 minutes, last hour, last 6 hours, last day, last 3 days, last 7 days, today, yesterday, 2 days ago, this week, or a custom time period. You can preset a frequently used time period to

quickly filter the monitoring data within that period.



Error management

Note：

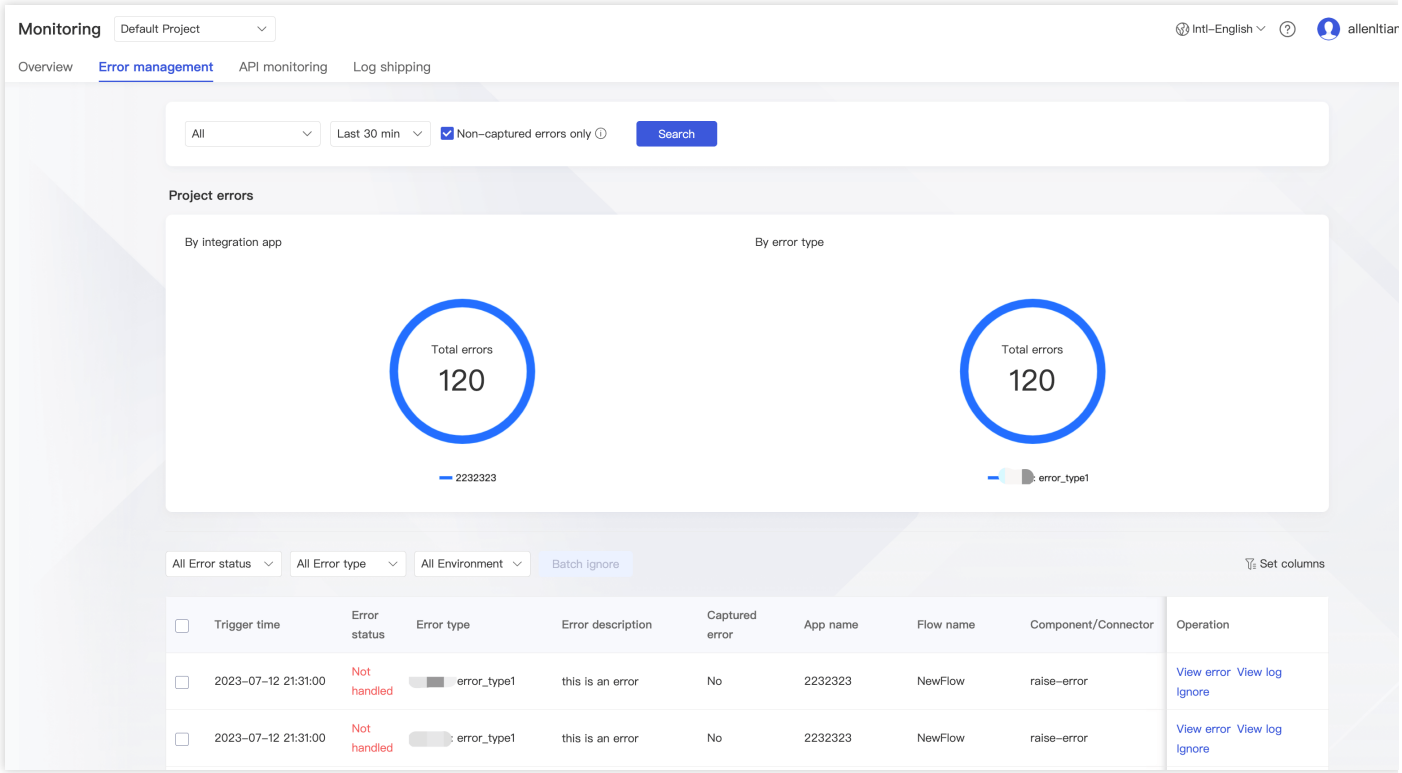
The console currently lets you search for error statistics only from within the last 30 days. To search for earlier statistics, please submit a ticket for assistance.

1. Viewing the error statistics list

You can centrally view data such as number of errors and error types when an app fails and quickly fix the errors. You can search by project, app, error status, environment/region, flow, error type, and time. Below is the list:

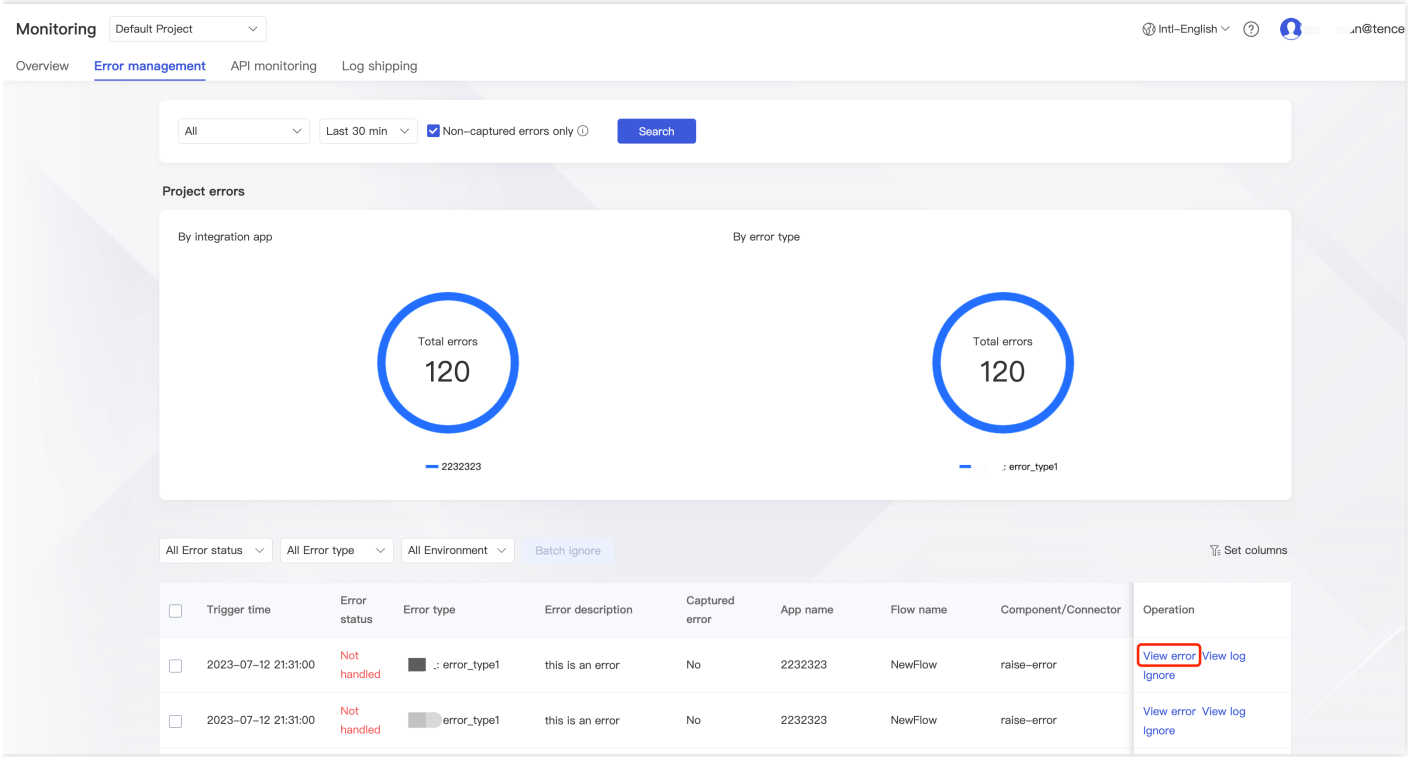
- App error statistics: Indicates the total number of errors of all flows in an app.
- Error type statistics: Indicates the numbers of errors of all types in an app failure.
- Error status: Only unfixed or ignored errors are displayed.
- Error type: Common errors are divided into different types, so that you can filter flows with errors by error type.
- Error description: Describes the detailed cause of the error to help you to locate the root cause and fix the error as soon as possible.

- Flow: Indicates the name of the flow that reported the error.
- Environment/Region: Indicates the specific environment/region of the app triggering the error and helps you quickly locate the environment where the error occurred.
- Trigger time: Indicates the specific time when the flow triggered the error.



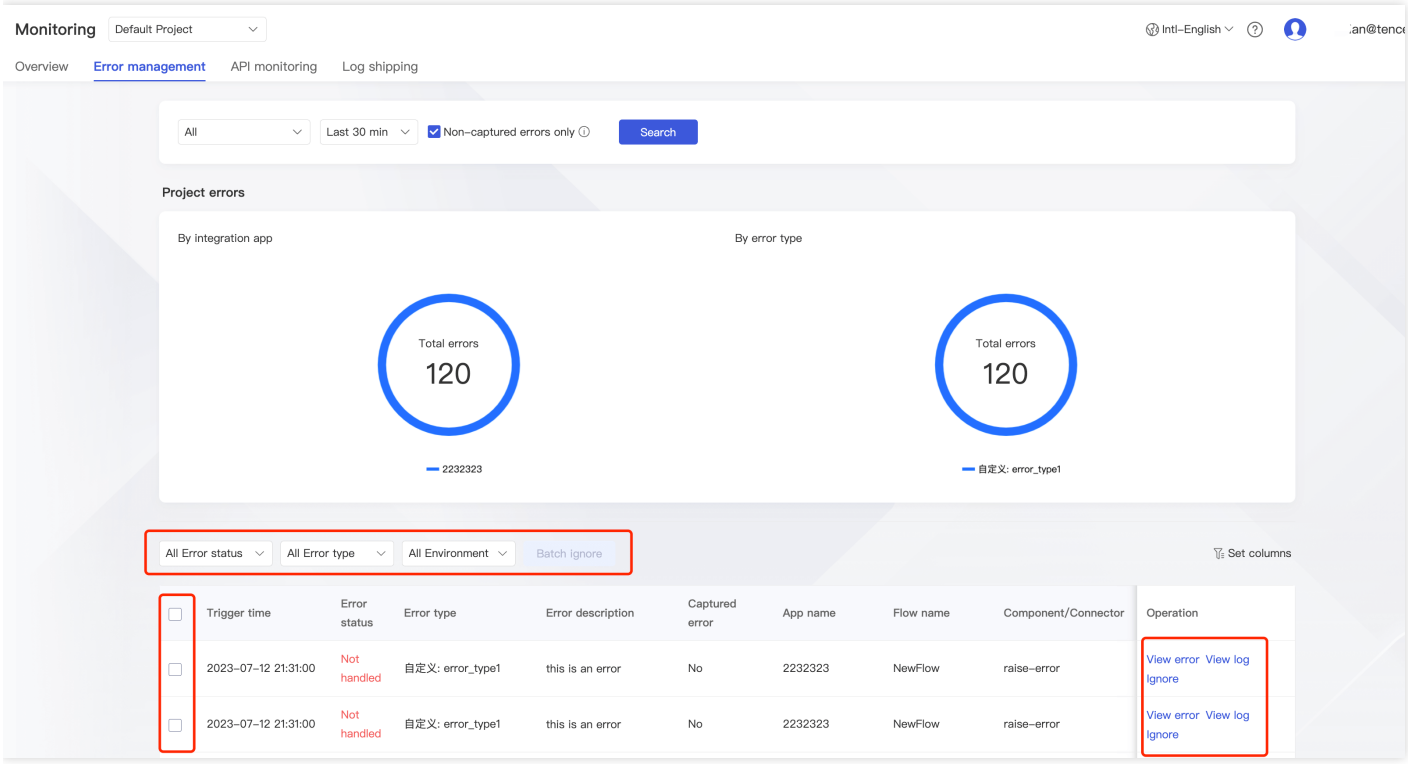
2. Viewing an error

Click **View error** to quickly locate the flow with the error.



3. Ignoring/Batch ignoring errors

If an error of a flow is fixed, you can click **Ignore** to ignore the same errors in the adjacent time periods. You can also select multiple errors and click **Batch ignore**.



Monitoring Metric Description

Category	Metric	Description
App	App monitoring overview	It aggregates the data by integration app (app data is displayed by default if **All flows** is selected) and allows you to view the data and charts of the selected app meeting the filter conditions.
	App triggers	It indicates the total number of times all flows under an integration app are triggered through triggers such as Scheduler, HTTP Listener, and Kafka Consumer. Once an app is triggered, all referenced flows will be executed. Therefore, the number of times an app is triggered is not greater than the number of flow executions.
	App executions	It indicates the total number of times all flows under an integration app are executed and consists of the total number of executions, number of successful executions, and number of failed executions. Taking the total number in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the total number of app executions in the last 10 minutes.

Category	Metric	Description
	App execution duration in milliseconds	It indicates how long it takes to execute an integration app. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all durations of the selected app in the last 10 minutes sorted in ascending order, that is, 99.9% of the durations are not greater than the value, which is also the case for other Nth percentile values.
	App traffic usage in KB	It indicates the amount of traffic generated by executing an app. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all traffic usage values of the selected app in the last 10 minutes sorted in ascending order, that is, 99.9% of the traffic usage values are not greater than the value, which is also the case for other Nth percentile values.
Flow	Flow monitoring overview	It aggregates the data by flow and allows you to view the data and charts of the selected flow based on the filter conditions.
	Flow executions	It indicates the number of times a flow is executed and consists of the total number of executions, number of successful executions, and number of failed executions. Taking the total number in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the total number of flow executions in the last 10 minutes.
	Flow execution duration in milliseconds	It indicates how long it takes to execute a flow. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all durations of the selected flow in the last 10 minutes sorted in ascending order, that is, 99.9% of the durations are not greater than the value, which is also the case for other Nth percentile values.
	Flow traffic usage in KB	It indicates the amount of traffic generated by executing a flow. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all traffic usage values of the selected flow in the last 10 minutes sorted in ascending order, that is, 99.9% of the traffic usage values are not greater than the value, which is also the case for other Nth percentile values.
Trigger	Connector monitoring overview	It aggregates the data for an executed connector and allows you to view the data and charts of the selected connector of the selected flow of the selected app based on the filter conditions.

Category	Metric	Description
	Connector executions	It indicates the number of times a connector of a flow is executed and consists of the total number of executions, number of successful executions, and number of failed executions. Taking the total number in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the total number of connector executions in the last 10 minutes.
	Connector execution duration in milliseconds	It indicates how long it takes to execute a connector of a flow. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all durations of the selected connector in the last 10 minutes sorted in ascending order, that is, 99.9% of the durations are not greater than the value, which is also the case for other Nth percentile values.
	Connector traffic usage in KB	It indicates the amount of traffic generated by executing a connector. Taking the 99.9th percentile in the last 24 hours at a 10-minute granularity as an example, each value on the line indicates the 99.9th percentile of all traffic usage values of the selected connector in the last 10 minutes sorted in ascending order, that is, 99.9% of the traffic usage values are not greater than the value, which is also the case for other Nth percentile values.

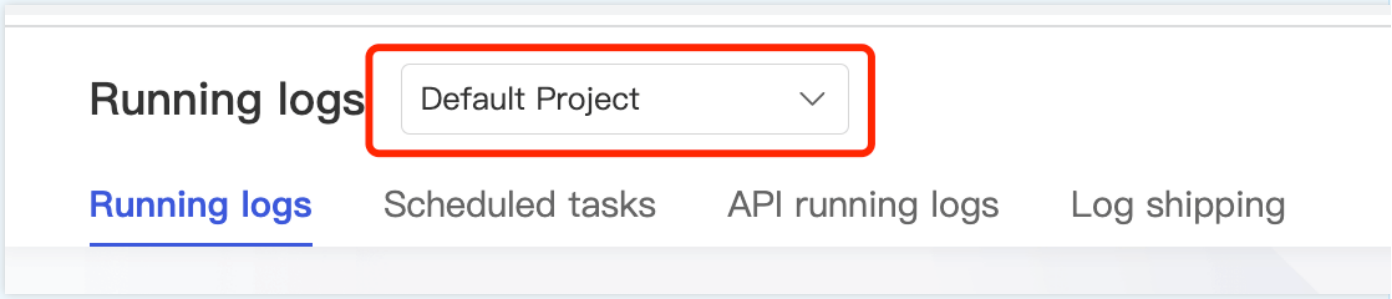
Execution Log

Last updated：2023-08-03 17:24:01

Overview

The running logs page displays the details of running logs of an integration app and allows you to manage scheduled tasks (flows whose trigger is the Scheduler component). You can view the real-time and historical running logs of a flow in a certain period of time.

Note：
Currently, iPaaS running logs are divided by project. To view logs, select a project first.



Directions

Running logs

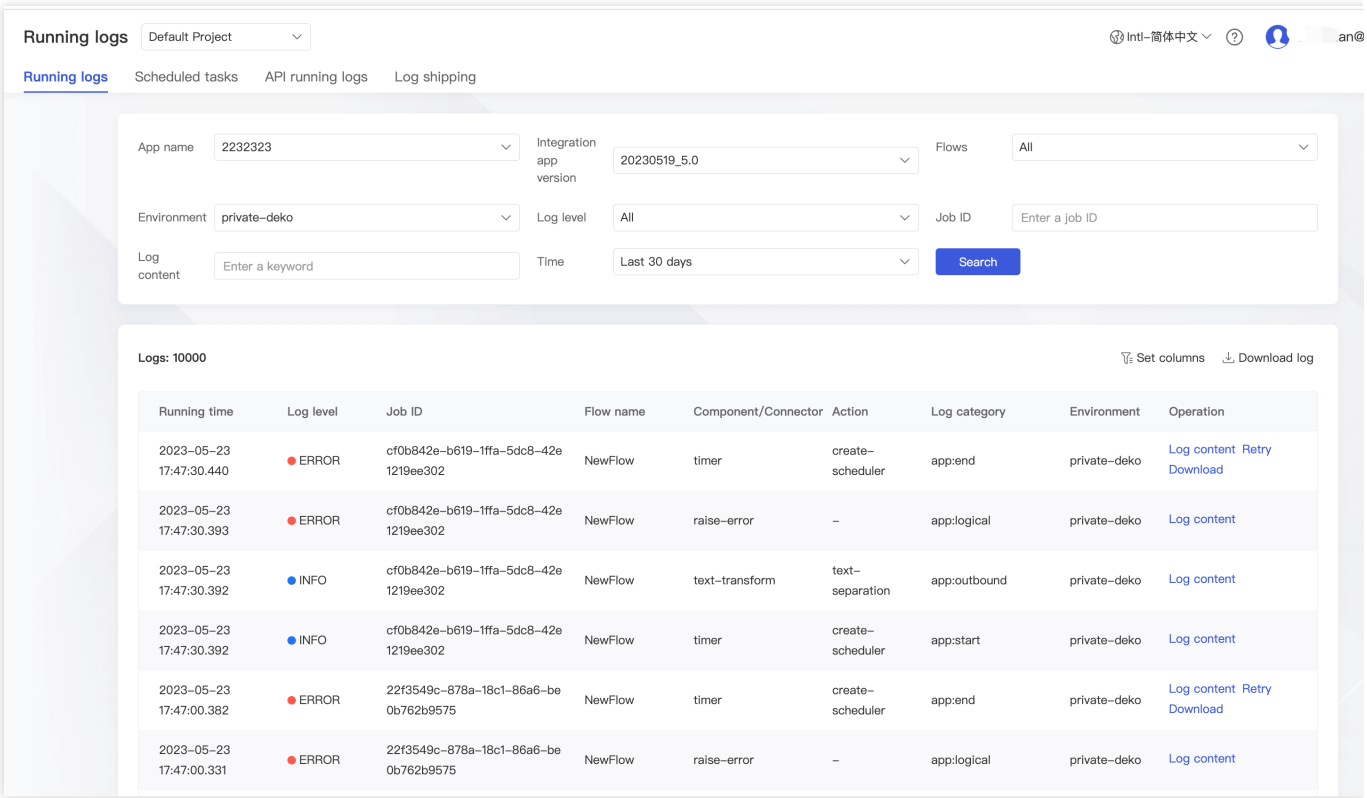
Step 1. Query running logs

You can view historical and real-time running logs and review the running information and message data of a job based on logs.

Note：
The console currently lets you search for running logs only from within the last 30 days. To search for earlier logs, please submit a ticket for assistance.

1. Log in to the iPaaS console and select **Running logs** > **Running logs**.

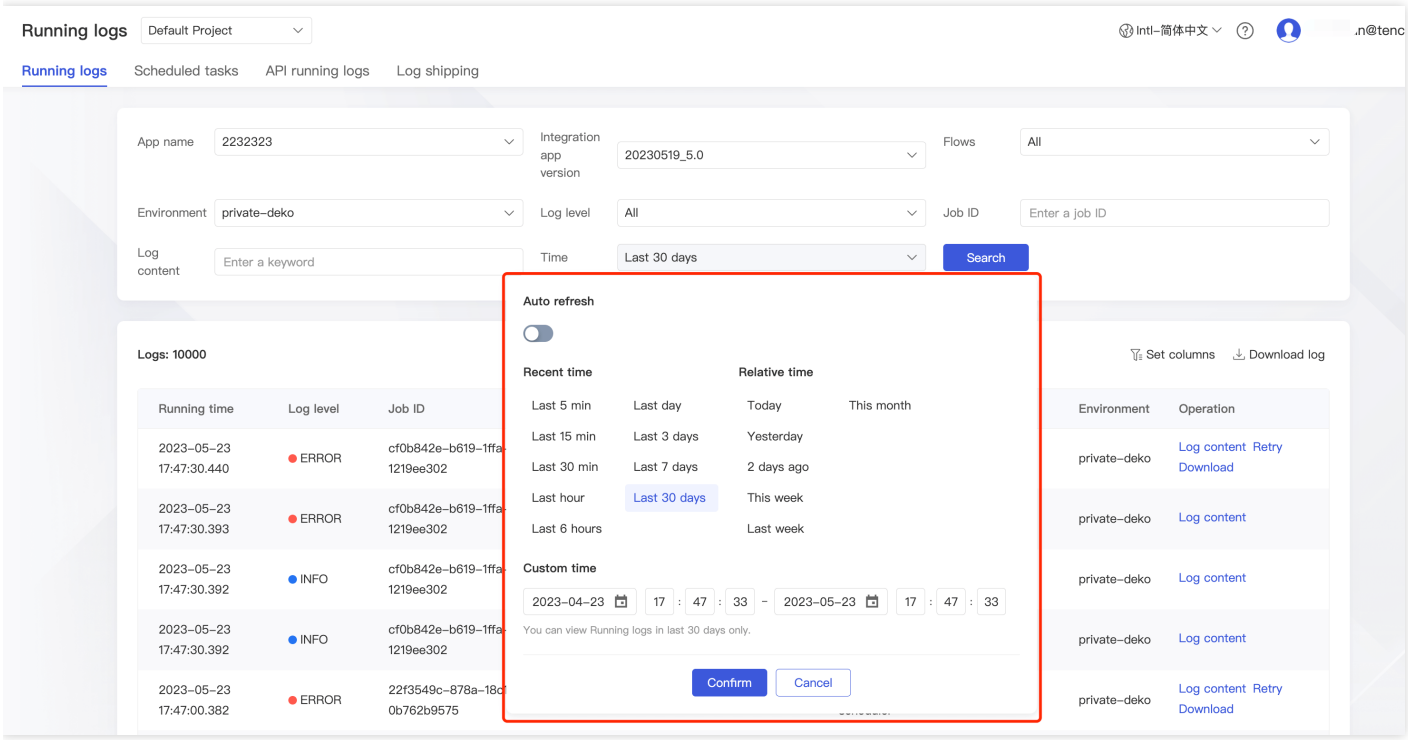
2. On the **Running logs** tab, the app running logs in the last 30 minutes are displayed in reverse chronological order by default. The log list contains the following information:
   - Running time: The time when the log is generated.
   - Log level: Logs at different levels are marked in different colors.
   - Job ID: The ID of the execution when the log is generated. You can find all logs generated by an execution by using the job ID.
   - Flow name: The flow generating the log.
   - Component/Connector: The component or connector generating the log.
   - Action: The action performed when the log is generated.
   - Log category: If the log is output by the Logger component, the custom log category will be recorded. For other components or connectors, the default output event will be recorded.
   - Environment: The environment where the app runs.
   - Log content: The summary of the log content.



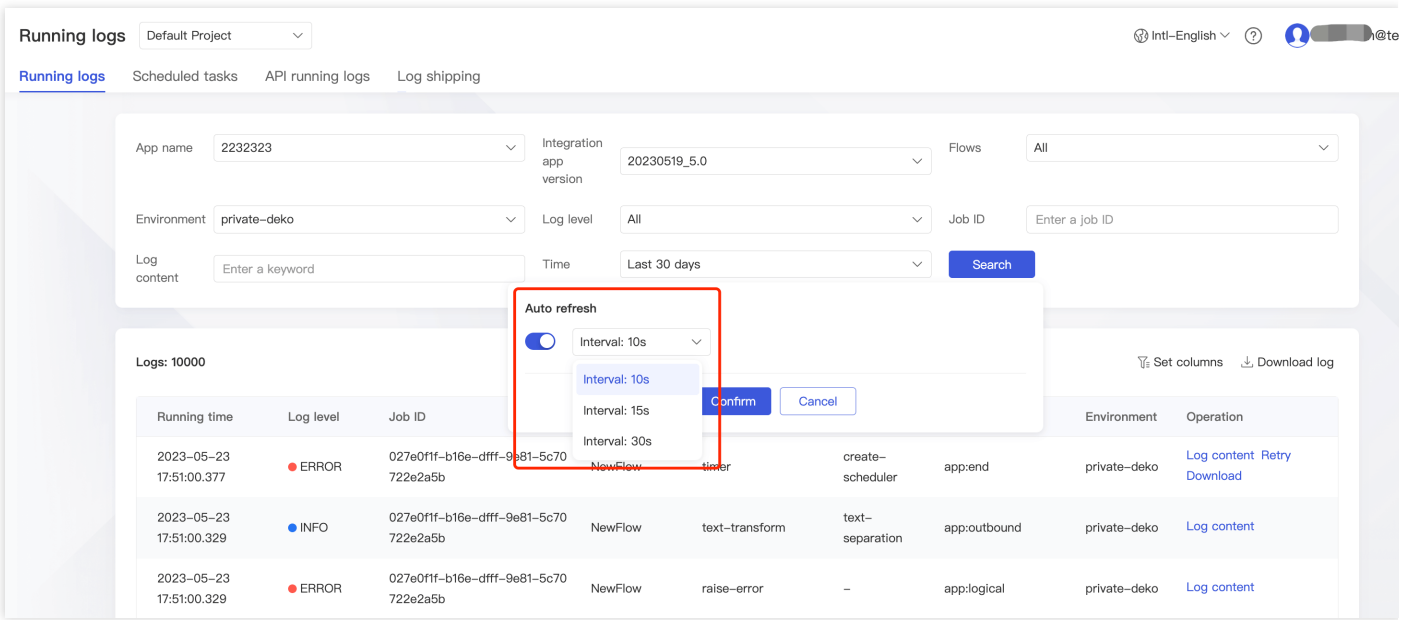
Description of log search: You can search by app name, version, flow name, runtime environment, log level, job ID, log content, and time.

Click the time drop-down list, and the panel for setting the time period for search will be displayed. You can select the last 5 minutes, last 15 minutes, last hour, last 6 hours, last day, last 3 days, last 7 days, last 30 days, today, yesterday, 2 days ago, this week, last week, or a custom time period. You can preset a frequently used time period to quickly filter

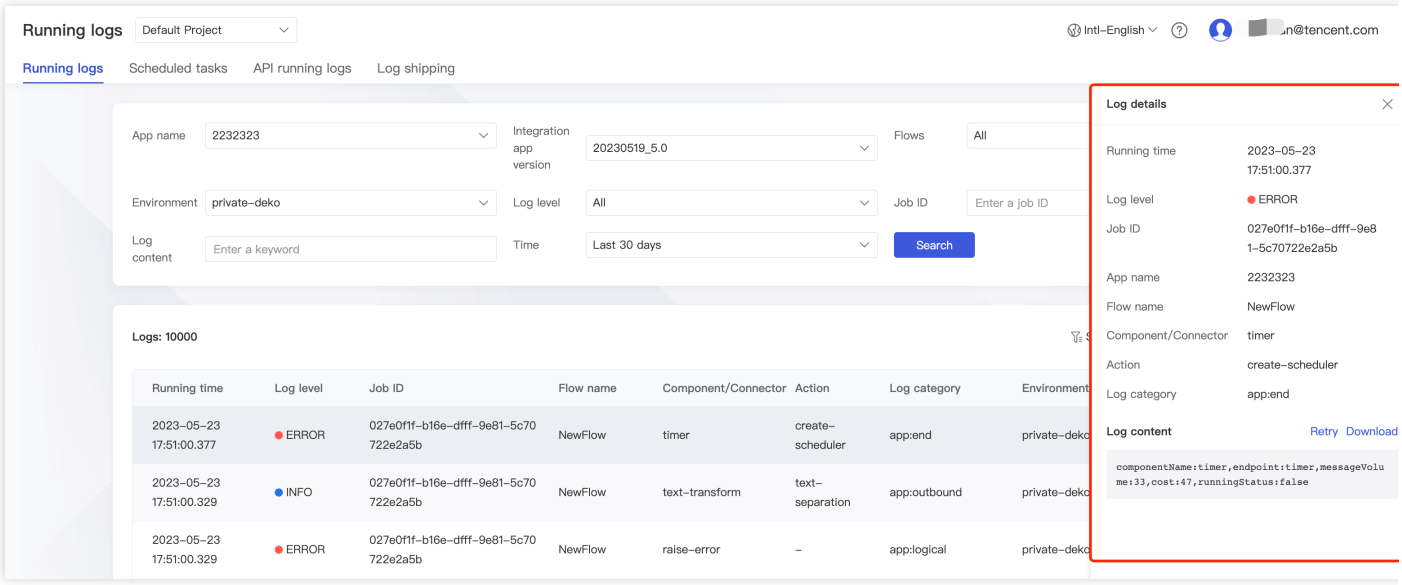
the logs within it.



- Automatic log refresh: Toggle on **Auto refresh**, and the refresh interval options will be displayed. After you select an interval, the logs will be refreshed at the set interval.

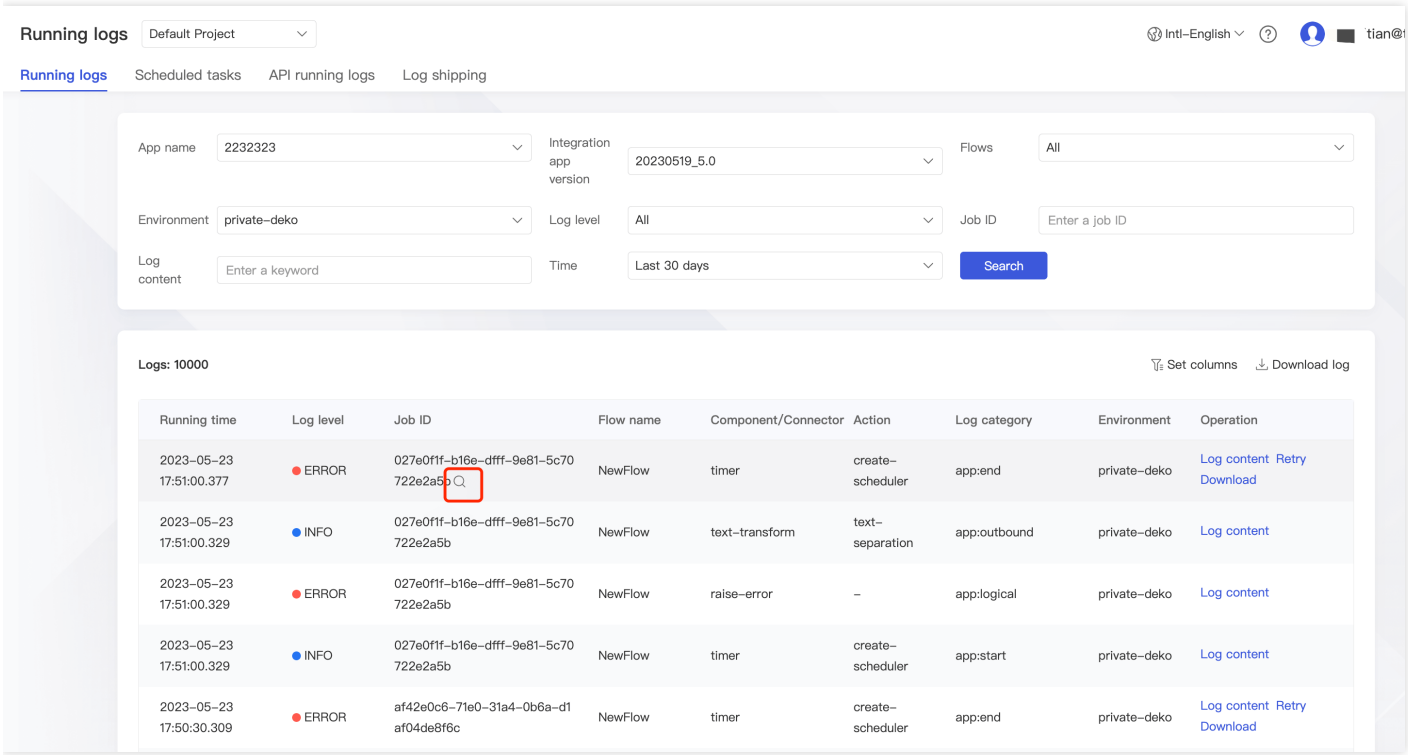


- Log details: Click the **Log content** of a log, and the log details will be displayed on the right.

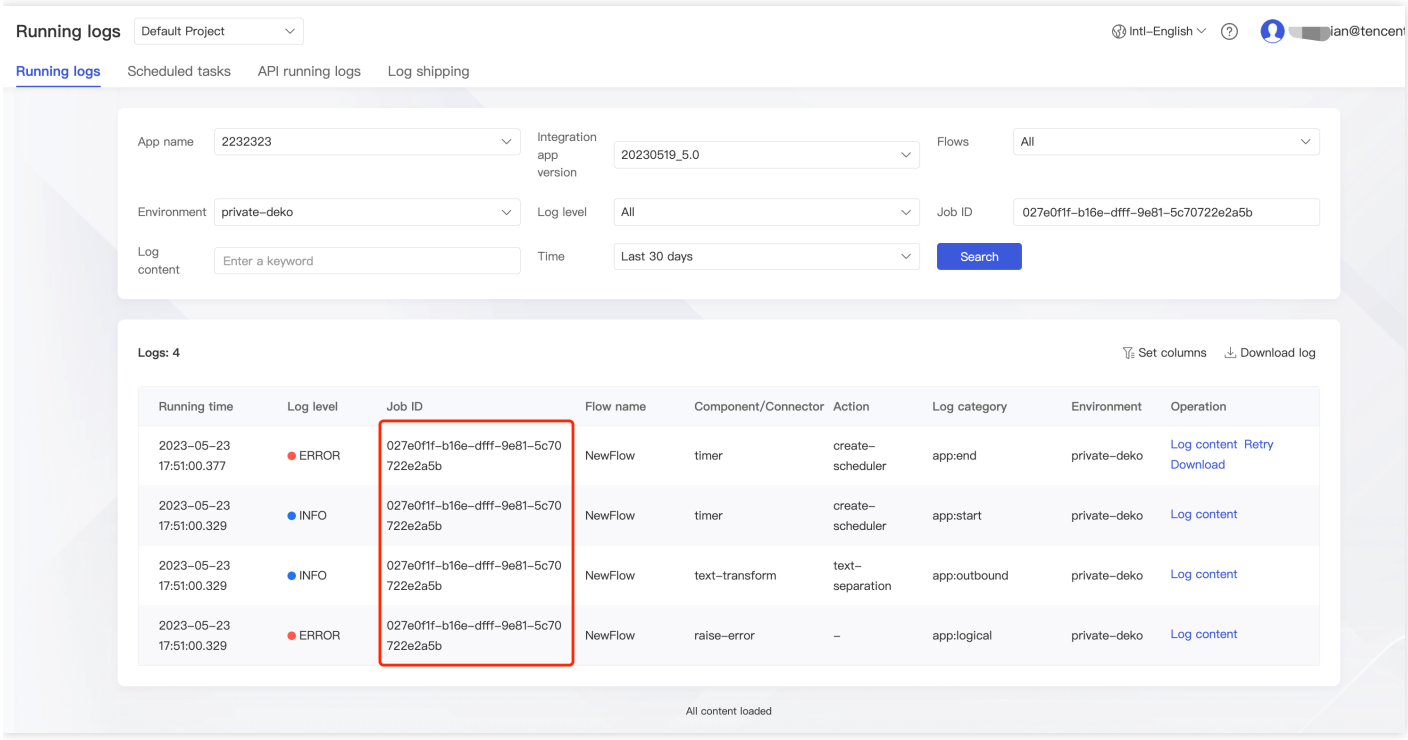


Step 2. Quickly search for logs by job ID

A job ID is generated each time a flow is executed. It is used to mark the logs generated during the execution. You can use the job ID to quickly find all logs generated during a flow execution.



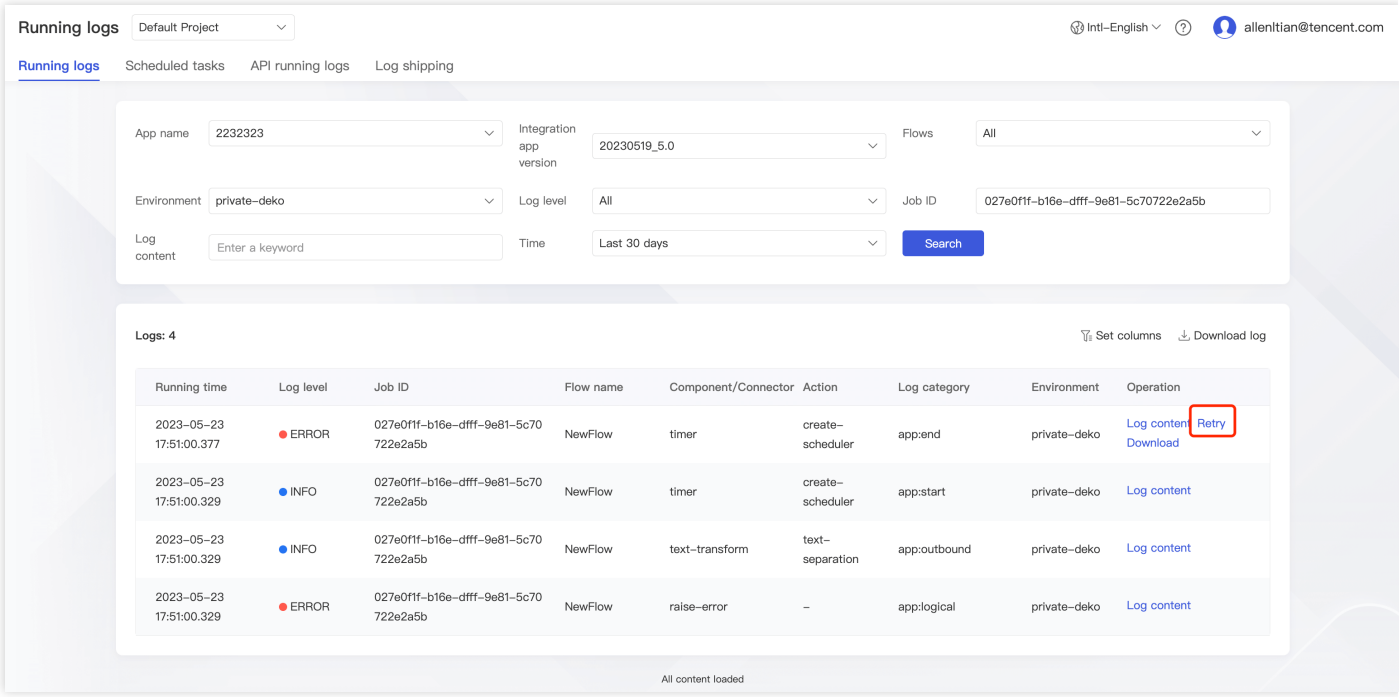
Quickly find all logs for an execution.



Step 3. Retry an integration task

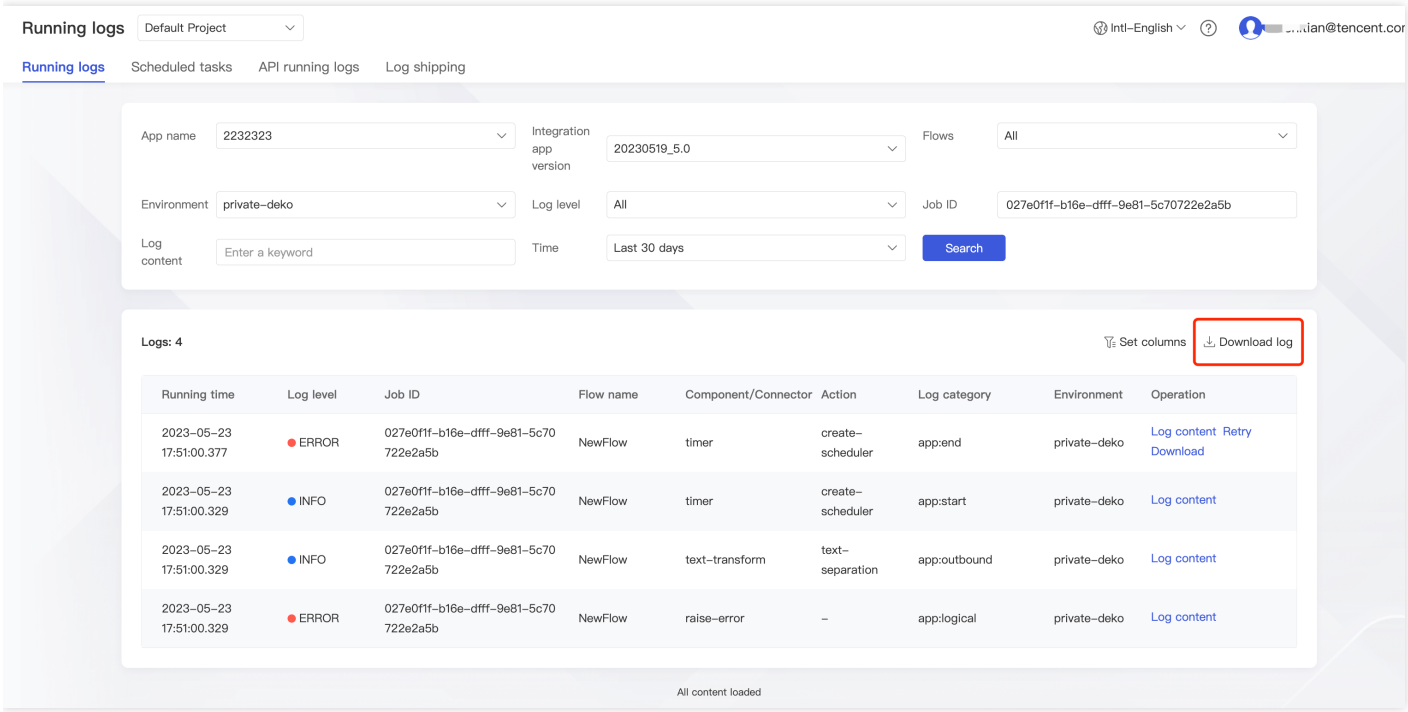
To guarantee the normal execution of integration tasks after an integration app is published, iPaaS improves the flow monitoring metrics and adds reliability analysis capabilities. If an integration task failed to be triggered, it can be retried automatically or manually to avoid data loss.

- Automatic retry: The backend automatically retries the task. If the retry succeeds, the log status will be updated to **Successful** in the console, and the previous failure won't be logged.
- Manual retry: If the eventual result of automatic retry is failure, the log status will be updated to **Failed** in the console, and you can retry the task manually.



Step 4. Download logs

In the **Running logs** tab, you can download running logs by clicking **Download log**.

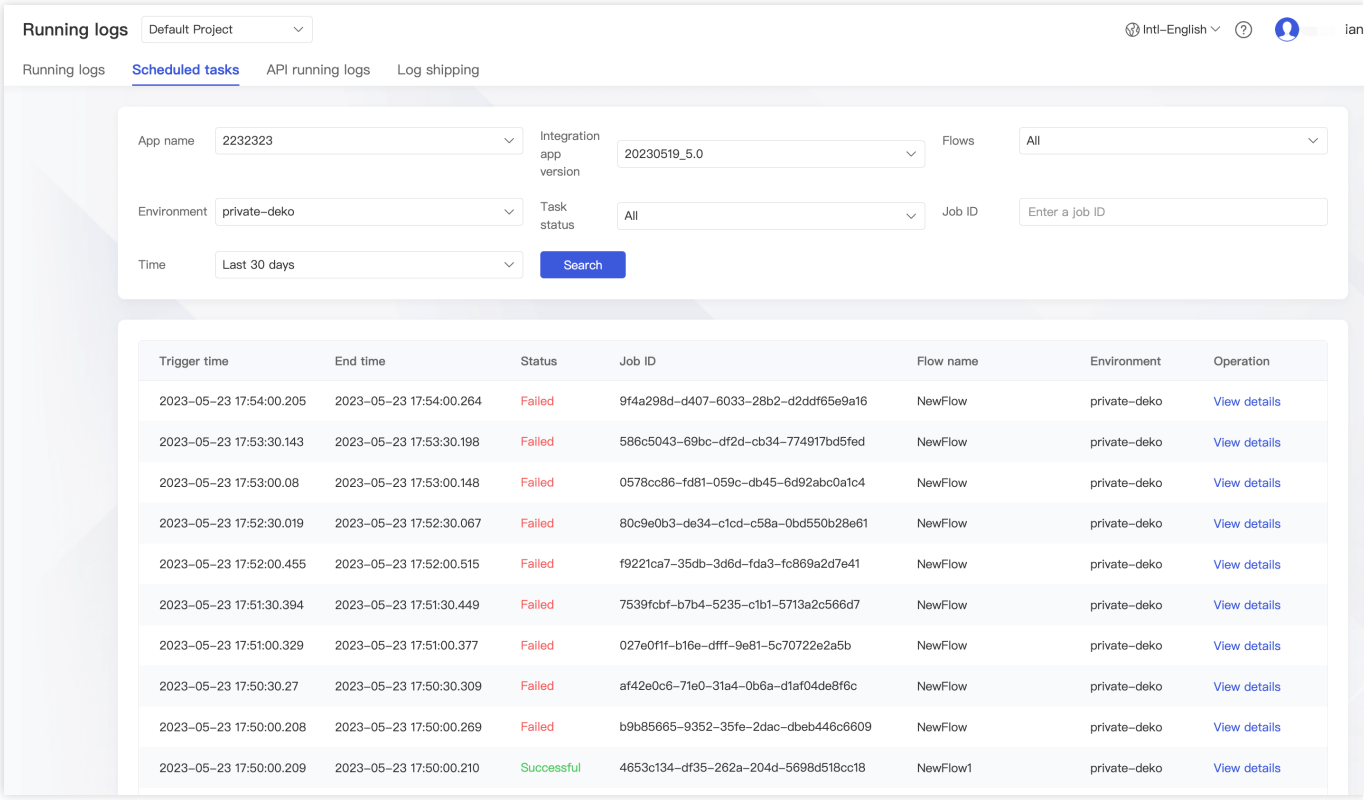


Scheduled task management

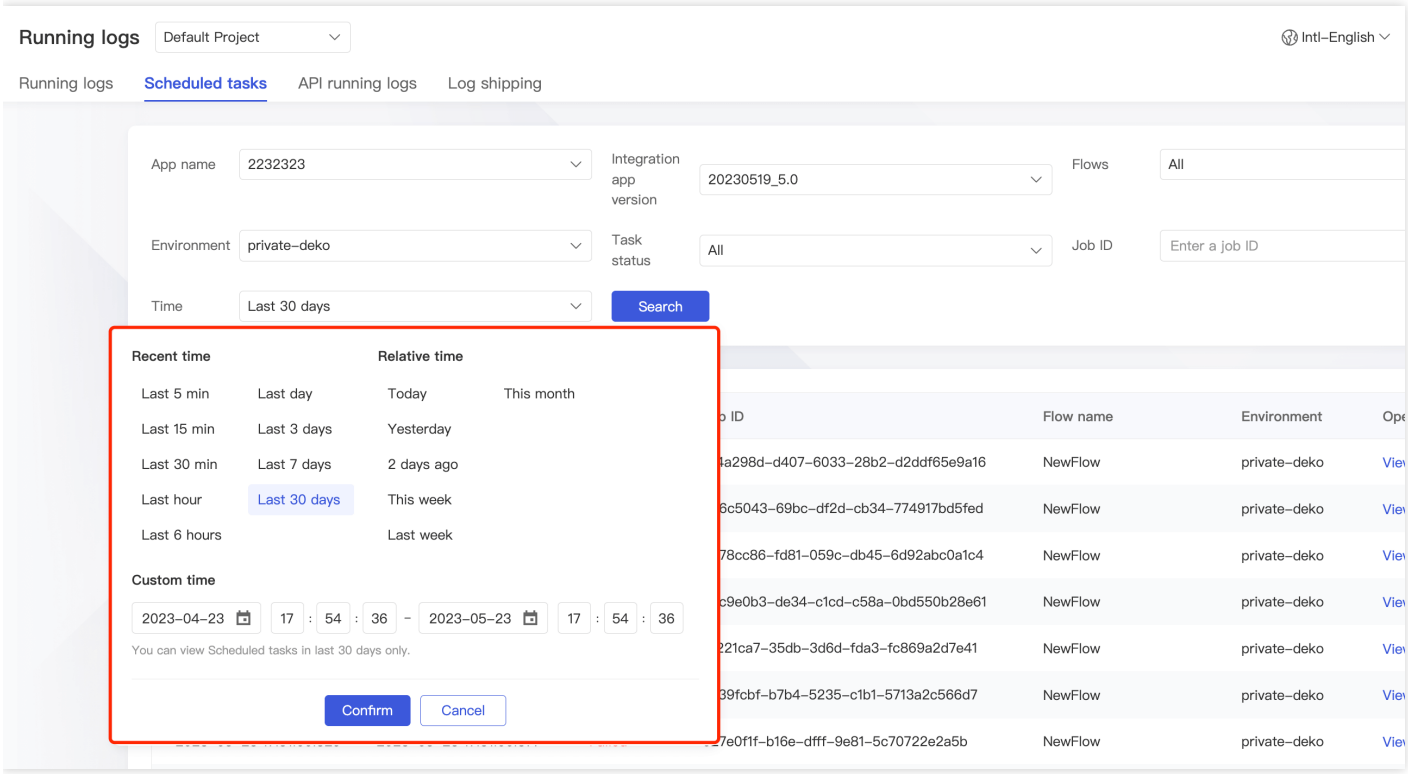
For apps that use the Scheduler component to trigger their flows at the scheduled time according to the configured rule, you can view the flow execution details in the last 30 days, including the trigger time, end time, status, and actions of the integration app.

> Note：
> Currently, you can search for the scheduled tasks in the last 30 days in the console. To search for earlier scheduled tasks, please submit a ticket for assistance.

1. Log in to the iPaaS console and select **Running logs** > **Scheduled tasks**.
2. On the **Scheduled tasks** tab, the app running logs of scheduled tasks in the last 30 minutes are displayed in reverse chronological order by default. The log list contains the following information:
   - Trigger time: The time when the app is triggered.
   - End time: The time when the trigger ends.
   - Status: The status of execution, which can be successful or failed.
   - Job ID: The ID of the execution when the log is generated. You can find all logs of an execution through the job ID for analysis.
   - Flow name: The flow generating the log.
   - Environment: The environment where the app runs.
   - Operation: View log details.



Description of log search: You can search by app name, version, flow name, runtime environment, job ID, and time. Click the time drop-down list, and the panel for setting the time period for search will be displayed. You can select the last 5 minutes, last 15 minutes, last hour, last 6 hours, last day, last 3 days, last 7 days, last 30 days, today, yesterday, 2 days ago, this week, last week, or a custom time period. You can preset a frequently used time period to quickly filter the logs within it.



Alarm Configuration
Alarm Policy

Last updated：2023-08-04 09:55:16

Overview

This document describes how to create, modify, enable, disable, copy, and delete an alarm policy in the alarm module of iPaaS.

You can use an alarm policy to set a threshold alarm for metrics such as number of execution failures for a launched integration app. In this way, you will promptly receive notifications and can then take measures accordingly when an exception occurs. An alarm policy consists of alarm name, alarm object, alarm condition, and alarm notification template.

Basic Concepts

Term	Definition
Alarm policy	It consists of alarm name, description (optional), alarm condition, alarm object, and alarm notification template.
Alarm name	It is the name of an alarm policy.
Description	The alarm description is a simple definition of the purpose of the alarm policy.
Alarm object	It can be a launched integration app.
Alarm condition	An alarm condition is a semantic condition consisting of metric, judgment logic, judgment condition, statistical period, and number of consecutive monitoring periods.
Notification template	A notification template can be quickly reused for multiple alarm policies, making it suitable for alarm receipt in various use cases. For more information, see Creating notification template.

Use Limits

Feature	Limit

Feature	Limit
Alarm object	Up to three items can be added.
Alarm condition	Up to four items can be added.

Directions

Creating an alarm policy

1. Log in to the iPaaS console.
2. On the left sidebar, select **Ops center** > **Alarm settings** > **Alarm policies** to enter the alarm policy settings page.
3. Click **Create** and configure a new alarm policy as detailed below:

Configuration Type	Configuration Item	Description
Configure basic information	Policy name	Custom policy name.
	Description	Custom policy description.
Configure alarm rule	Alarm object	◦ Select an app name. The alarm policy will be bound to the selected app and all of the app's flows. ◦ Select one or more target regions. Only the integration app in the selected regions will be triggered when the trigger condition is met.
	Alarm condition	◦ **Alarm trigger condition:** It is a semantic condition consisting of metric, judgment logic, judgment condition, statistical period, and number of consecutive periods. You can set an alarm threshold based on the trend shown in the chart. For example, if the metric is set to `Number/Total number of app triggers`, judgment logic to `>`, judgment condition to `10`, statistical period to `1` minute, and the number of consecutive periods to `Last 3 periods`, the number/total number of app triggers is collected once every minute, and if the number/total number of triggers of an app is greater than 10 for three consecutive times, an alarm will be triggered. ◦ **Alarm frequency:** You can set a repeated notification policy for each alarm rule. In this way, an alarm notification will be sent at a specified frequency when an alarm is triggered. Frequency options: Once every 5 minutes, once every 15 minutes, once every 30 minutes, once every 12 hours, and other frequency options.

Configure alarm notification	Notification template	You can create a custom alarm notification template. Each alarm policy can be associated with one template. For more information, see Notification Template.

4. After configuring the above information, click **Confirm**.

Note：
For the metric description, see Monitoring Metric Description.

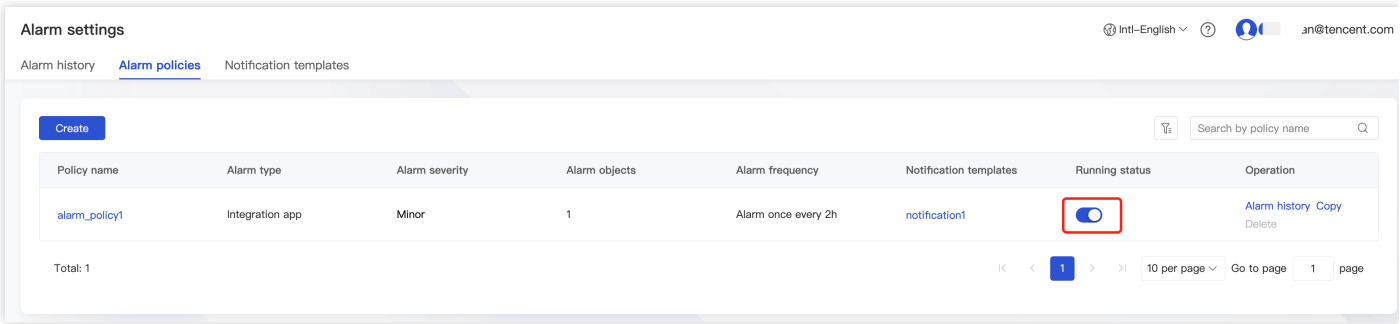
Modifying an alarm policy

1. Log in to the iPaaS console and go to the **Alarm policies** page.
2. Click the name of the target alarm policy.
3. On the **Edit alarm policy** page, directly modify the relevant information and click **Confirm**.

Enabling/Disabling an alarm policy

You can use the alarm toggle feature to enable or disable an alarm policy as needed. This allows you to disable unwanted alarm policies to get rid of redundant messages. You can also quickly enable the disabled alarm policy again when needed.

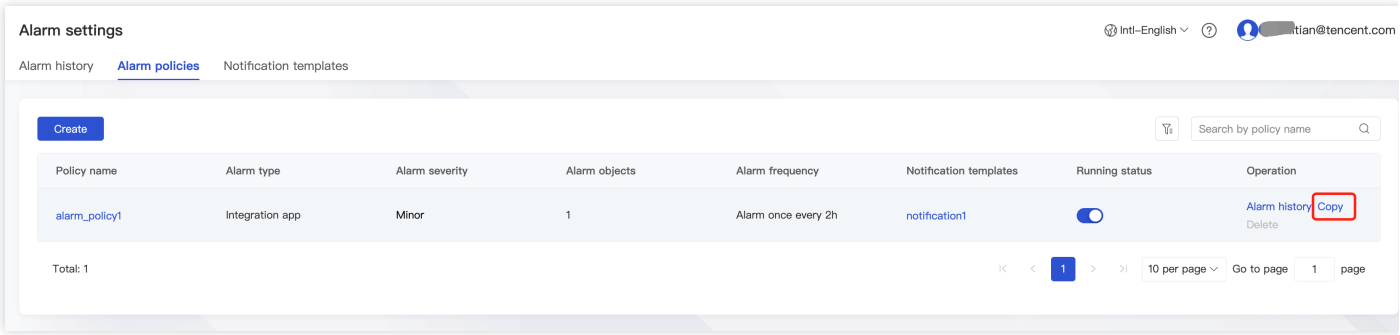
1. Log in to the iPaaS console.
2. Select **Alarm settings** on the left sidebar and click **Alarm policies** to enter the management page.
3. Click the toggle in the **Status** column of the target policy to enable or disable alarms for the policy.



Copying an alarm policy

1. Log in to the iPaaS console and go to the **Alarm policies** page.
2. Click **Copy** in the **Operation** column of the target alarm policy.

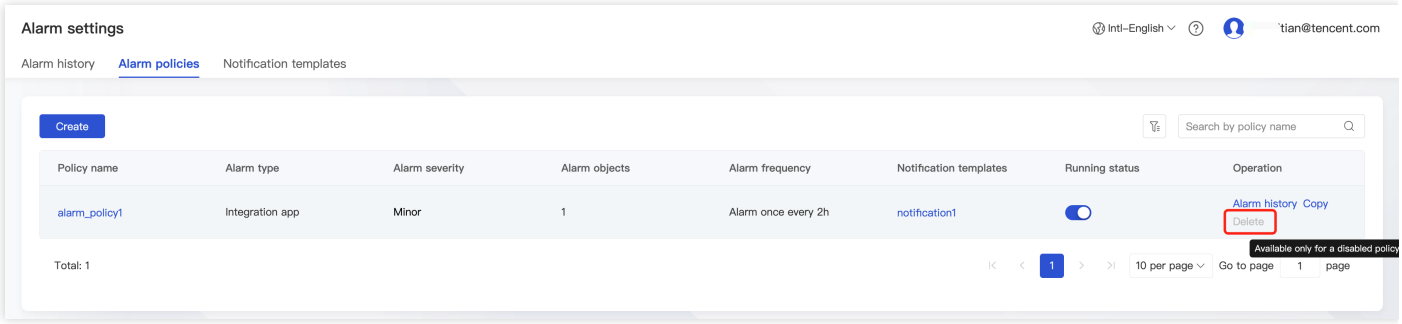
3. Modify the information of the copied alarm policy on the redirected page and click **Complete**.



Deleting an alarm policy

1. Log in to the iPaaS console and go to the **Alarm policies** page.
2. Click **Delete** in the **Operation** column of the target alarm policy and confirm the deletion in the pop-up window.

> Note：
> Only disabled policies can be deleted.



Notification Template

Notification Template

Last updated：2024-12-30 14:12:47

Overview

This document describes how to configure a notification template in the alarm module in iPaaS.
You can quickly reuse a template for multiple policies to reduce repeated user notification configurations. You can customize the user notification methods; for example, you can configure alarm notification channels as Message Center, email, and SMS. You can also use different notification time periods for different users; for example, you can configure user A to receive alarm notifications in the daytime and user B to receive notifications at night.
You can create, edit, and delete notification templates.

Prerequisites

Viewing a notification template: The sub-account must have the project read permission in iPaaS.
Creating/Editing/Deleting a notification template: The sub-account must have the project write permission in iPaaS.
Note:
For more information on how to grant sub-accounts permissions, see Granting Tencent Cloud Service Permissions.

Use Limits

Feature	Limit
User notification	Up to five items can be added.
Webhook	Up to five URLs accessible over the public network can be entered.

Directions

Creating a notification template

1. Log in to the iPaaS console and select **Alarm settings** > **Notification templates**.
2. Click **Create** and enter basic information in **Create notification template**.

Template name: Enter a custom template name.

3. Configure the notification operation. The parameters are as detailed below:

User notification

Note:

System admin and project admin: They can select all member accounts of the current project in the drop-down list.

Project member: They can select only their own accounts.

Ordinary member: They can select only their own accounts.

Parameter	Description
Recipient	Select one or more users as recipients.
Time range	Define the time period for receiving alarms.
Recurrence	You can select days of the week for receiving notifications. By default, notifications are sent every day.
Notification type	Three alarm channels are supported: Message Center, email, and SMS. You can also set different channels and notification time periods for different users. For more information, see Alarm Receiving Channel.

Webhook

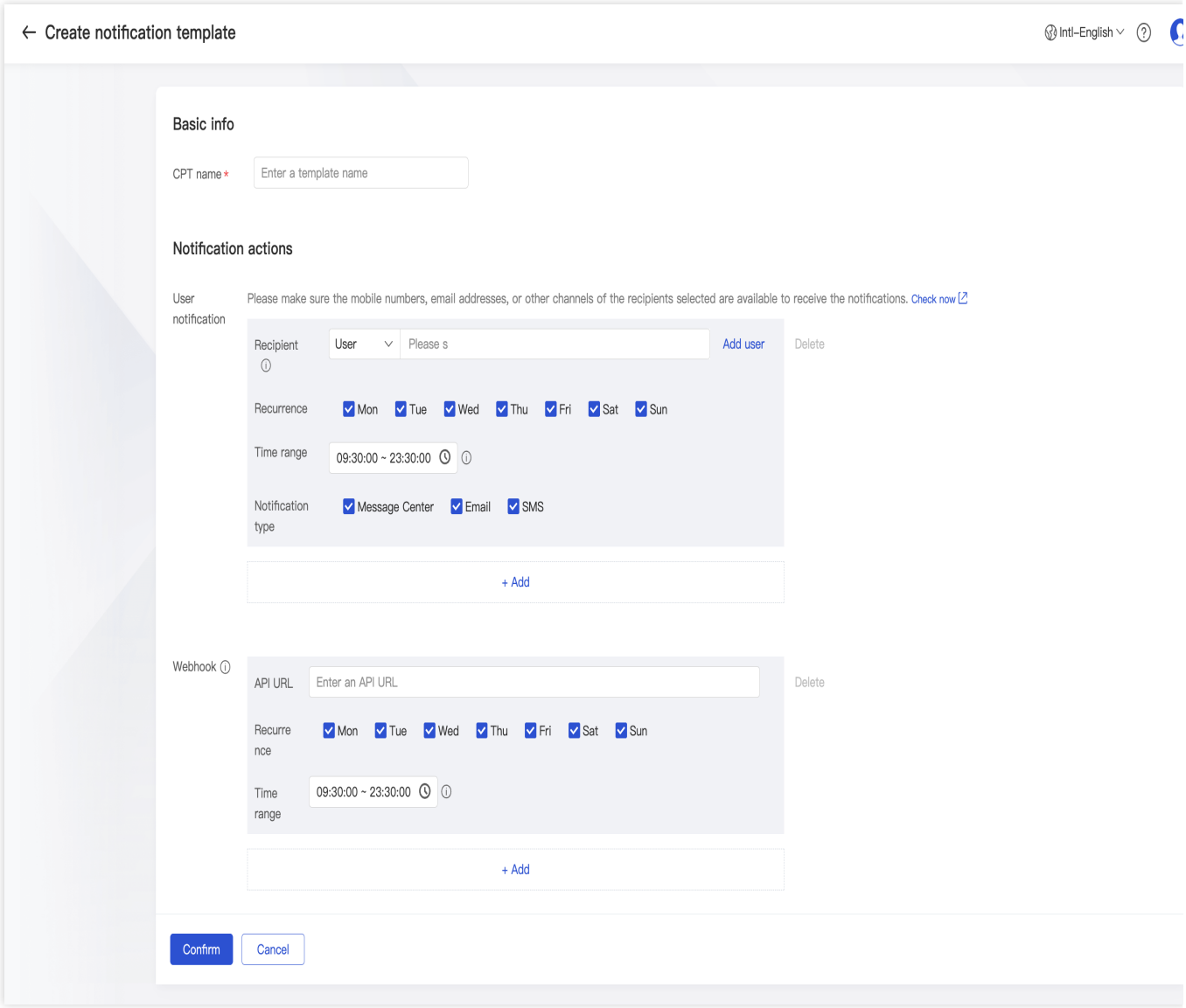
Parameter	Description
API URL	You can enter up to five URLs accessible over the public network as the callback API addresses, and iPaaS will push alarm messages to them promptly.
Time range	Define the time period for receiving alarms.

Note:

After the callback URL is saved successfully, when a created alarm policy is triggered or the alarm is cleared, the alarm messages will be pushed through webhook.

When a created alarm policy is triggered or the alarm is cleared, the alarm messages will be pushed through webhook. Webhooks also support repeated alarms.

4. Click **Confirm**.



Modifying a notification template

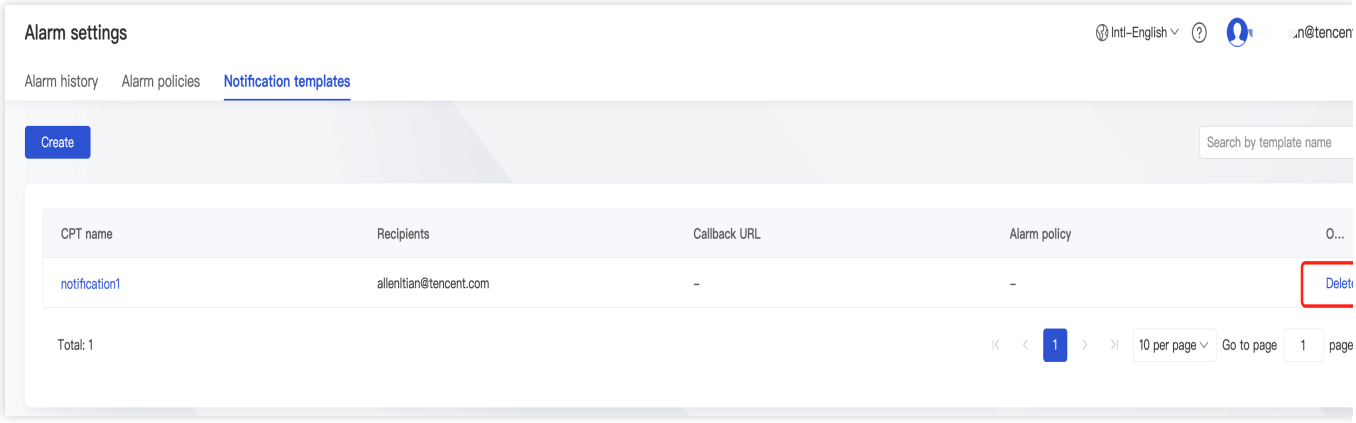
1. Log in to the iPaaS console and select **Alarm settings** > **Notification templates**.
2. Click the target template name to enter the editing page.
3. Edit the target content and click **Confirm**.

Deleting a notification template

1. Log in to the iPaaS console and select **Alarm settings** > **Notification templates**.
2. Find the name of the target template, click **Delete** in the **Operation** column on the right, and confirm the deletion in the pop-up window.

Note:

A template referenced by an alarm policy cannot be deleted. Cancel the reference in the alarm policy first before deleting the template.


Alarm settings
Intl-English
Alarm history
Alarm policies
Notification templates
Create
Search by template name
CPT name
Recipients
Callback URL
Alarm policy
O...
notification1
allenltian@tencent.com
-
-
Delet
Total: 1
1
10 per page
Go to page
1
page

API Callback

Last updated：2023-08-03 17:24:01

Overview

Your self-built system can directly receive iPaaS alarm notifications through API callbacks. API callbacks can push alarm notifications to URLs that are accessible over the public network through GET requests. You can take further actions based on the alarm notifications you receive from API callbacks.

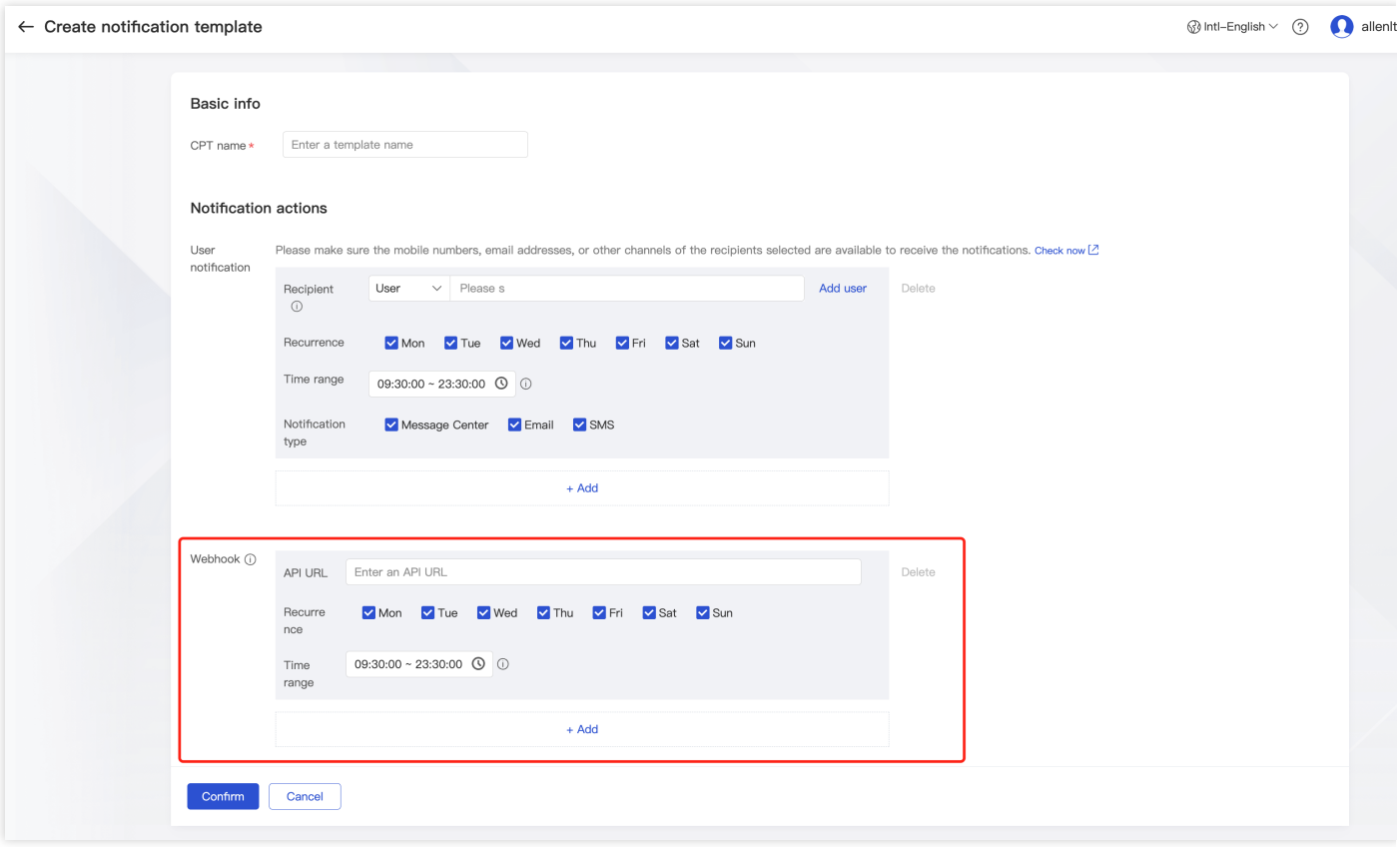
Notes

- Currently, alarm callback does not have an authentication mechanism and does not support HTTP authentication.
- When a created alarm policy is triggered or the alarm is resolved, the alarm messages will be pushed through the API callbacks. API callbacks also support repeated alarms.

Directions

1. Log in to the iPaaS console and select **Alarm settings** > **Notification templates**.
2. Click **Create** to enter the **Create notification template** page.
3. After configuring the basic information, enter a URL accessible over the public network as the callback API address (such as `domain name or IP[:port][/path]`) in the API callback module, and iPaaS will push alarm messages to this address promptly.
4. Enter the alarm policy list, click the name of the target policy to enter the **Edit alarm policy** page, and click the notification template.

5. Alarm messages will be pushed to URLs that are accessible over the public network through GET requests.



Alarm History

Last updated：2023-08-04 09:57:03

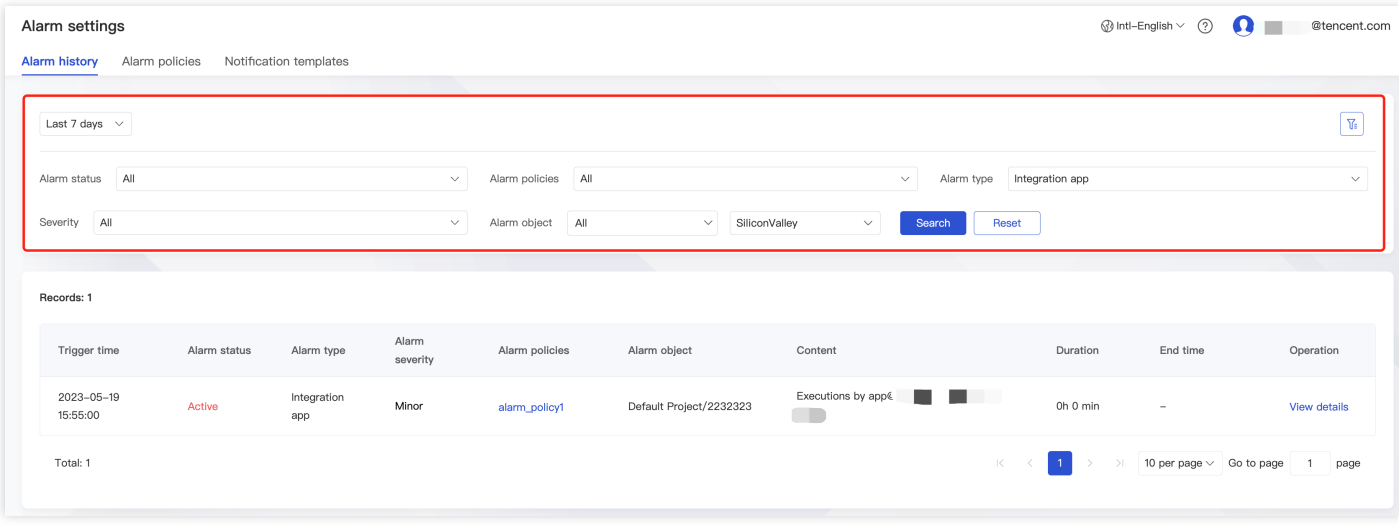
Overview

You can view alarm records in the last 30 days on the alarm history page of iPaaS.

Directions

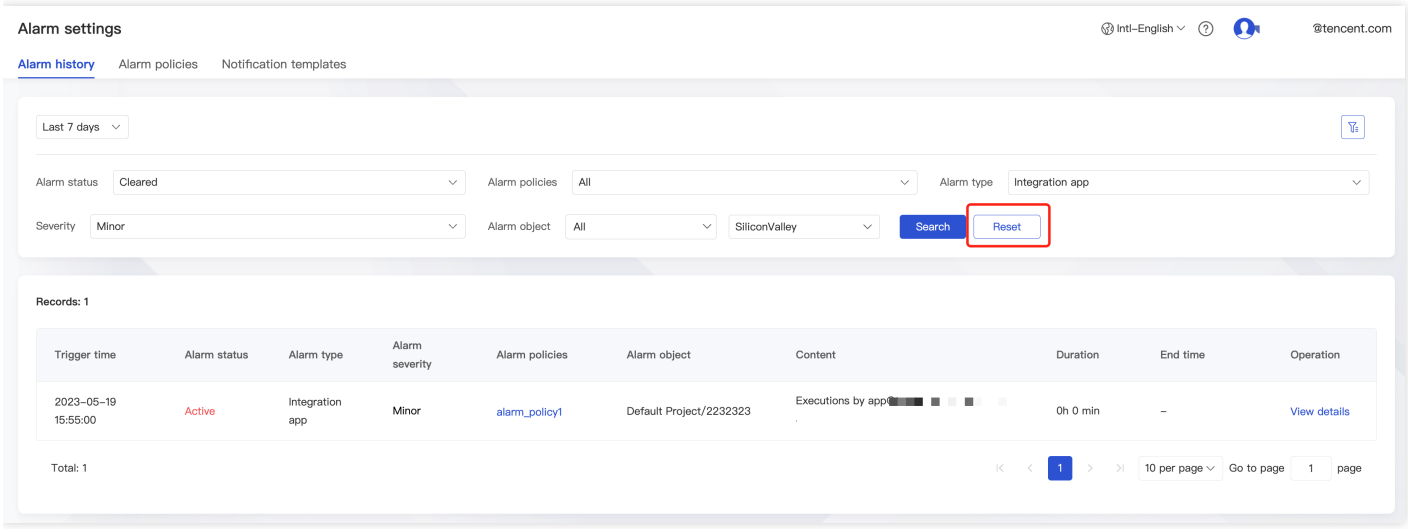
Viewing alarm history

1. Log in to the iPaaS console and select **Alarm settings** > **Alarm history**.
2. (Optional) Click the time filter in the top-left corner to select the time range of the alarm records to be viewed. You can quickly filter alarm records in the last 5 minutes, last 15 minutes, last 30 minutes, last hour, last 6 hours, last 24 hours, yesterday, last 7 days, or last 30 days. You can select a custom time range within the last 30 days.
3. (Optional) You can enter the information of an alarm object (such as app name) in the **Alarm object** search box to search for the records.
4. (Optional) You can also search for alarm records by alarm status, policy name, and environment.



Resetting filters

After successfully filtering alarm records, click **Reset** in the list.



Alarm Status

Alarm Status	Description
Active	The alarm has not been processed or is being processed.
Cleared	The alarm has been restored to the normal status.
Invalid	The alarm is invaild, as the policy has been deleted and so on.

Integration Tools
Connection

Last updated：2023-08-04 09:58:51

Overview

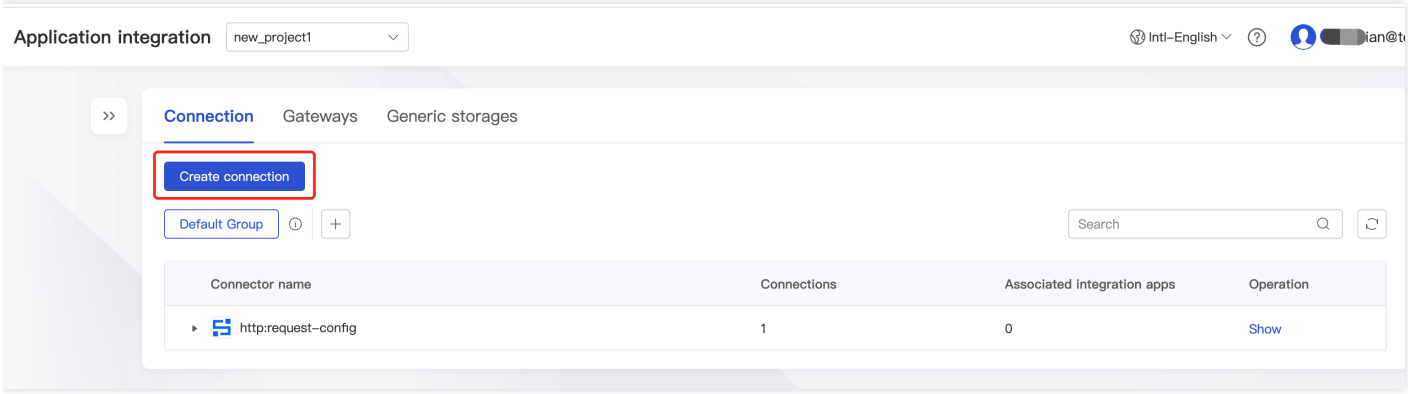
The **Connection** page centrally displays the configuration information of all authorized app accounts under the current account. In the same integration app, you can add multiple connections. In the same project, you can use an applicable configuration for different integration apps. You can also quickly add and modify connections. The configuration content of used integration apps will also be updated in real time.

Directions

Adding a connection

Step 1. Configure the basic information

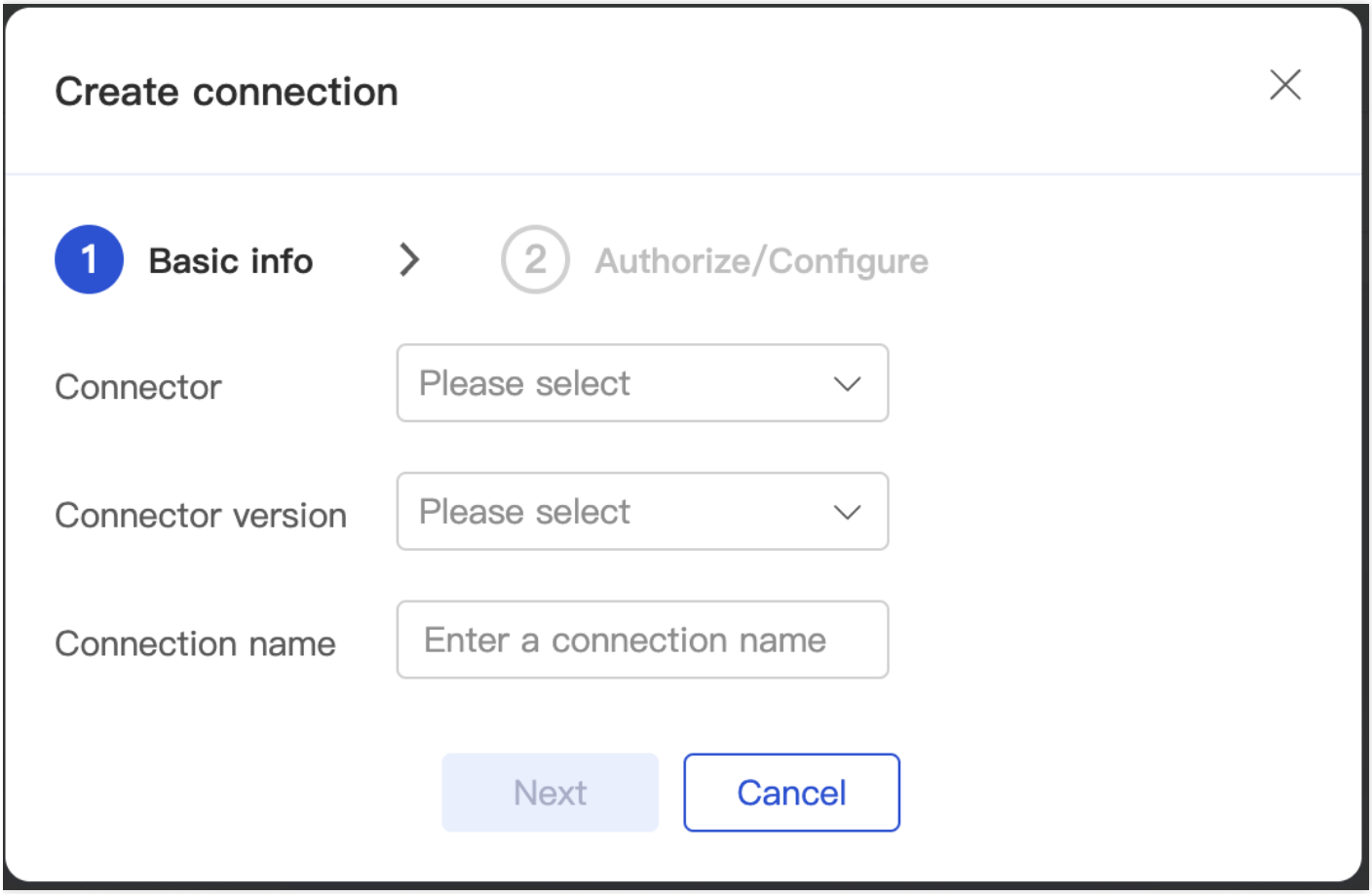
Log in to the iPaaS console and select **Integration tools** > **Connection**. On the **Connection** page, select the target project name and click **Create connection** to enter the connection creation page.



Step 2. Complete authorization and configuration

Enter the information as prompted and click **Next** to enter the authorization/configuration page. The information required for third-party app authorization varies by connector. Therefore, enter the information after getting the key or

account password document from the third party.



Step 3. Test the connection

Some connections can be tested. After entering the connection configuration, you can test the connection to verify whether the connection configuration is normal. If the test succeeds, the configuration is completed and can be used normally; if the test fails, you need to check whether the entered information is correct.

Modifying a connection

You can modify existing connections but need to note the following:

Note：

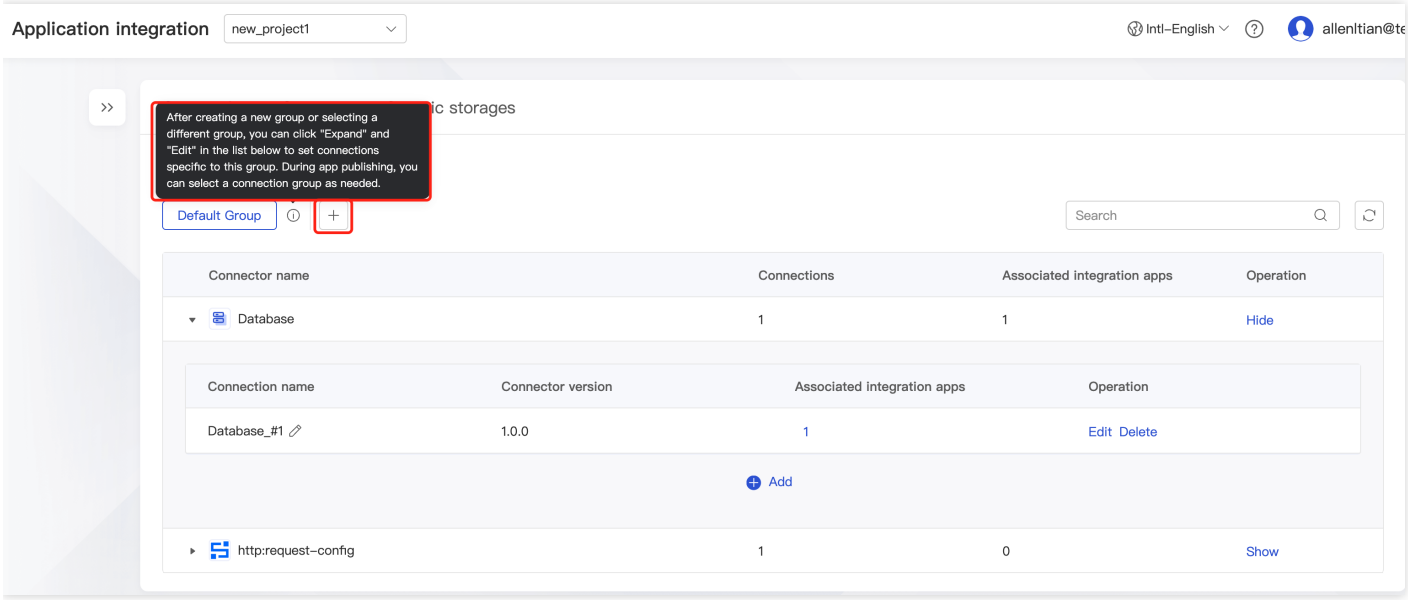
- If the connection is modified, the connector's associated integration apps will be affected.
- To make the modification of the connection take effect, you need to restart associated integration app tasks; otherwise, running tasks will throw an exception. Restarting tasks will stop running tasks instantly.

Connection group

Connection management uses groups to suit multi-environment and multi-account use cases. You can switch between groups to quickly switch to different configurations for connection. For example, in the production/test environment, you can use the configurations in group 1/2.
Group: The same connection can exist in different groups, and different groups can have different connections, so that you can switch between them quickly. The default group cannot be deleted.

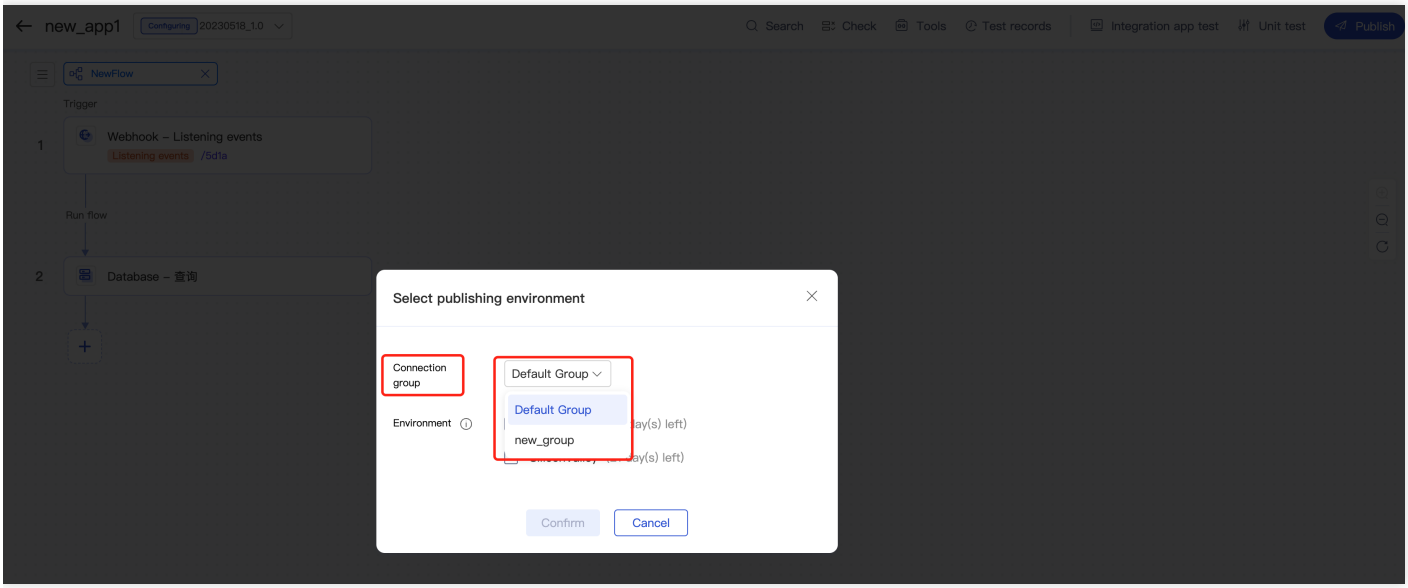
Step 1. Add a group

You can create multiple groups. In each group, you can configure different authorized accounts for the same connection for different business environments.



Step 2. Switch between groups

When publishing an app, you can select different connection groups to quickly switch to the connection required for the environment.



Security Gateway

Last updated：2023-08-04 10:01:01

Overview

A security gateway is a proxy system designed to integrate and interconnect iPaaS and your private network service. You can use it when you want to integrate your private network service through iPaaS (deployed in the public cloud) but your private network service is inaccessible over the public network.

A security gateway consists of the Server and the Agent:

- The Server is deployed in the private network of iPaaS, and you don't need to care about it.
- The Agent is deployed in your private network. You can deploy multiple Agents in different regions or for different services and use the security gateway for forwarding data to implement data interaction between iPaaS and your private network service.

Configuring the Agent

Step 1. Download the Agent

1. Log in to the iPaaS console and select **Security gateway**.
2. Click **Create** and upload a public key as instructed in Generating Public and Private Keys.
3. Configure the Agent IP allowlist and private network service, confirm the information, and click **Save**.
4. Click **Download Agent** in the security gateway list.

- The directory structure of the decompressed Agent package is as detailed below:
  - The `bin` directory contains the executable programs of the Agent, which are in sub-directories `Linux`, `Windows`, and `Mac` for use on different operating systems.
  - The `configs` directory contains the configurations required for Agent execution. In `configs`:
    - The `client` directory stores the configurations such as key required for Agent TLS communication. Such configurations correspond to those of the Server. Files in this directory **cannot be deleted or modified**.
    - The `secret` directory stores the private key for the Agent to connect to the Server. For more information on how to generate a private key, see Generating Public and Private Keys.
    - The `config.yaml` file contains configurations that must be depended on for Agent execution.
    - The `logger_config.yaml` file contains the log configuration for Agent execution. **You can modify the log level and log backup policy**.
  - The `log` directory stores logs generated during Agent execution.

- The `scripts` directory stores the Agent startup/stop scripts (`start.sh/stop.sh`).

Step 2. Configure Agent logs

You can modify the `logger_config.yaml` file in the `ipaas-private-cloud-agent/configs` directory of the Agent to modify the gateway log level and log backup policy as needed. The meaning of each parameter has been detailed in the file.

Step 3. Start the Agent

Run the startup script for your operating system to start the Agent:

- macOS
- Linux
- Windows

Run the following command to start the Agent:

```
./ipaas-private-cloud-agent/scripts/mac/start.sh
```

Relevant commands

Below are the commands for stopping the Agent on different operating systems:

- macOS
- Linux
- Windows

Run the following command to stop the Agent:

```
./ipaas-private-cloud-agent/scripts/stop.sh
```

Generating Public and Private Keys

Step 1. Check the OpenSSL version

Run the following command to check whether OpenSSL has been installed:

```
openssl version
```

If the OpenSSL version information can be output normally after the command is executed, OpenSSL has been installed and you can skip step 2; otherwise, install OpenSSL as instructed below.

Step 2. Install OpenSSL

The OpenSSL installation method varies by operating system as follows:

- macOS
- Linux
- Windows

Run the following command to install OpenSSL:

```
brew install openssl
```

Step 3. Generate and update public and private keys

1. Run the following command to generate a private key:

```
openssl genrsa -out private.pem 1024
```

> Note：
> Place the generated private key in the `ipaas-private-cloud-agent/configs/secret` directory.
> 2. Run the following command to generate a public key for the private key. The `public.pem` file generated in the current directory is the public key.

```
openssl rsa -in private.pem -RSAPublicKey_out -out public.pem
```

3. To generate a new private key, replace the `private.pem` file in the `ipaas-private-cloud-agent/configs/secret` directory.

General Storage

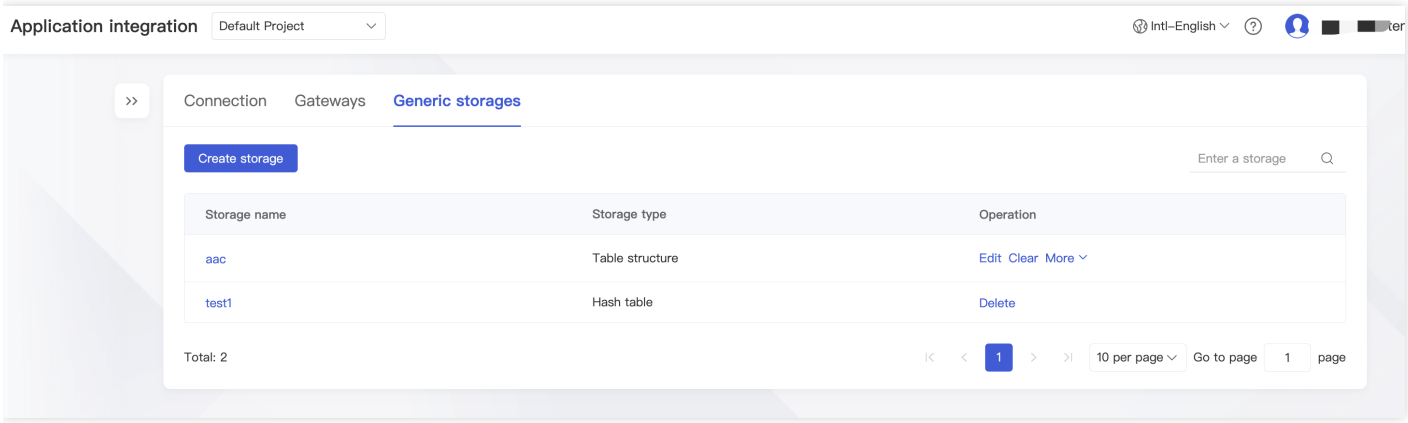
Last updated：2023-08-04 10:02:25

Use Cases

iPaaS provides various status management solutions. You can log in to the iPaaS console and click **Integration tools** > **General storage** on the left sidebar to manage storage structures and data used in your project.

General Storage Management

The **General storage** homepage is the storage management page, where you can create storages and view the list of all created storages, including storage name, storage type, and operations that can be performed on different storage types.



Creating a Storage

You can quickly create a storage simply by configuring the storage name and type.

- Storage name: Enter up to 25 letters and underscores.

- Storage type: Select table structure, hash structure, list, or string.

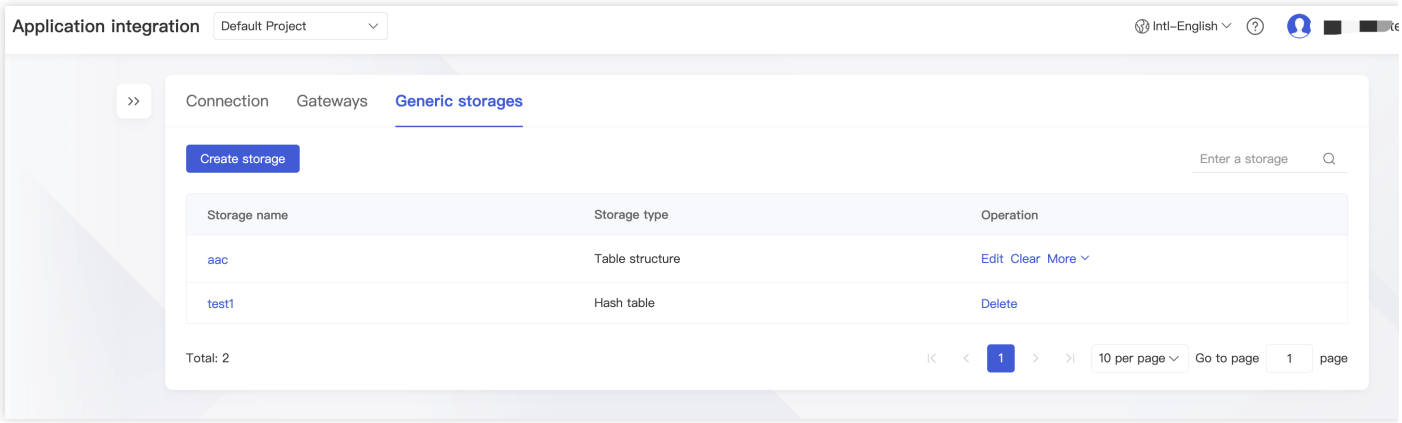
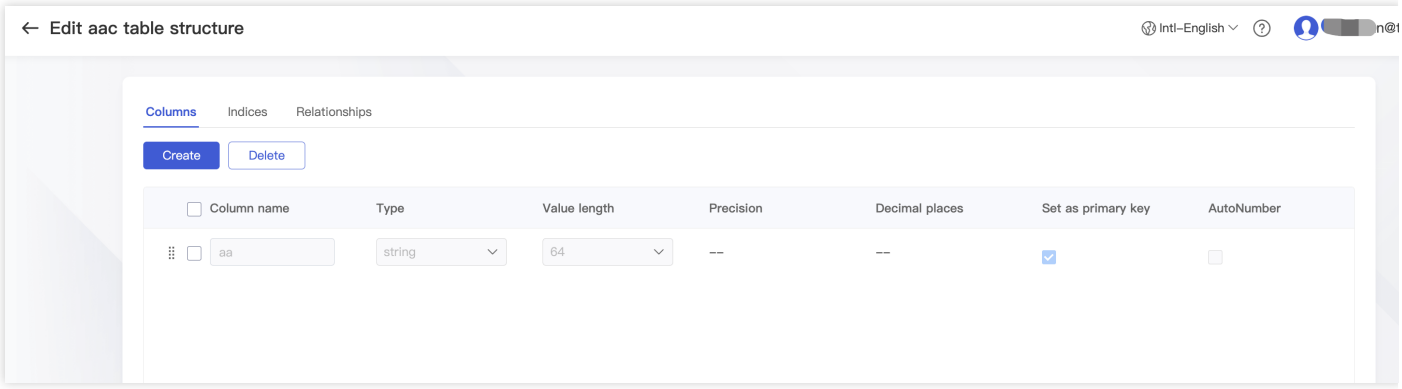


Table structure

Editing a structure

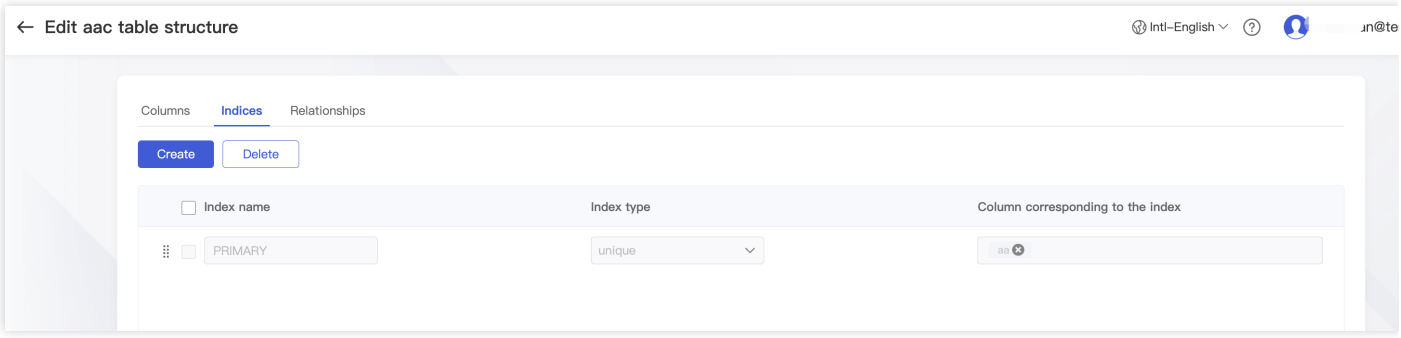
You can use the structure editing feature to maintain table structures, including column information, index information, and relationship configuration.

- **Column information**: You can add, modify, and delete a column in the table to quickly orchestrate the structure. Currently, you can edit the column name, data type (the following data types are supported: `string`, `bool`, `int`, `float`, `decimal`, `datetime`, `date`, and `time`), value length (if the type is `string`), precision, and decimal places (if the type is `decimal`), set a column as the primary key, and enable automatic numbering (only if the column is the primary key).

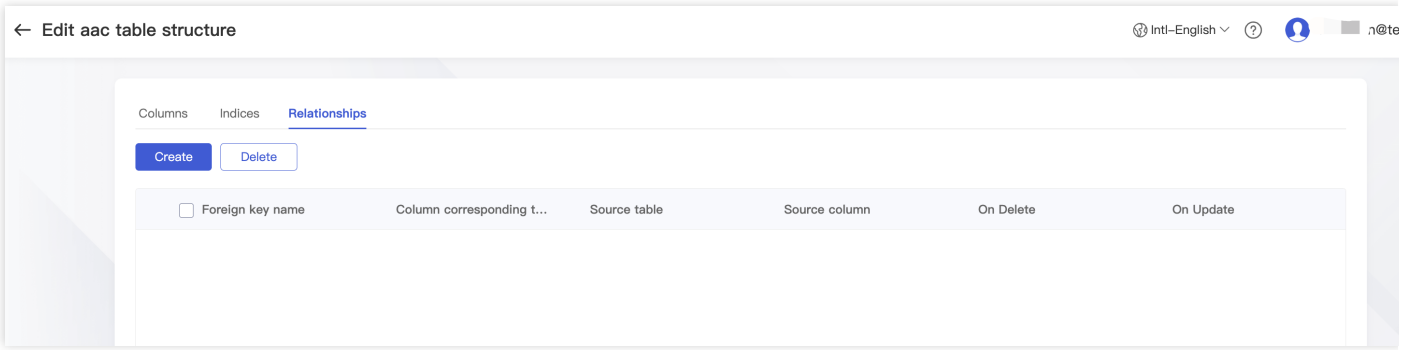


- **Index configuration**: The index configuration is similar to that in MySQL. On this tab, you can create indexes to accelerate MySQL search. Currently, you can create, delete, and name indexes, configure the index type (two types are supported: `index` and `unique`; for their differences, see 13.1.15 CREATE INDEX Statement), and

configure the index column (select a column in the maintained column list).



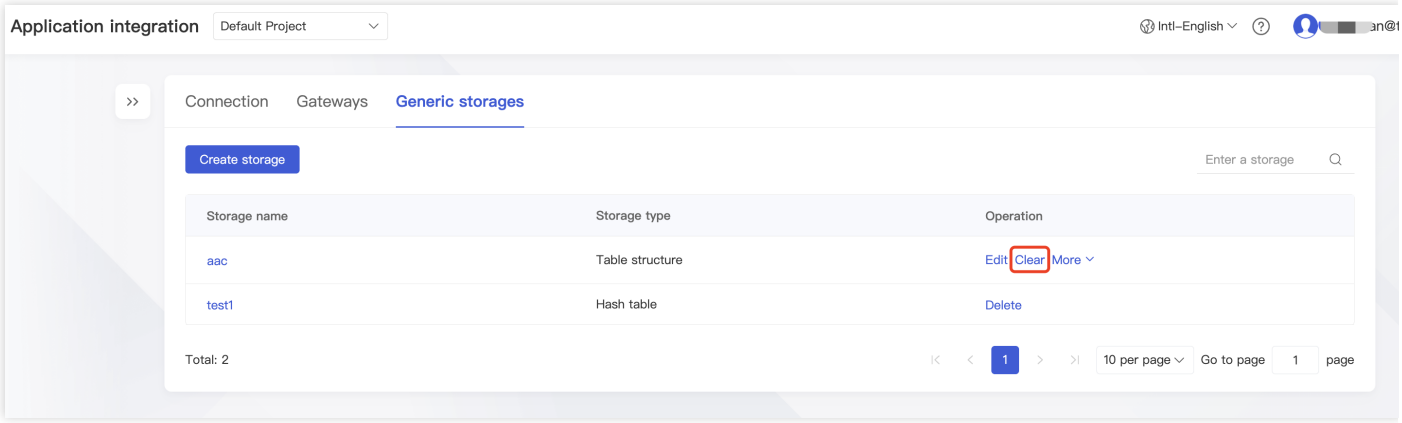
- **Relationship configuration**: You can also configure foreign key information for an iPaaS general storage. You can name a foreign key, configure the foreign key column, and select the foreign key source table and column. To configure foreign key event triggering, you can configure **On Delete** and **On Update** options as instructed in 13.1.15 CREATE INDEX Statement.



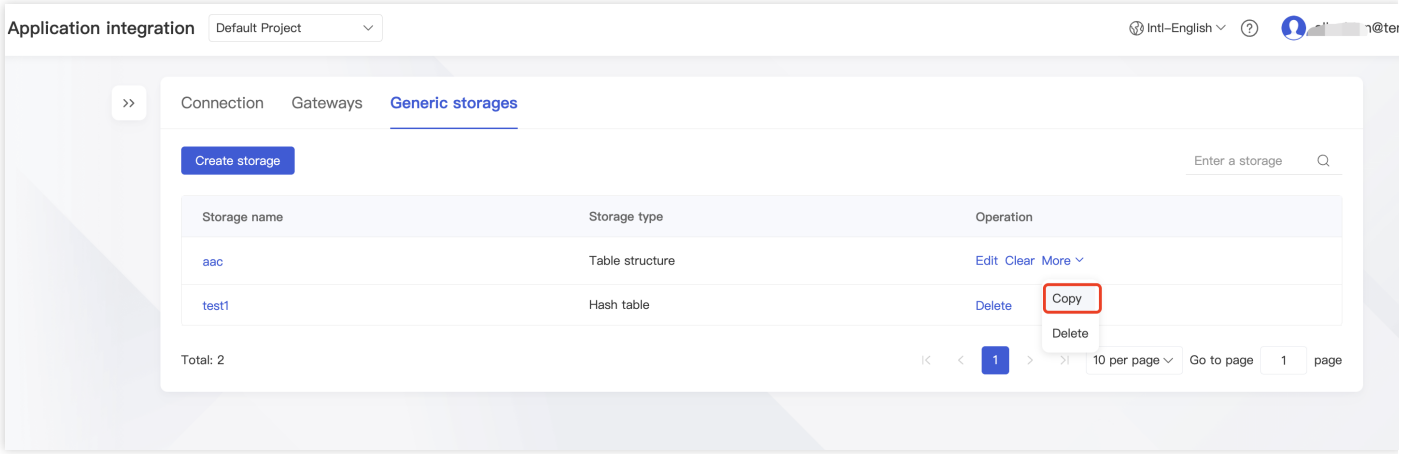
More operations

- **Clearing the table structure**: You can click **Clear** to clear all data in the current table structure.

> Note：
> This operation is a high-risk operation, after which all data in the storage will be cleared and cannot be recovered, so proceed with caution.



- **Copying a table with the same structure**: You can click **Copy table with same structure** to quickly create a table with the current structure, which facilitates data structure migration and backup.



- **Deletion**: You can click **Delete** to delete the corresponding storage record. **This operation is also a high-risk operation and will affect the data in the production environment, so proceed with caution.**

Hash structure

A hash is a data structure that can be directly accessed through a key value. It maps a key value to a position in the table to access the record, so as to accelerate search. iPaaS allows you to create hash table structures containing data in key-value format.

List

A list is also called an array, which is an ordered element sequence, i.e., a group of data elements with IDs. It is used to store a set of elements of the same data type.

String

A string consists of digits, letters, and underscores. You can create and use strings through the general storage feature in iPaaS, so as to quickly reuse strings to simplify operations.

Management Center
Member Management

Last updated：2023-08-03 17:27:29

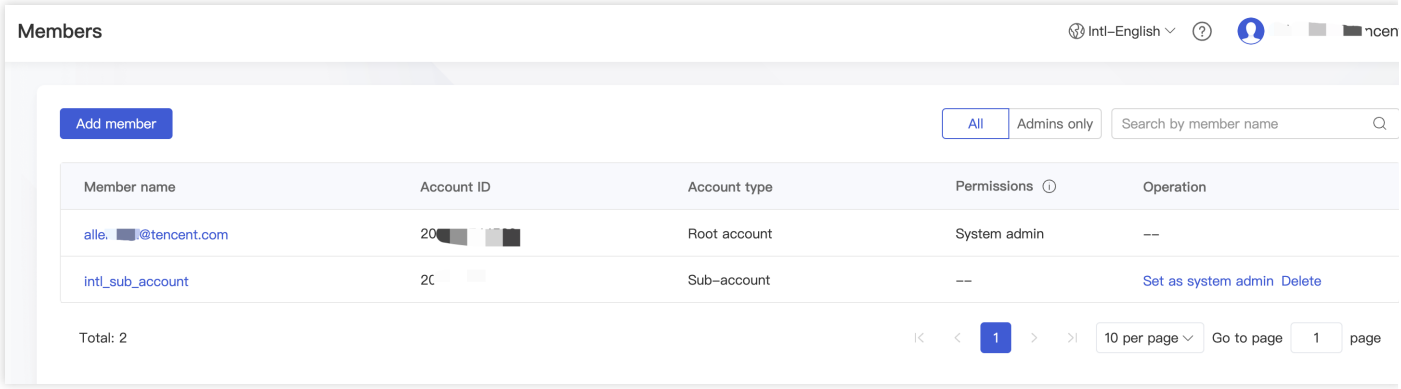
Overview

iPaaS combines the account system of CAM and iPaaS's own permissions for permission control.
iPaaS automatically pulls all sub-accounts under the current root account to the member list. To add/delete a member, you need to switch from the iPaaS console to the CAM console. The root account is the system admin, who has the highest privileges in iPaaS and can set or unset a sub-account as the system admin. For more information, see Member management.

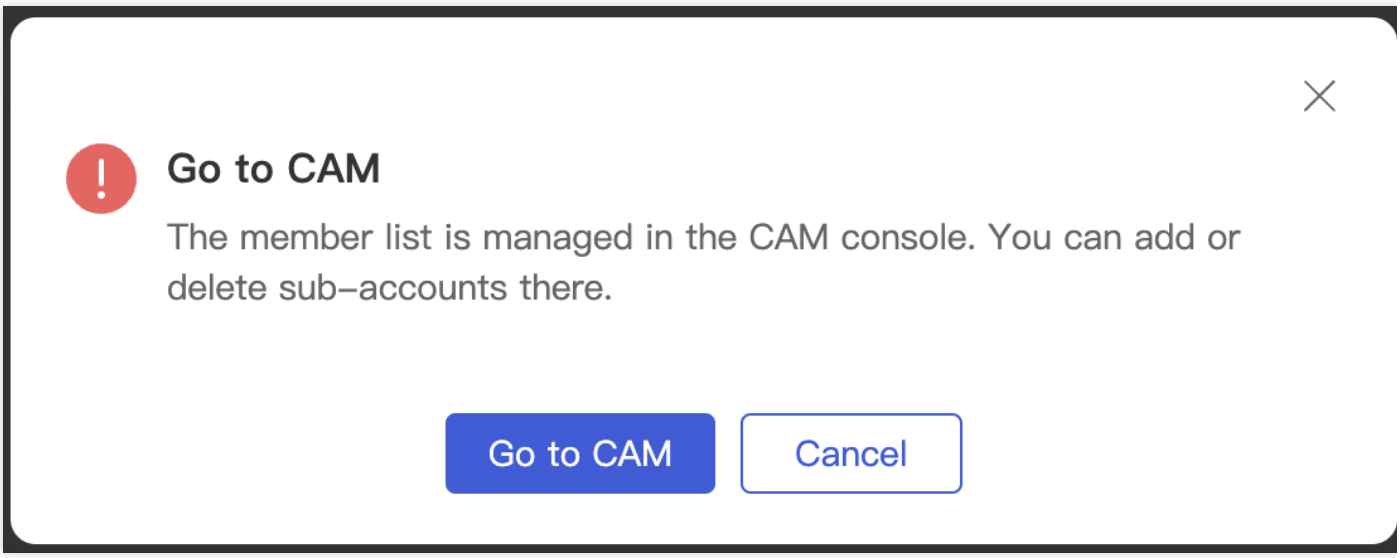
Directions

Adding/Deleting a member

1. Log in to the iPaaS console and click **Management center** > **Member management** on the left sidebar.
2. On the **Member management** page, click **Add member** or **Delete**.

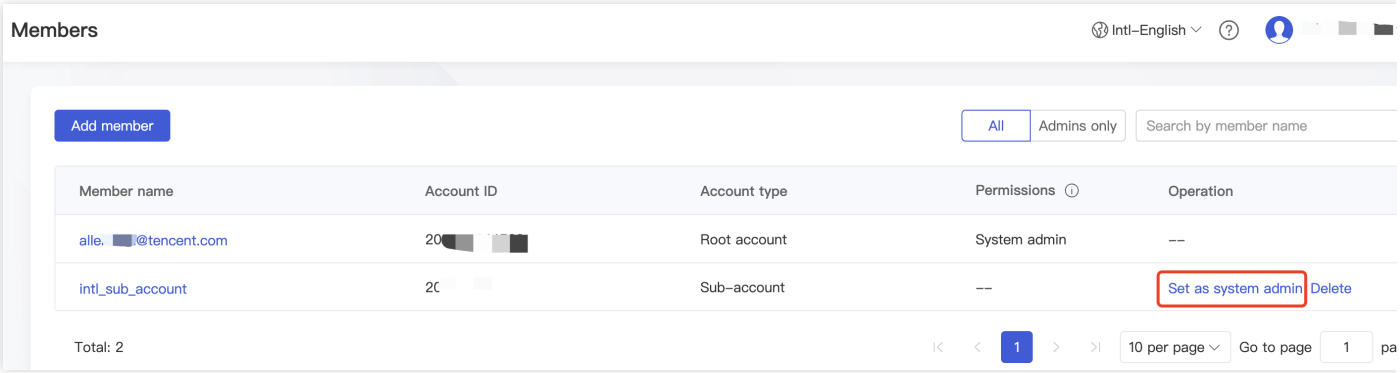


3. In the pop-up window, click **Go**, and you will be redirected to the user list page in CAM.



Setting/Unsetting as the system admin

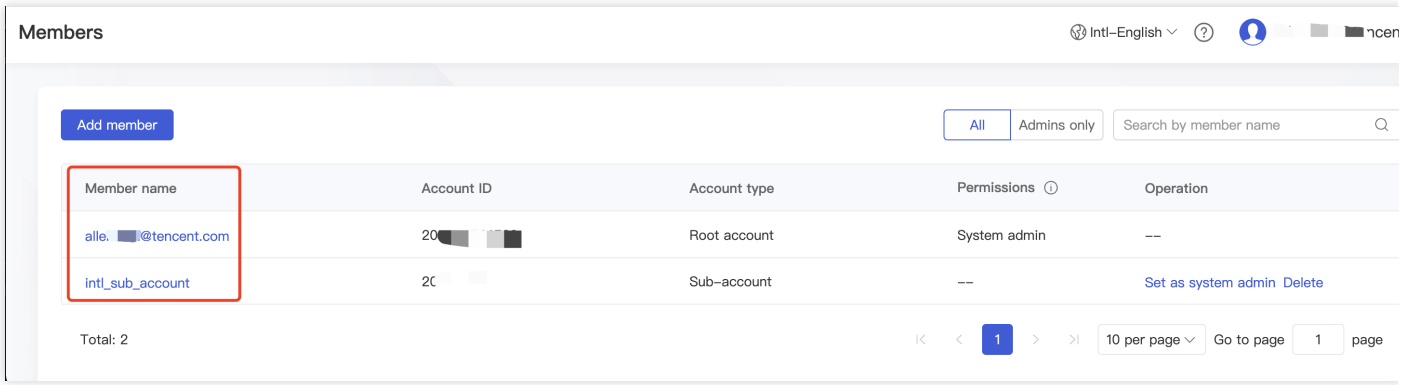
1. Log in to the iPaaS console and click **Management center** > **Member management** on the left sidebar.
2. On the **Member management** page, click **Set/Unset as system admin** in the **Operation** column. The root account is the system admin by default and cannot be unset.



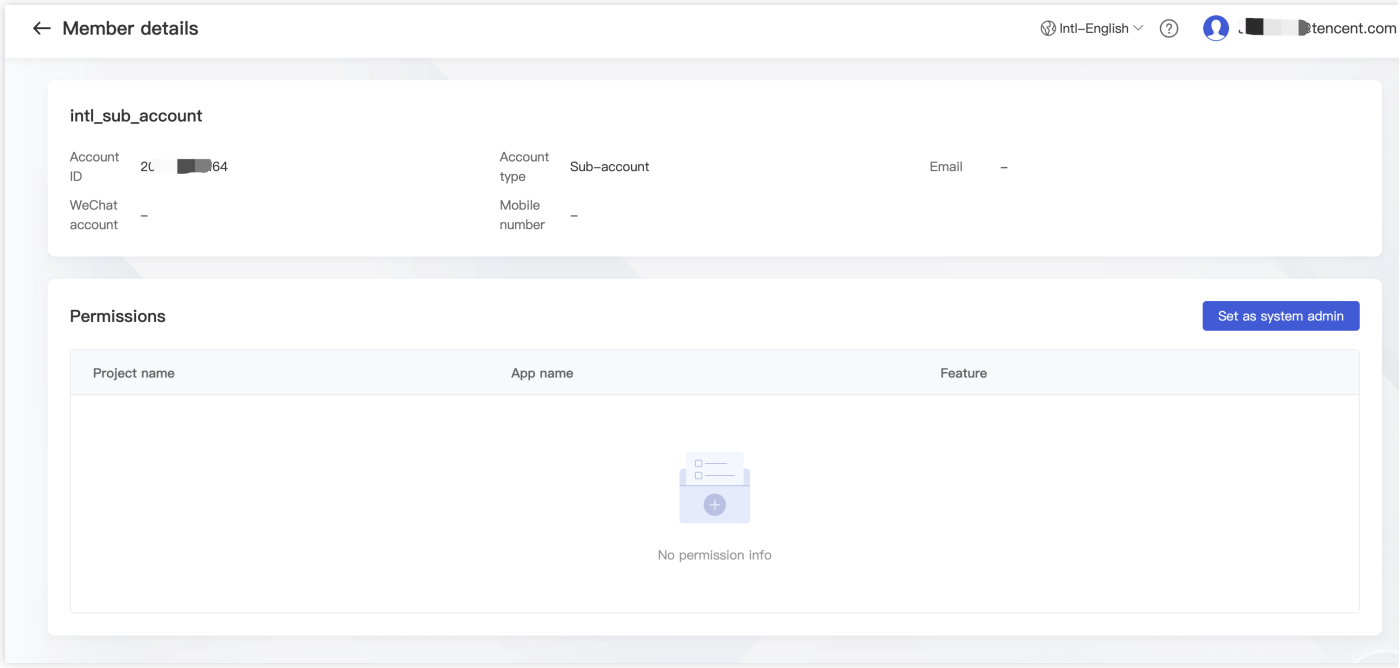
Viewing member details

1. Log in to the iPaaS console and click **Management center** > **Member management** on the left sidebar.

2. On the **Member management** page, click the name of the target member to view the member details.



The member details include the basic and authorization information as shown below:



Project Management

Last updated：2023-08-03 17:27:29

Overview

You can create integration apps to meet your integration needs. You can create multiple integration apps for different integration scenarios in each project.
After creating an integration app, you can view the project details or rename or delete the project.

- Project name: The list displays the names of all projects for which you have permissions.
  Default project: All members have the permissions for the default project by default. You cannot add or delete members of the default project, or delete or rename the project.
- Project admin: Up to three project admins are displayed.
- Description: You can enter text to briefly describe the project content so as to help members identify and understand the project.
- Project members: It is the number of project members.
- Creation time: It is the time when the project is first created.
- Operation: You can add members or rename or delete the project.

Prerequisites

You have created an integration app.

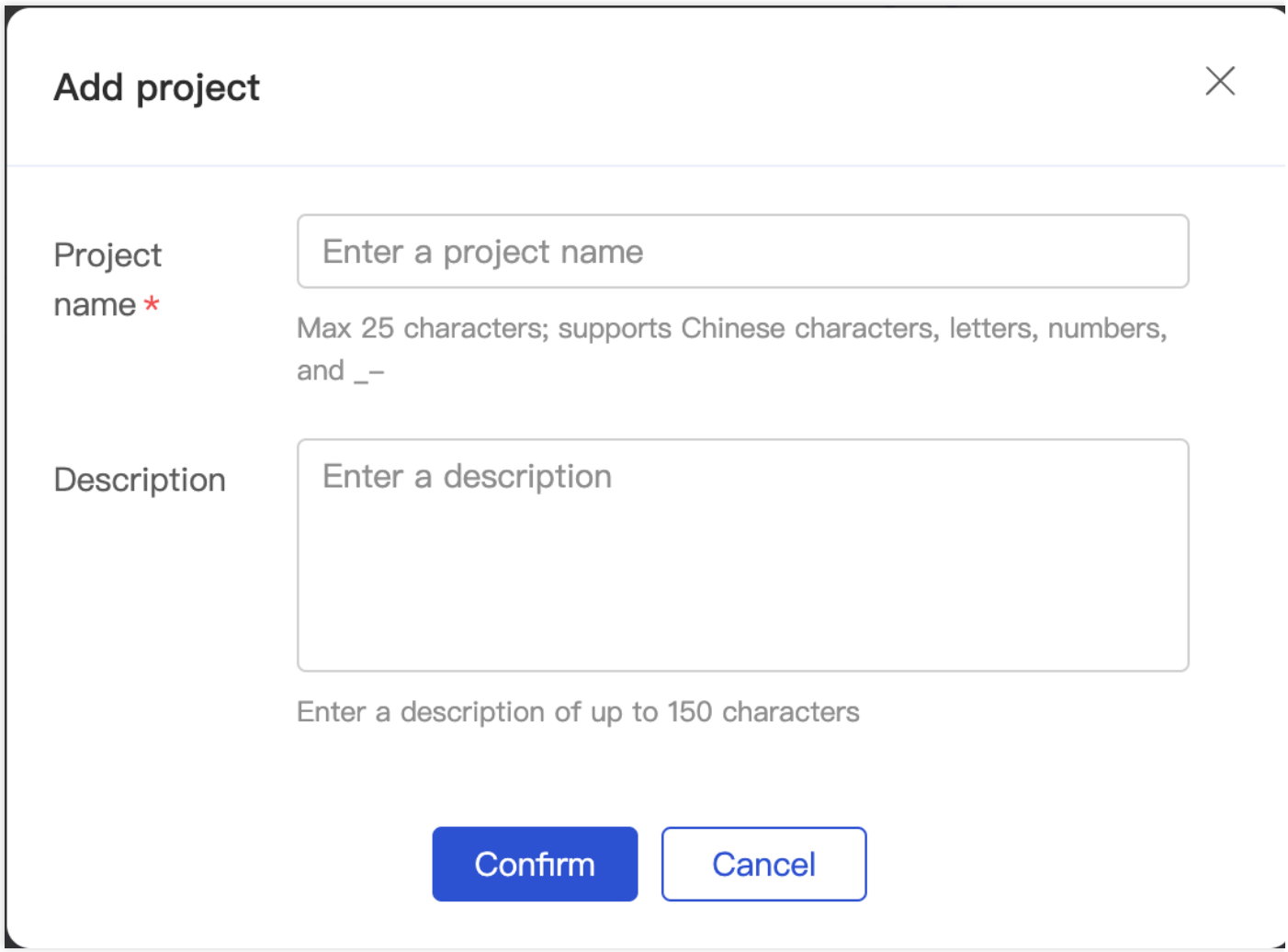
Directions

Adding a project

You can manage integration apps by project for business scenario identification and permission assignment. You can view the details of, rename, and delete projects.

1. Log in to the iPaaS console and click **Management center** > **Project management** on the left sidebar.
2. On the **Project management** page, click **Add project**.

3. Enter a project name and description and click **Confirm**.



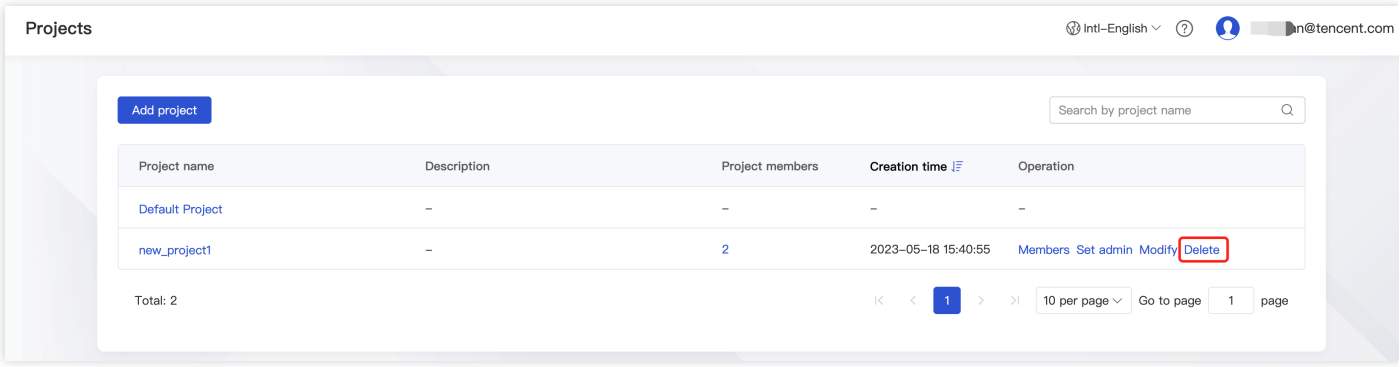
Deleting a project

If you no longer use a project, you can **stop** all apps in it and then delete it.

> Note：
> A project with running apps cannot be deleted. Once a project is deleted, its data cannot be recovered, so proceed with caution.

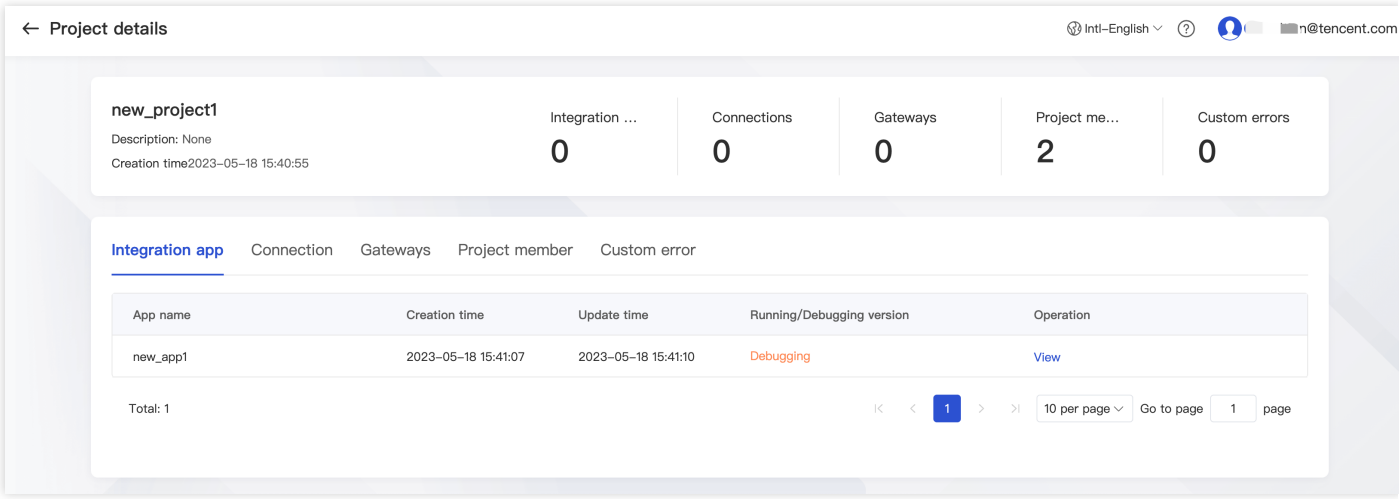
1. Log in to the iPaaS console and click **Management center** > **Project management** on the left sidebar.

2. On the **Project management** page, click **Delete** in the **Operation** column. In the pop-up window, click **Confirm**.



Viewing a project

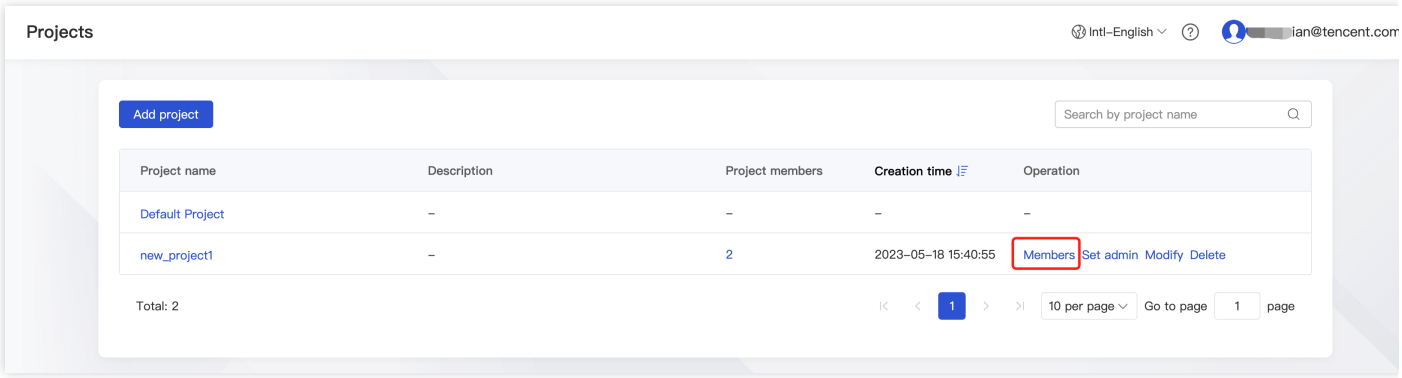
1. Log in to the iPaaS console and click **Management center** > **Project management** on the left sidebar.
2. On the **Project management** page, click the name of the target project to enter the **Project details** page.
   The **Project details** page displays the lists of integration apps, connections, security gateways, and custom errors under the project. You can click the target tab to quickly enter the editing page.



Member management

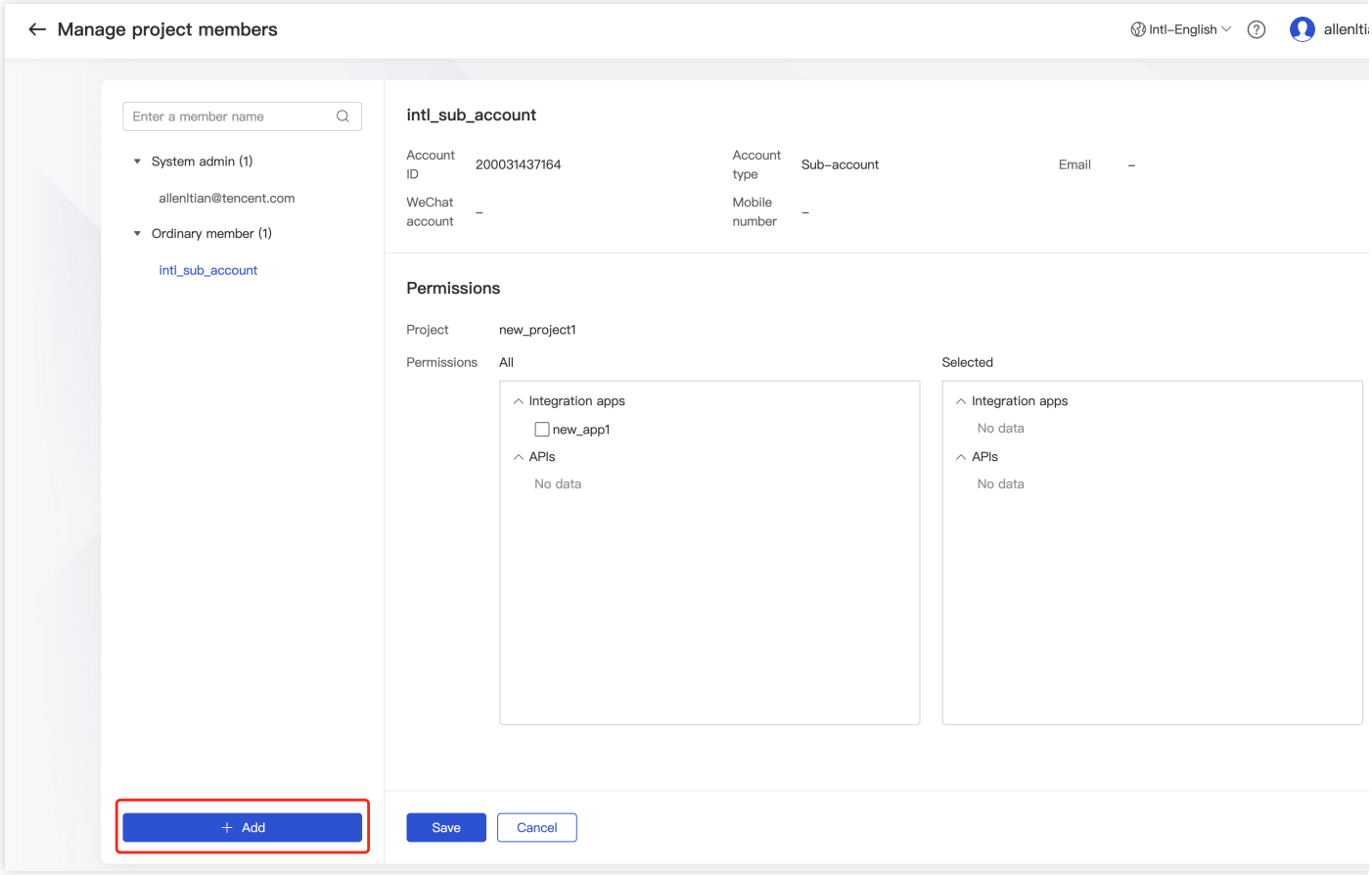
You can manage project members for collaborative development, Ops, and publishing by adding or deleting members. The list of members who can be added or deleted are from the member management module.
System and project admins can add or delete general members in the project, set or unset general members as the project admin, and modify the read-only, edit, and edit and publish permissions of general members for integration apps under the project.
iPaaS has a default project, for which all iPaaS members have permissions. In the default project, you cannot add or delete members, so the member management operation is not supported for the default project.

1. Log in to the iPaaS console, click **Management center** > **Project management** on the left sidebar, and select **Member management**.



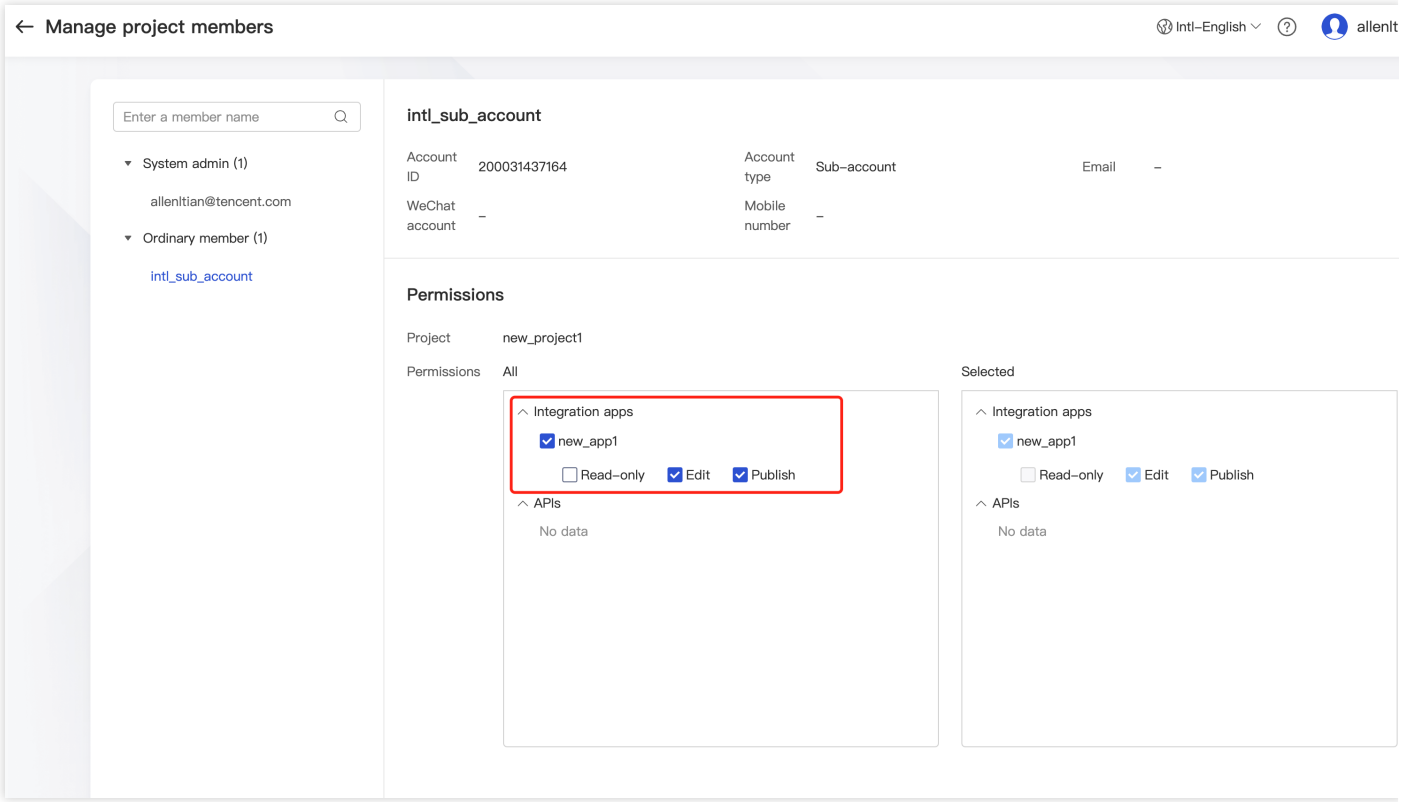
2. Add a member.

   You can add only members of existing sub-users in CAM.



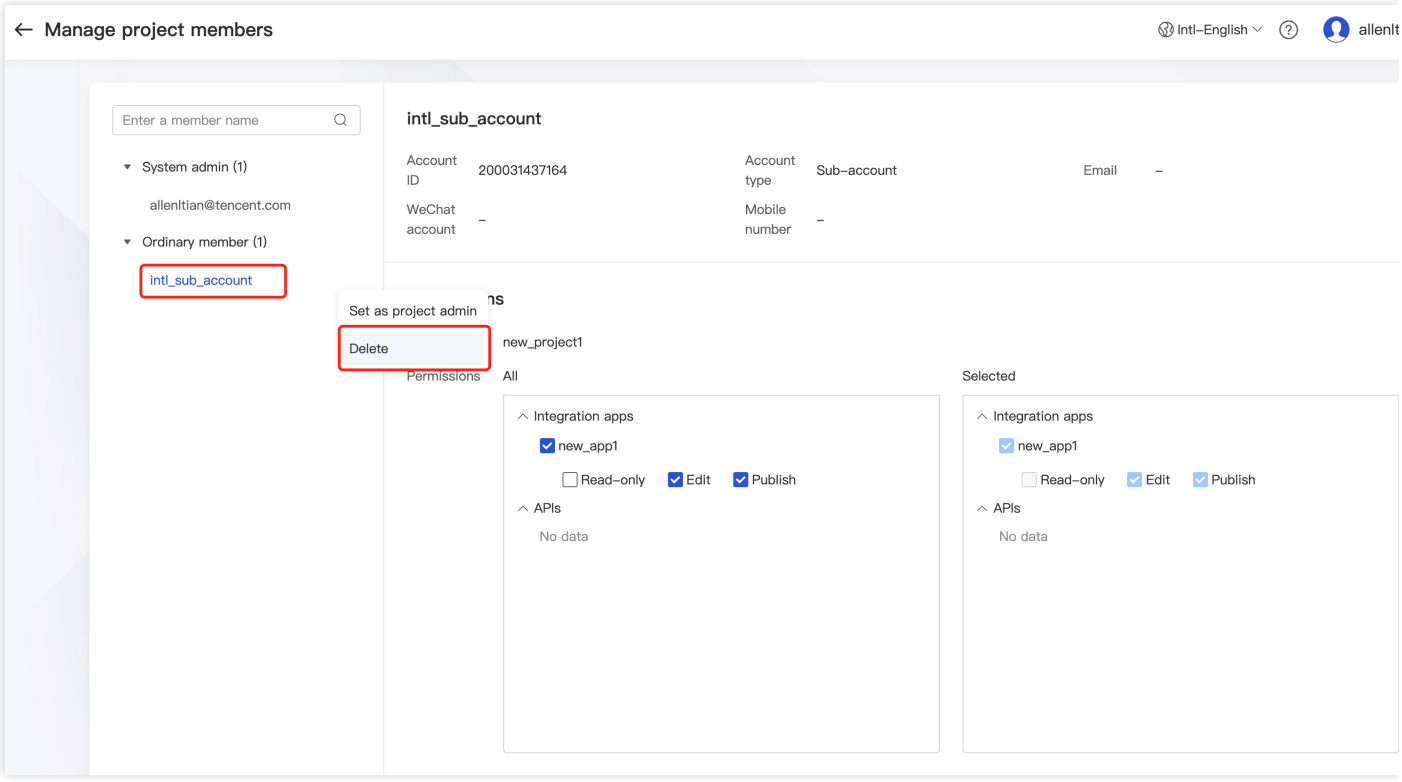
3. Configure member permissions.

   Grant the member the read-only, edit, or edit and publish permission for apps in the project.



4. Delete a member.

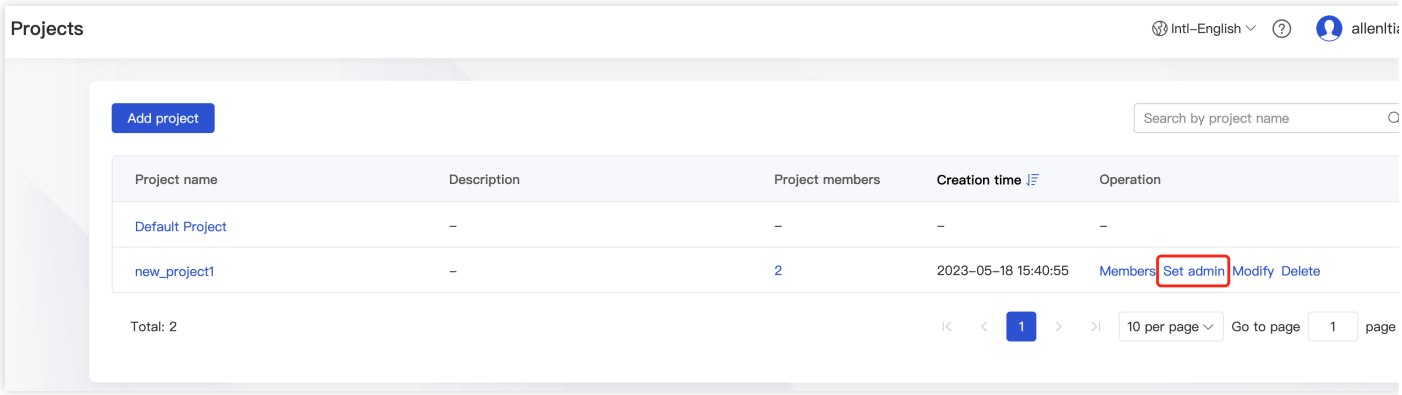
   Once deleted, a member no longer has the permissions for the project.



Setting an admin

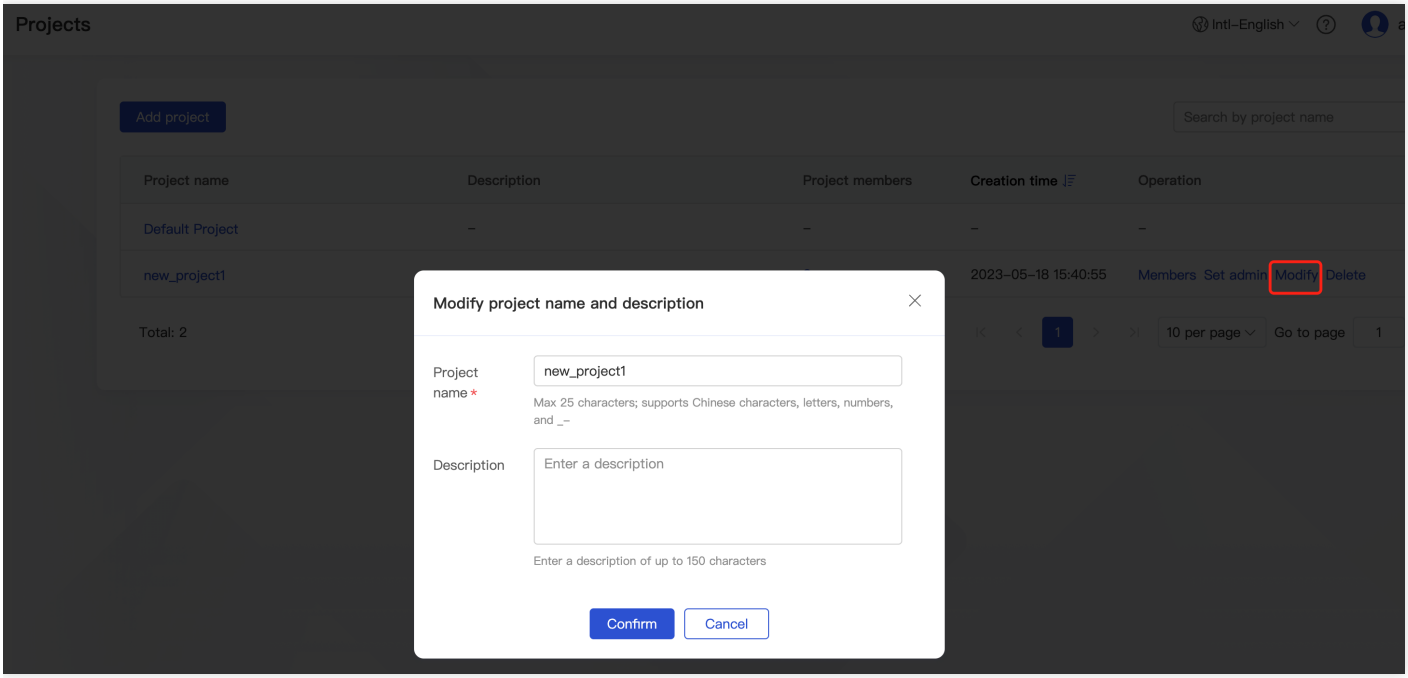
You can set a project admin for an existing project. After you set a project member as a project admin, the member will have the highest privileges for the project. Only members in the project can be set as a project admin.

1. Log in to the iPaaS console, click **Management center** > **Project management** on the left sidebar, and select **Set admin**. After the admin is set successfully, the corresponding member name will be displayed in the **Project admin** column.



Modifying a project

You can rename an existing project or modify its description.



Environment Management

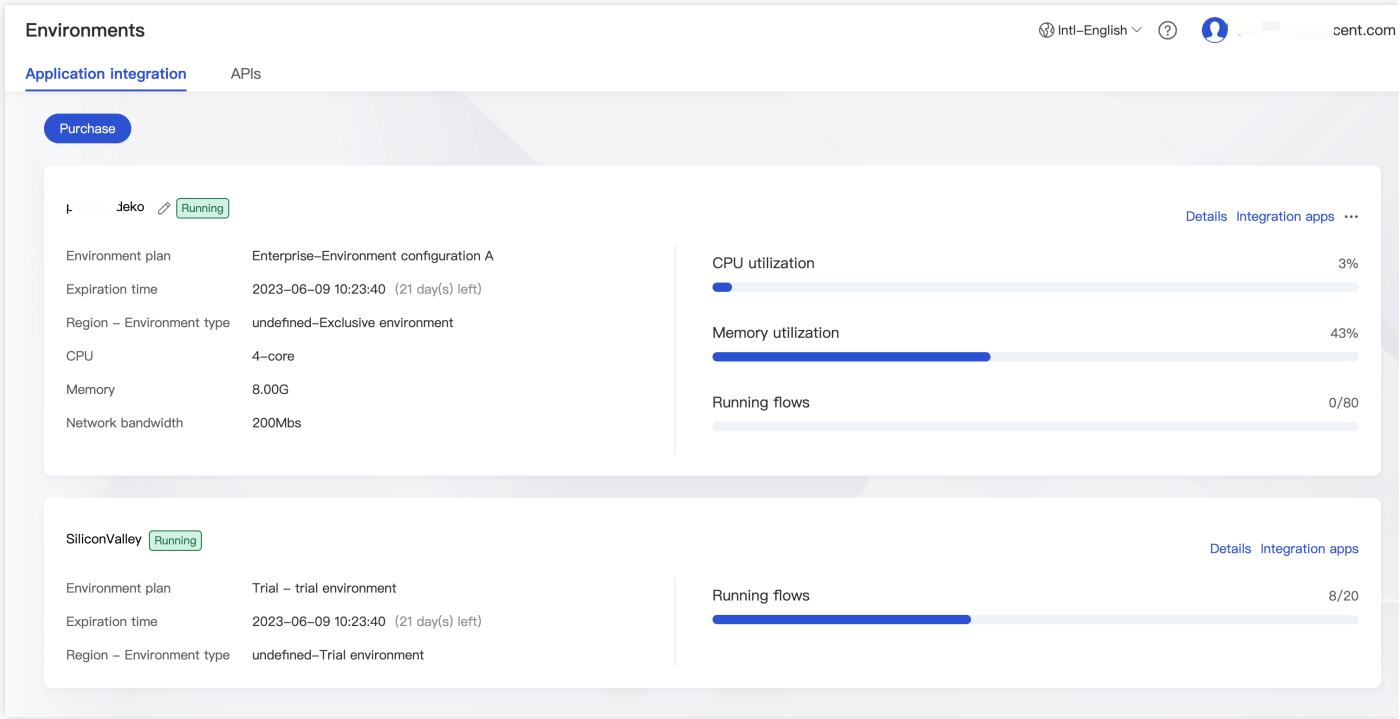
Last updated：2023-08-04 10:06:49

Overview

Environment management enables you to view and manage environments and runtime environments easily. You can view the configuration, usage, and Ops information of deployed environments at one stop.

Environment Management Page

1. Log in to the iPaaS console and click **Management center** > **Environment management** on the left sidebar.
2. The environment list page displays the basic information of each environment as shown below:



- **Environment name**: It displays the names of all environments for which you have permissions.
- **Status**: It displays the instantaneous status of each environment. Currently, you cannot create, delete, or extend environments in the console. The platform supports the following environment statuses (when an operation is performed on the backend, the status displayed on the frontend will be updated), and the environment availability varies by status:

Environment Status	Availability

Environment Status	Availability
Creating	Apps cannot be published
Running	Apps can be published
Extending	Apps can be published
Deleting	Apps cannot be published
Unavailable	Apps cannot be published
Creation failed	Apps cannot be published
Services suspended	Apps cannot be published

- **Region - environment type**: It displays the list of exclusive and shared environments. Currently, iPaaS supports two environment types with different environment deployment characteristics and resource capabilities.
  - Shared environment: It is a high-availability iPaaS runtime environment shared by all users for quick app publishing.
  - Exclusive environment: For data and resource security considerations, iPaaS also provides a deployment mode with stronger resource and data isolation, i.e., exclusive environment. You can use an iPaaS runtime environment exclusively to isolate computing resources and data and quickly increase the isolation level at low costs. Currently, you can create an exclusive environment by clicking **Purchase environment**.
- **Environment plan**: Enterprise (this field is displayed only for Enterprise Edition exclusive environment).
- **Expiration time**: Expiration time of the environment.
- **Resource**: The configuration information of CPU, memory, and network bandwidth resources are displayed only for Enterprise Edition environments during purchase or after upgrade but not for the shared environment.
- **Resource overview**: It displays the actual resource usage for Enterprise Edition environments but not for the shared environment.
- **Operation**: Currently, operations such as app management, execution overview viewing, and information configuration are supported.

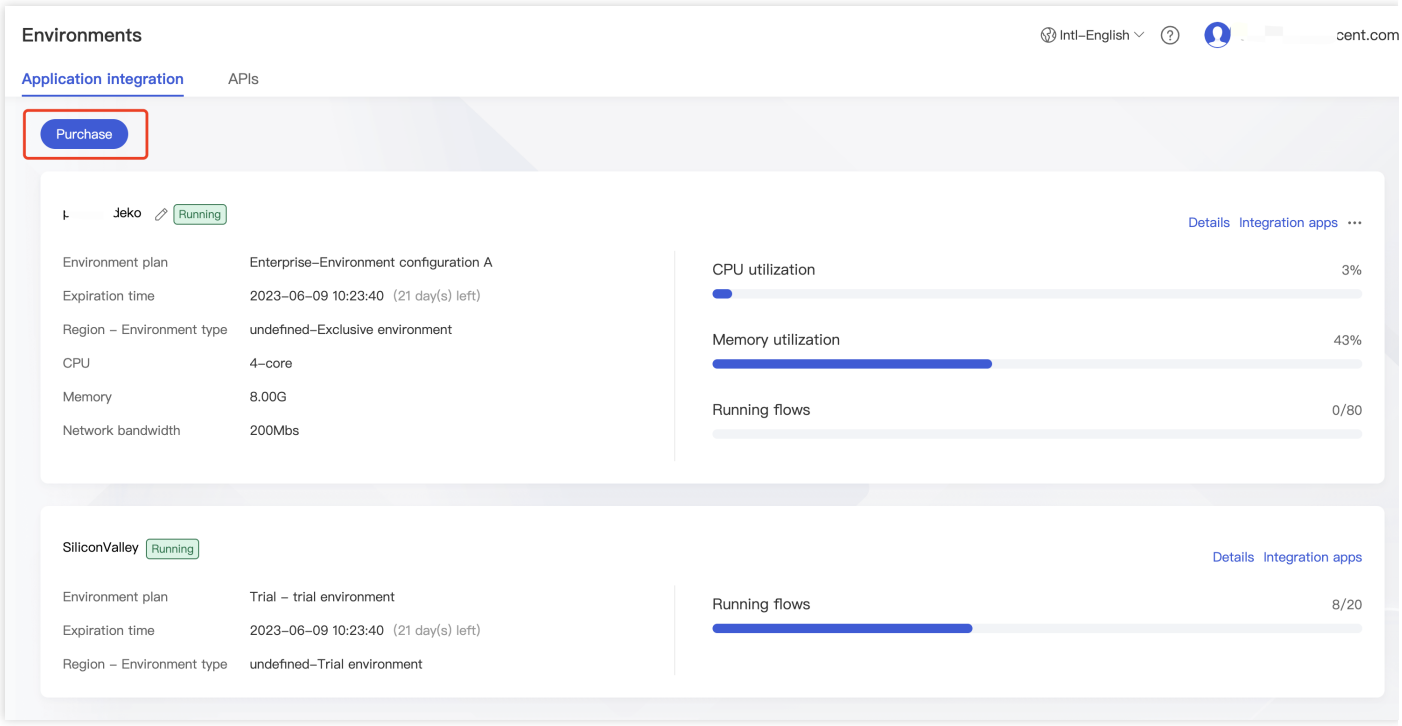
Environment Management Operations

Purchasing an environment

Note：
After being purchased, the environment will be in **Creating** status for one to three minutes. Wait patiently.

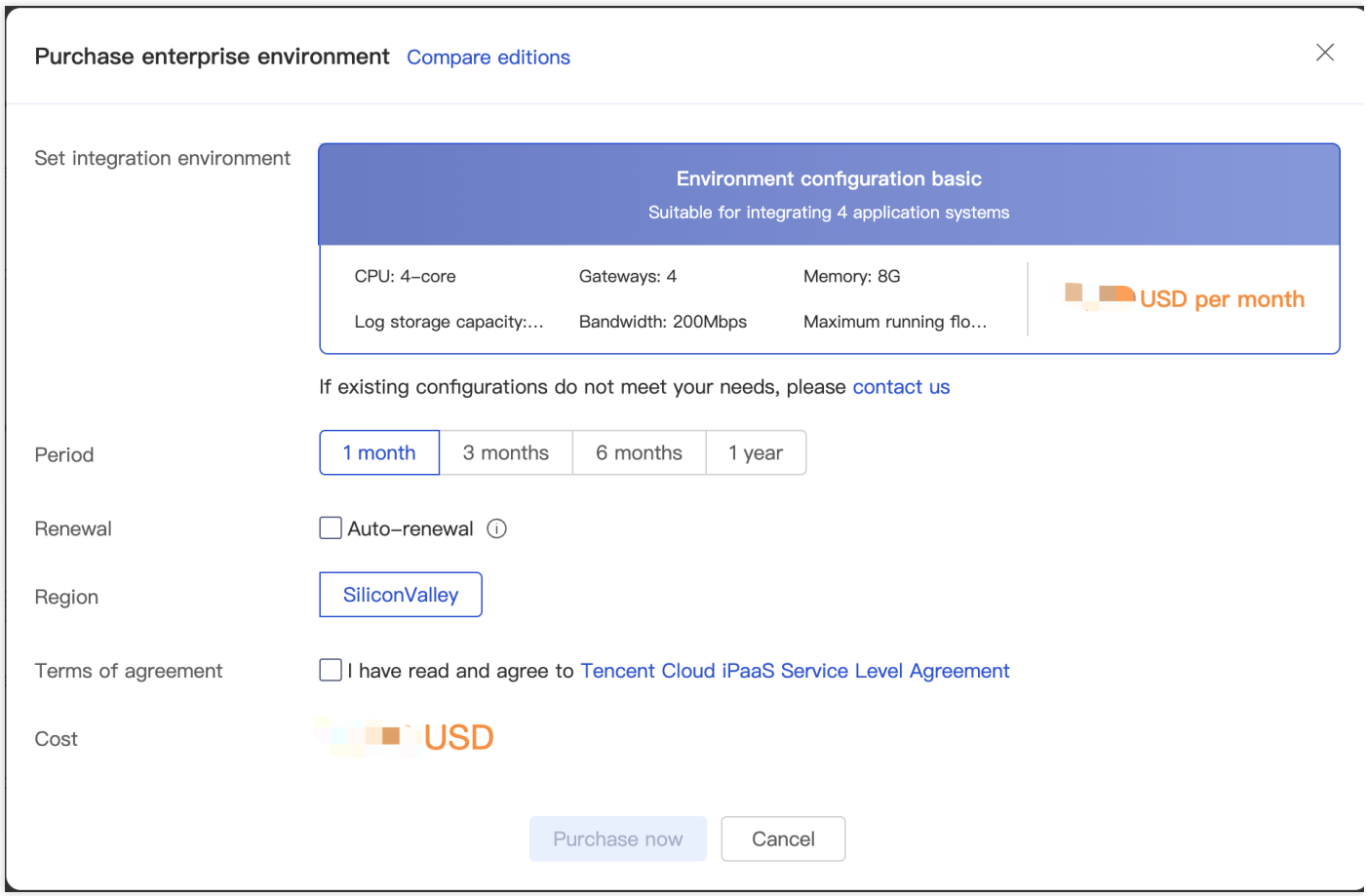
The **shared environment** is suitable for new users to test the basic product features, while an exclusive environment (Enterprise Edition) is more suitable for deploying businesses.
An **exclusive environment** provides a deployment mode with stronger resource and data isolation for data and resource security considerations. You can use an iPaaS runtime environment exclusively to isolate computing resources and data and quickly increase the isolation level at low costs. It is suitable for deploying businesses.

1. Log in to the iPaaS console, click **Management center** > **Environment management** on the left sidebar, and click **Purchase Enterprise Edition**.



2. On the configuration selection page, select an environment configuration based on your business needs. If you are worried that a high configuration may lead to resource waste, you can select the basic environment configuration

first and **change the configuration** subsequently. For more information on purchase, see Purchase Guide.



Renewal and upgrade

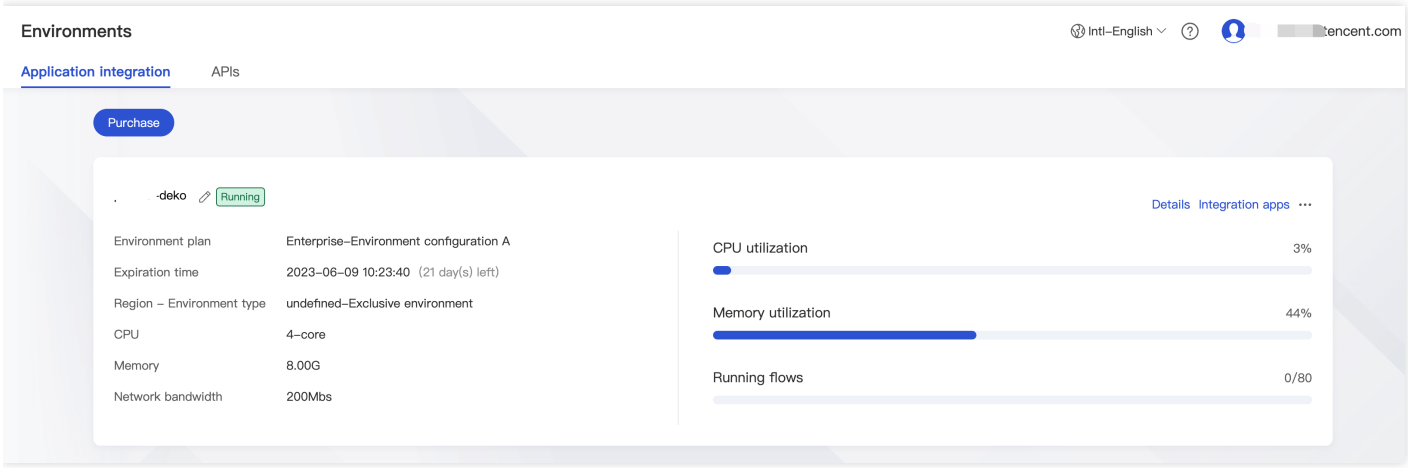
You can renew and upgrade Enterprise Edition environments on the **Environment management** page.

Upgrade

Please contact online customer service staff.

Renewal

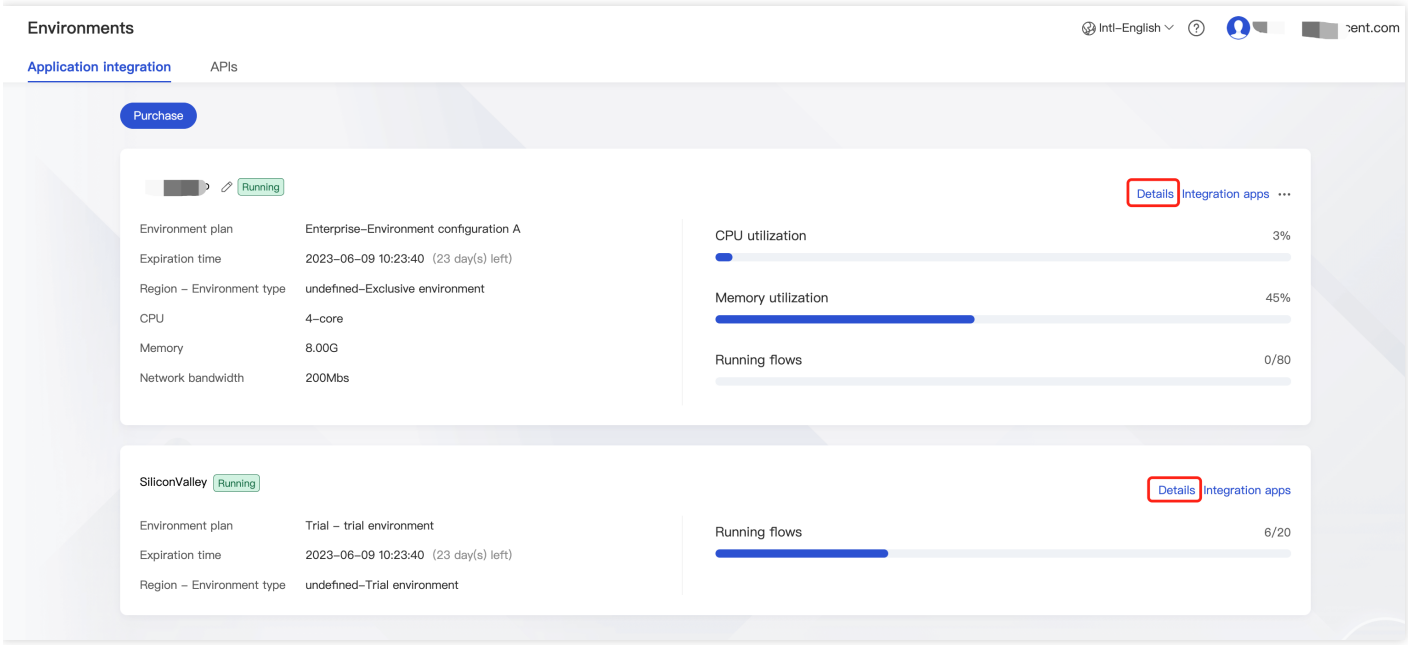
Log in to the iPaaS console, click **Environment management**, and click *Renew**.



Select a renewal period.

Viewing environment details

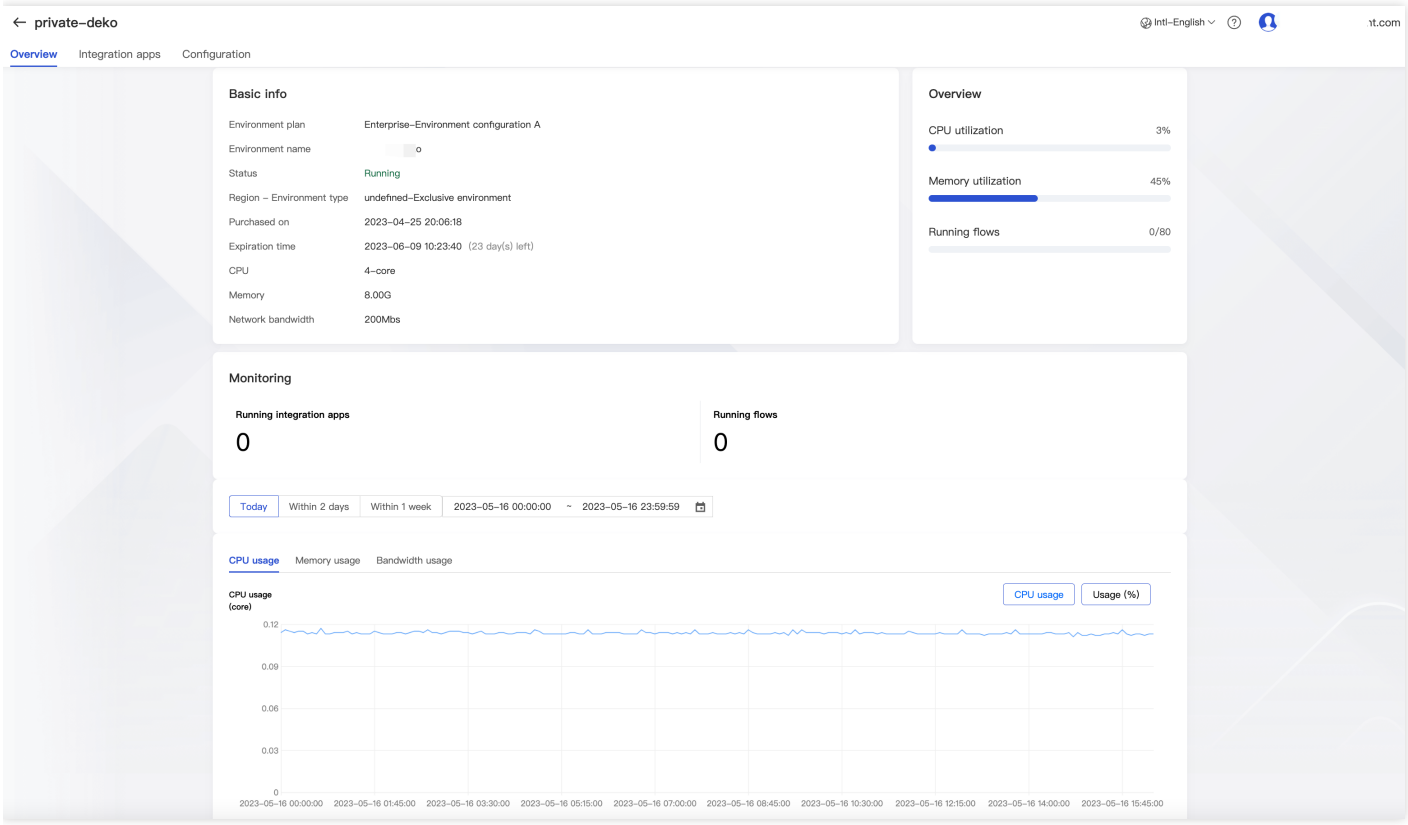
Click **Details** to view environment details, including overview, app management, and configuration details (available for exclusive environments).



- Overview
- Applications
- Configuration

The **Overview** tab centrally displays information such as environment name, status, region, and type and the number of running integration apps, but the displayed fields differ slightly for shared and exclusive environments as shown

below (fields marked in red boxes are displayed only for Enterprise Edition (exclusive) environments):



- Shared environment: The overview of the shared (Trial Edition) environment displays the basic information, region, status, and other information of the environment as well as the numbers of integration apps and flows running in the environment.
- Exclusive environment: The overview of an exclusive (Enterprise Edition) environment is similar to that of a Trial Edition environment. However, in addition to the basic information, region, status, and other information of the environment as well as the numbers of integration apps and flows running in the environment, it also displays the usage and running status (such as CPU and memory usage) of the current environment.

Audit Log

Last updated：2023-08-03 17:27:29

Overview

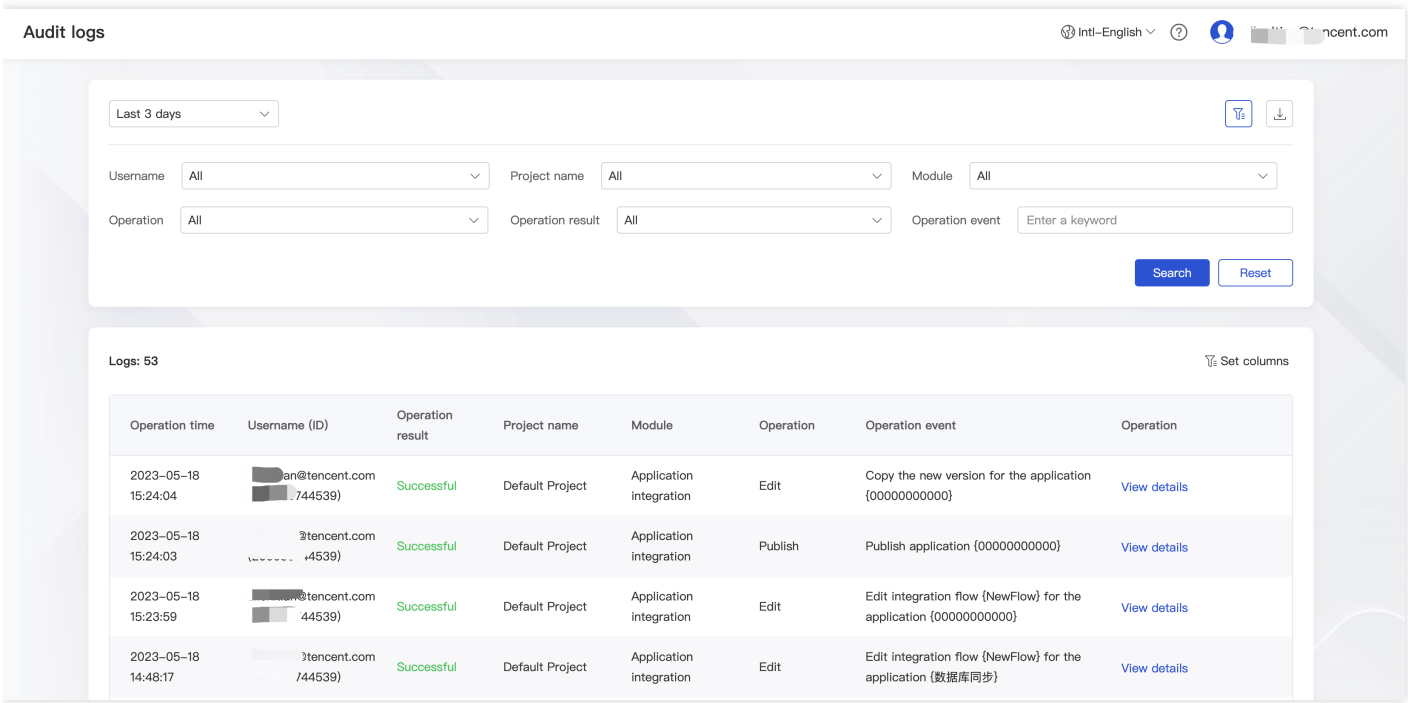
Audit logs enable you to view the information of historical operation events in the specified period of time, including basic information, operation information, and event details. Generally, audit logs can be used in scenarios such as compliance audit, project change tracking, and security analysis.

Directions

Step 1. Query audit logs

The console displays the audit logs in the last 24 hours by default. You can set filters to view the desired audit logs.

> Note：
> Currently, you can search for audit logs in the last 30 days in the console. To search for earlier logs, submit a ticket for assistance.

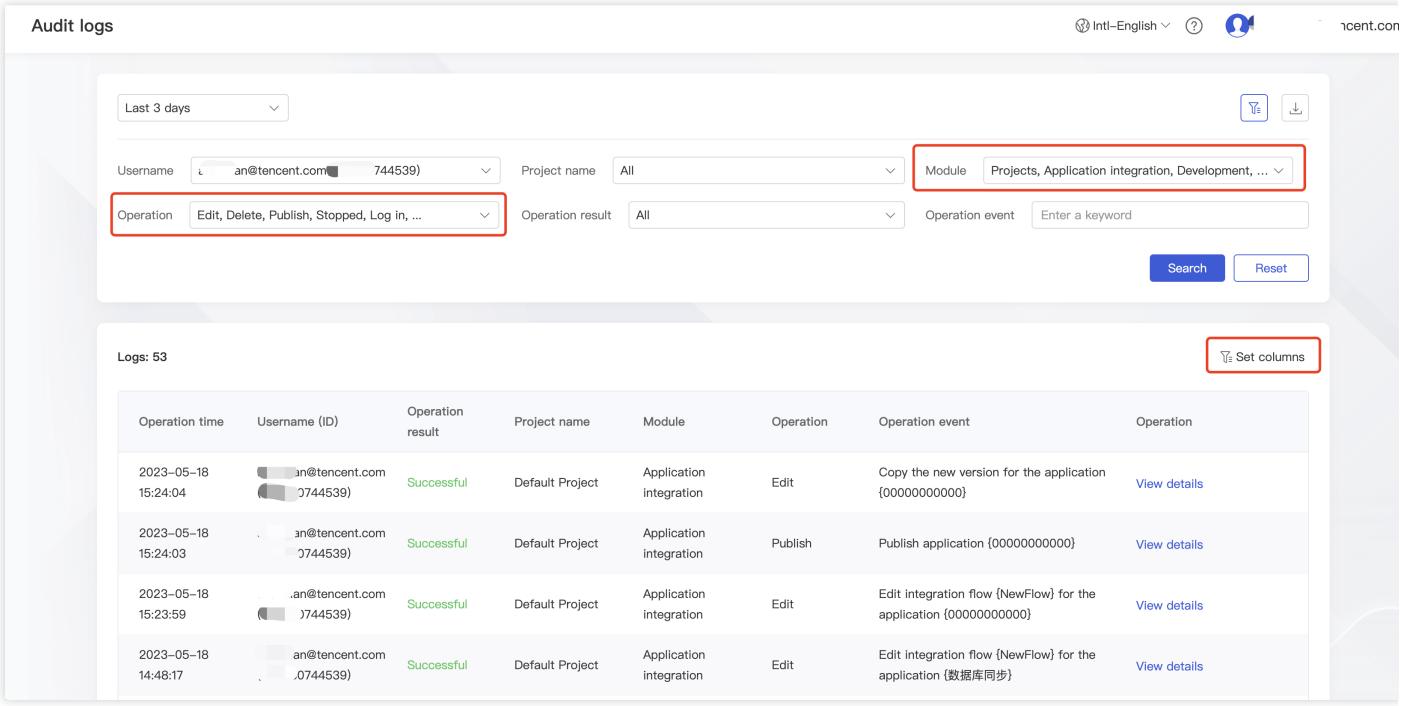


1. Log in to the iPaaS console and select **Management center** > **Audit log** on the left sidebar.
2. Set filters, which include:

Field	Remarks
Operation time	You can search by operation time. You can select the preset last 5 minutes, last 15 minutes, last hour, last 6 hours, last day, last 3 days, last 7 days, last 30 days, today, yesterday, the day before yesterday, this week, last week, or a custom time period.
Username	You can search by username or account ID to view the audit logs of the specified user.
Project name	You can search by project name to view the audit logs of the specified project.
Module	You can search by module such as app integration, connection management, security gateway, operation monitoring, general storage, integration resource, project management, API management, or API user center.
Operation type	You can search by operation type such as creation, editing, deletion, publishing, stop, login, or logout.
Operation result	You can search by operation result such as success or failure.
Custom search box	You can search by entering an operation event keyword. Generally, this filter is suitable for exception tracking.

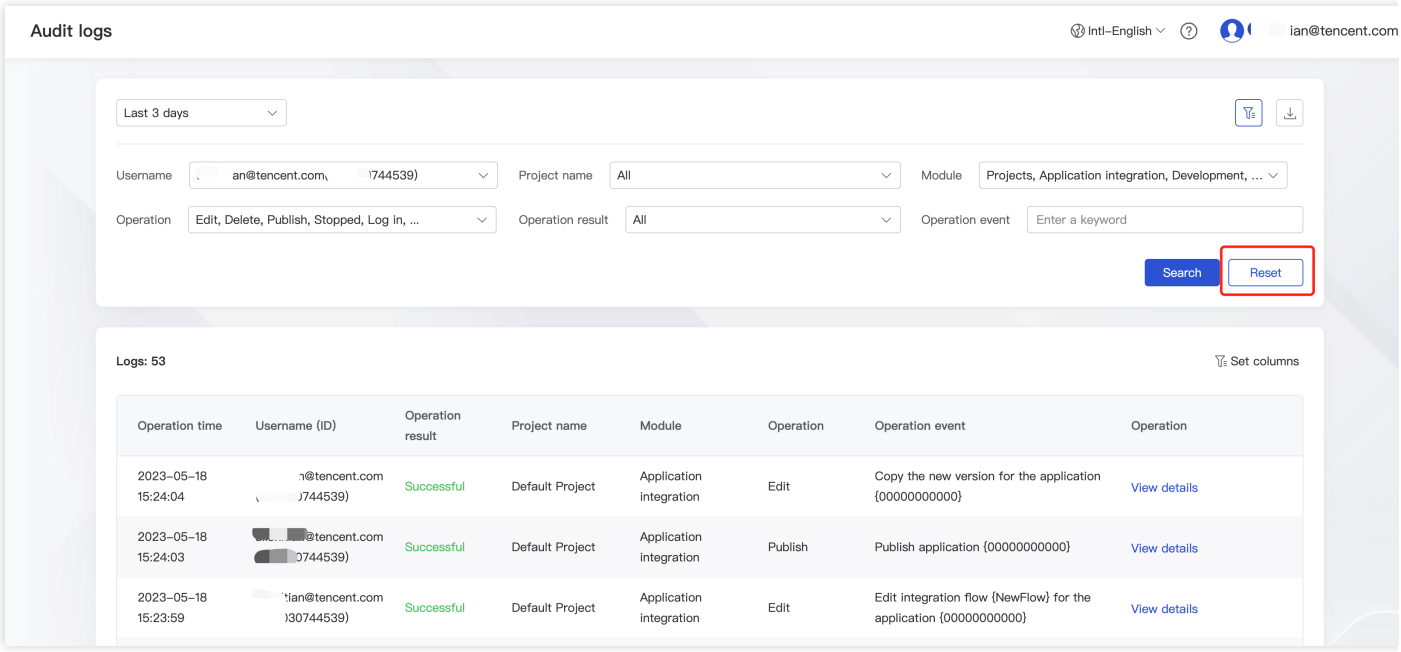
3. View logs. The audit logs meeting the specified filter conditions are displayed and sorted in reverse chronological order. The log list contains the following:

Field	Remarks
Operation time	The time when the operation event occurred.
Username	The username and account ID of the user executing the operation event.
Operation result	Whether the operation event is executed successfully. If the result is failure, the operation is not executed, and you can view the error message.
Project name	The project of the operation event.
Module	The feature module of the operation event.
Operation type	The abstract operation type of the operation event.
Operation event	The content digest of the operation event.
Operation	**View details**. You can view the basic information, operation information, and event details as instructed in step 2.



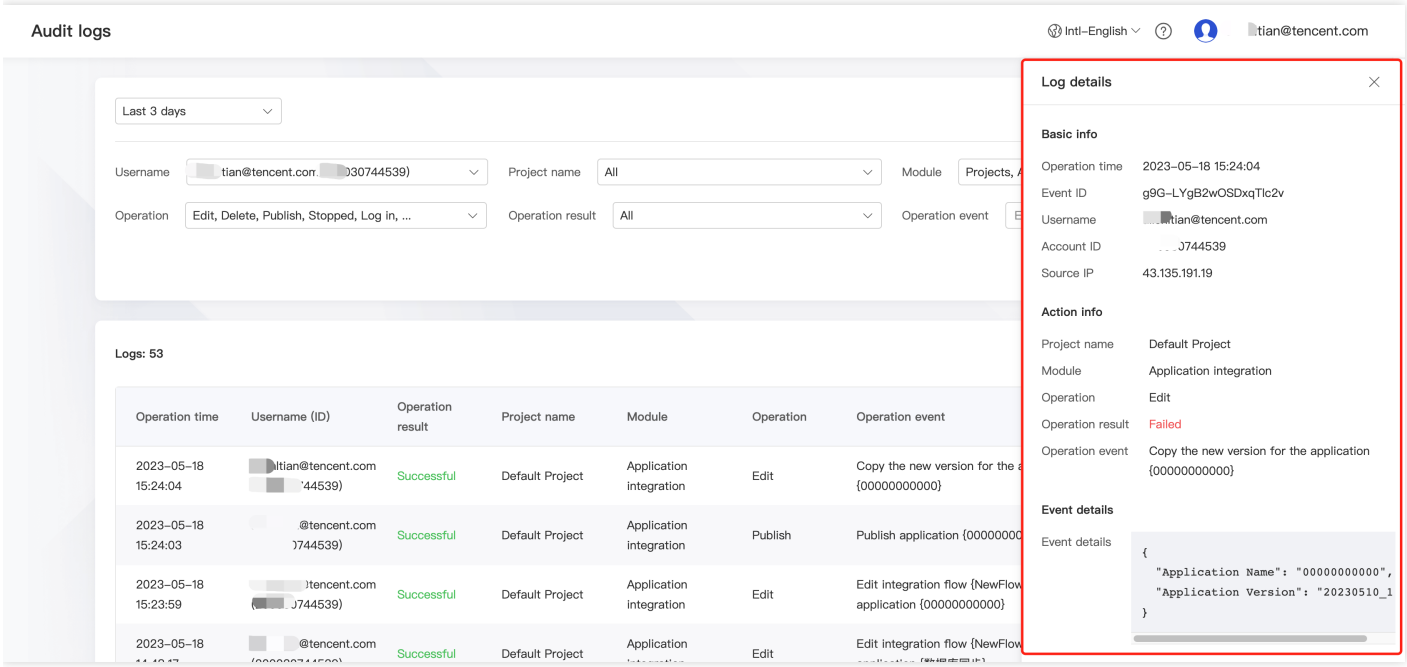
4. Reset logs filters.

   After filtering and querying logs, you can click **Reset** to quickly restore to the logs displayed by default.



Step 2. View log details

For a concerning or suspicious audit log, you can click **View details** to view its details. In addition to the information on the list page, the following information is also displayed:



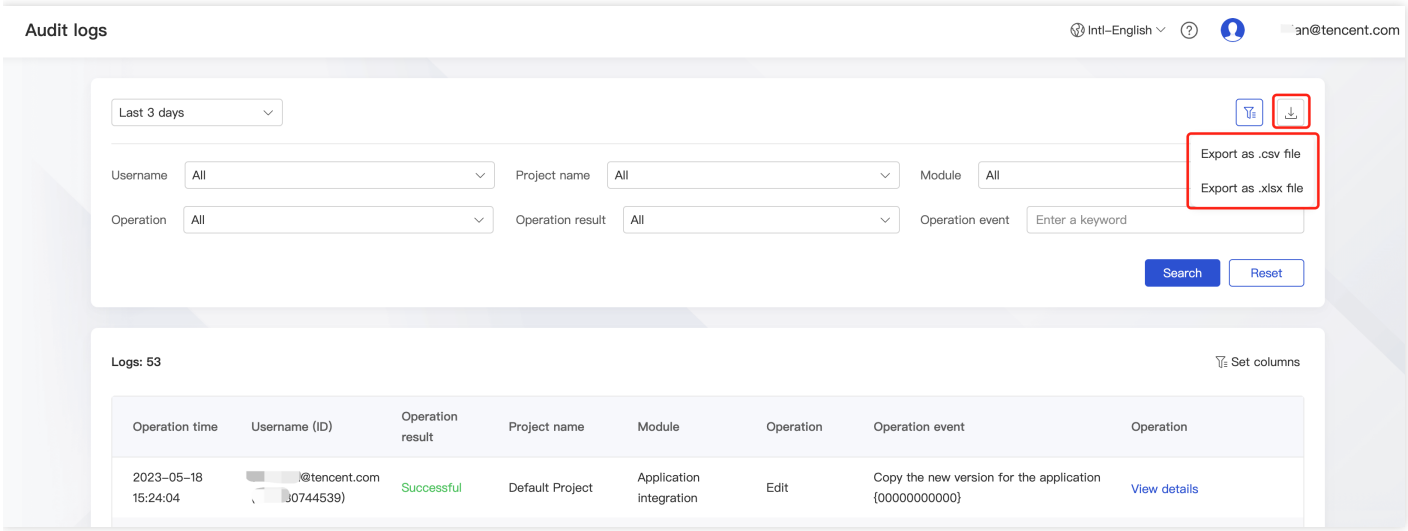
Type	Field	Remarks
Basic info	Event ID	The unique ID of each audit log.
	Source IP address	The source IP address of the user who executed the operation event.
Operation info	Error message	The error message displayed when the operation result is failure.
Event details	Event details	The request parameters of the operation event. For example, if the operation event is `API service{API service name} creation`, information such as API service name, protocol, description, tag, and version will be displayed here.

Step 3. Download logs

You can download logs on the platform. You can export audit logs in the current list as an .xlsx or .csv file for further analysis or sorting. For detailed fields, see the log list.

Log in to the iPaaS console, select **Audit log** on the left sidebar, click the download icon, and select **Export as .xlsx**

file or **Export as .csv file**.



Dataway Expression Overview

Last updated：2023-08-03 17:51:33

Dataway is a scripting engine for custom flow execution data conversion and processing in iPaaS. It is integrated in iPaaS and plays a key role in implementing iPaaS extensibility.

iPaaS's many built-in components and connectors provide Dataway script-based customization capabilities to dynamically process connector events. For example:

- In the **Set Variable** component, you can use a Dataway script to dynamically set the variable value.
- In the **Transform** component, you can fully utilize the flexible syntax of Dataway for complex data processing and calculation to output the expected result, which can be further processed by downstream components.

You can directly use a Dataway expression in the built-in component configuration of iPaaS.

- To get started with Dataway expressions, see Getting Started.
- If you have iPaaS development experience and want to learn more about Dataway features, see Basic Concepts of Dataway.
  - You can transfer data in text mode.
  - You can process simple data in expression mode.
  - You can process complex data in Python or Java code mode.
- To try out Dataway in use cases, see Use Cases.
- If you have any questions, see Dataway FAQs.

Getting Started

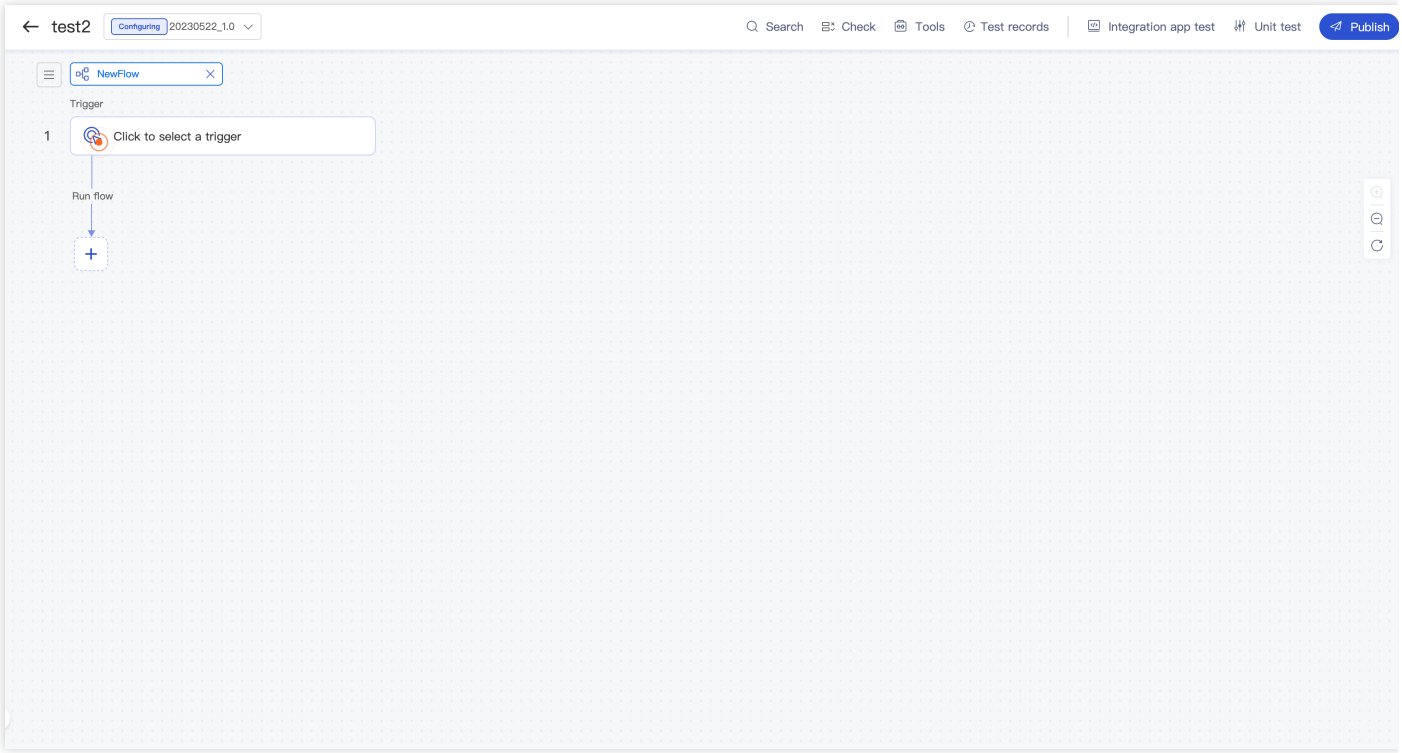
Last updated：2023-08-03 17:51:33

Overview

This document describes how to use a Dataway script to assist with flow design.

Preparations

1. Sign up for a Tencent Cloud account and log in to the iPaaS console.
2. After logging in successfully, create an integration app and a flow.

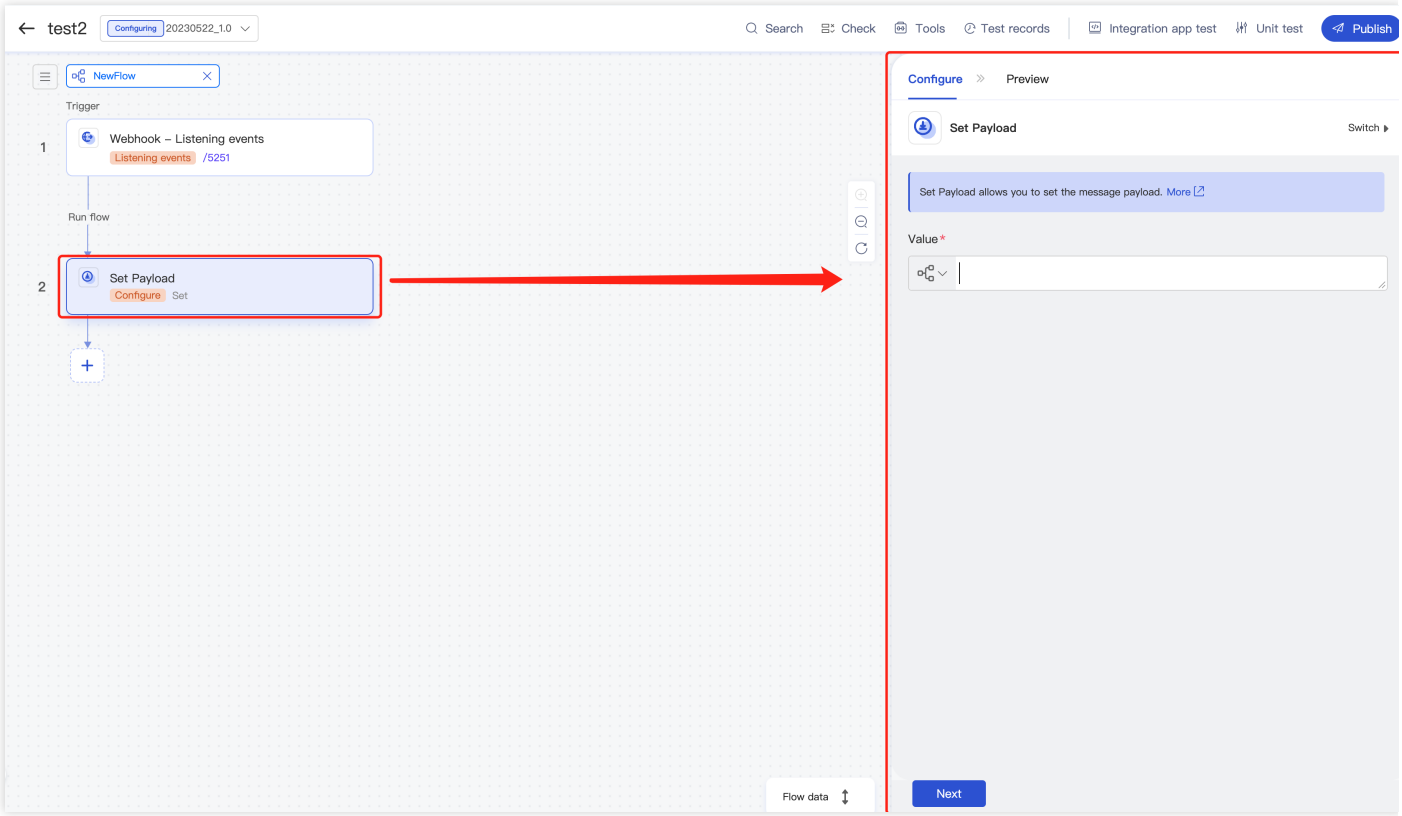


Using a Dataway Expression (Taking a Script in Python Code Mode as an Example)

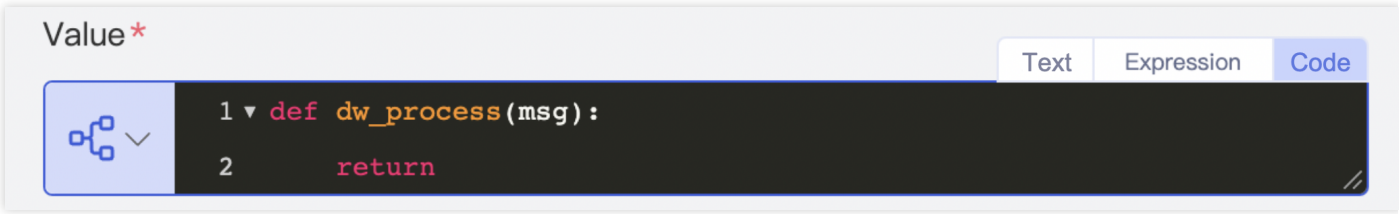
Here, concatenation of a simple string is used as an example:

1. On the **Integration apps** page in the iPaaS console, click **Create**, and create a **Set Payload** component.

2. The component configuration automatically pops up on the right. Here, you need to enter a Dataway expression for **Value**.



3. Hover over the **Value** textbox, and the mode selection buttons will pop up. Click **Code** to enter the code mode.



4. Click the textbox, and the code editor will pop up. Enter a Dataway script, during which the syntax is checked in real time, and errors will be prompted if there are any.

```
def dw_process(msg):
return 'Hello' + 'World'
```

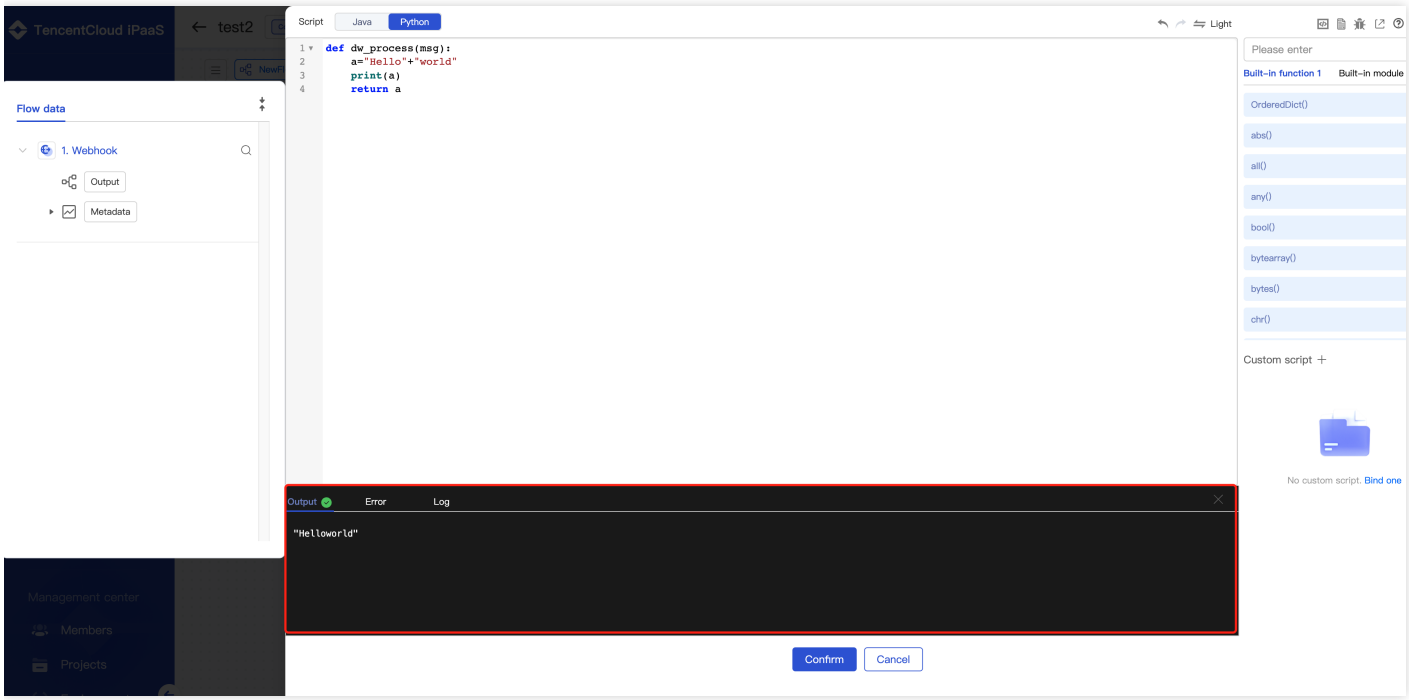
- A complete Python script in Dataway code mode must be a Python 3 code snippet in compliance with the syntax definitions, including the entry function definition `def dw_process(msg)` .

- Dataway is implemented based on Python 3 syntax and has various built-in third-party modules like `time`, `json`, and `math`. To use a module, you can directly reference the module name.

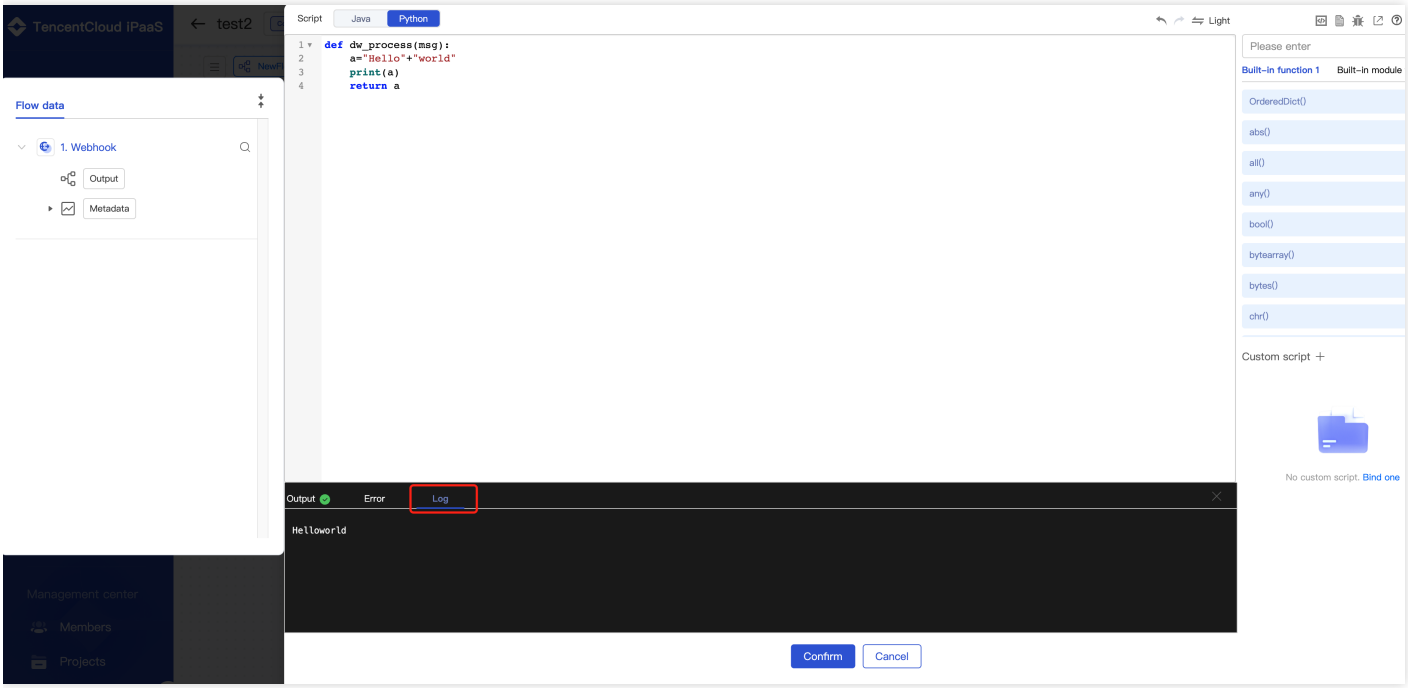
5. Verify the Dataway script execution result: Before the Dataway script passes syntax check and you click **Confirm** to save the expression, you can verify whether the script is correct.

Click **Debug** in the top-right corner of the editing window and click **Start test** in the pop-up window.

After the test is completed, the result will be output at the bottom of the Dataway code editing window. You can see that the Dataway script execution result is `HelloWorld`, which is as expected.



You can switch to the **Log** tab to view the printed output result.



6. Click **Confirm** to save the Dataway script.

Expression mode

You can enter simple expressions in expression mode.

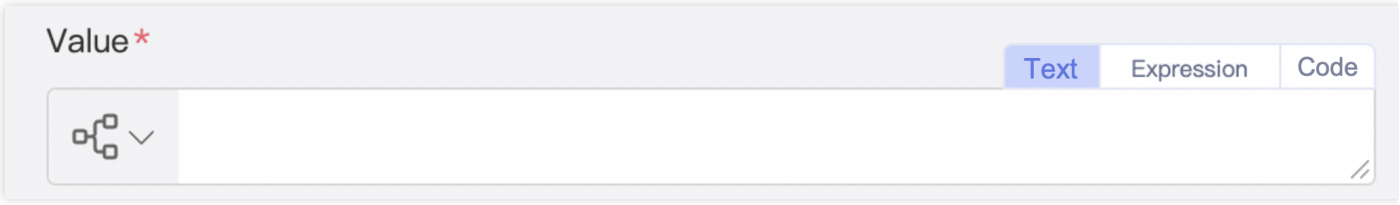
1. Hover over the **Value** textbox, and the mode selection buttons will pop up. Click **Expression** to enter the expression mode.
2. Click the textbox to enter a Dataway expression.

Text mode

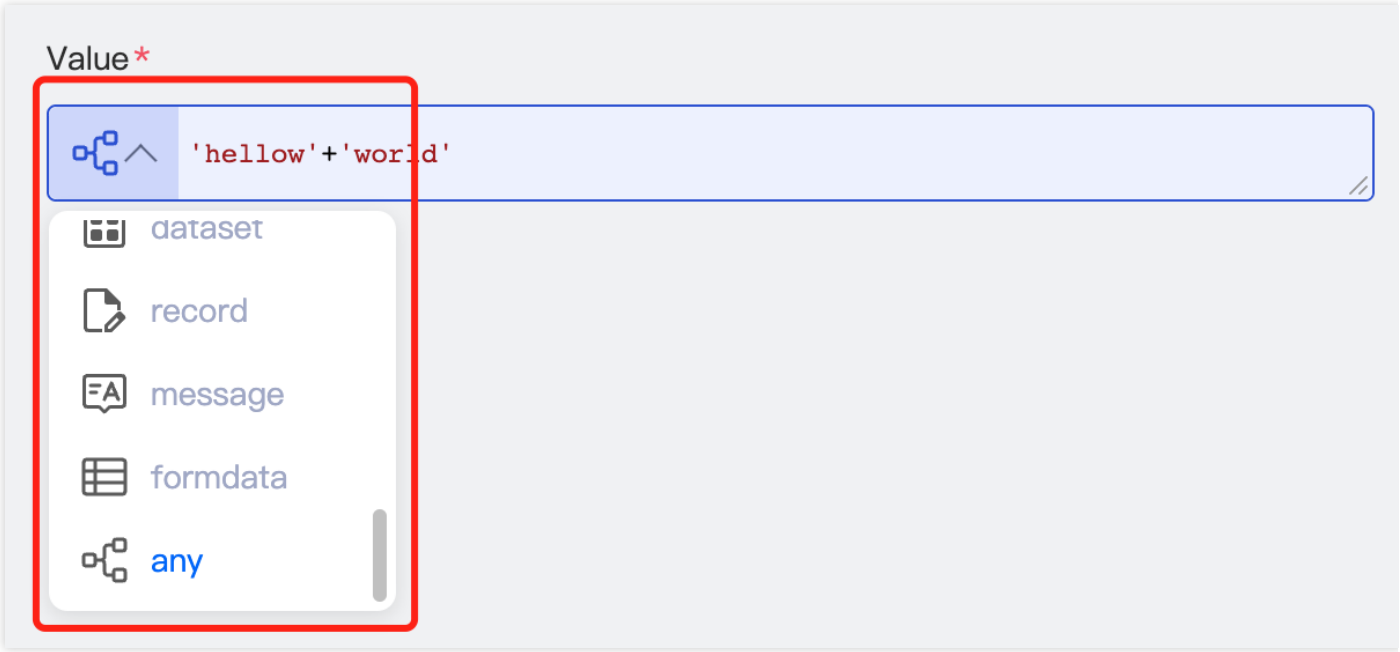
You can input simple data, such as creating literals or referencing flow data, in text mode.

Here, time data generation is used as an example:

1. Hover over the **Value** textbox, and the mode selection buttons will pop up. Click **Text** to enter the text mode.

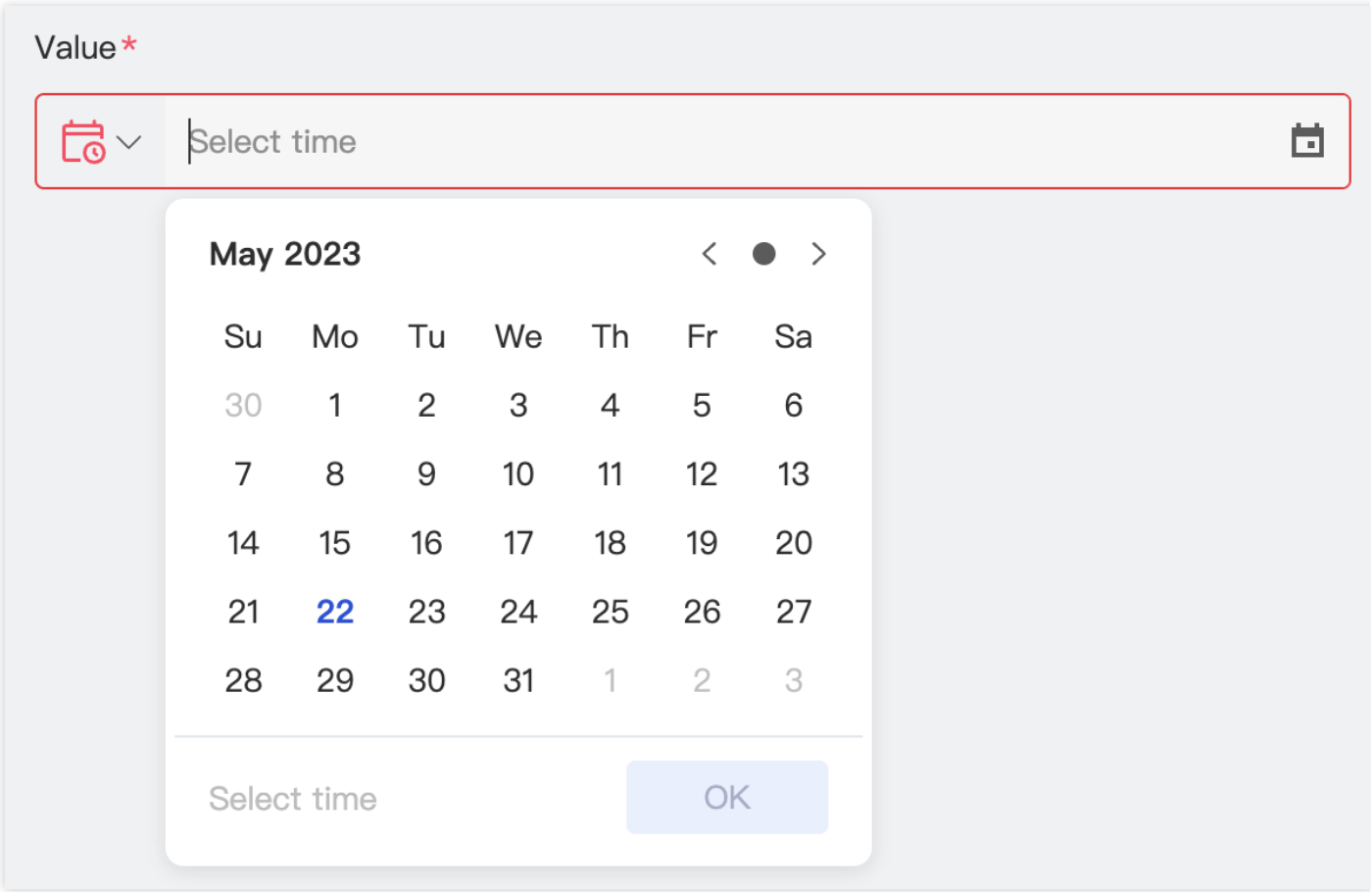


2. Click the data type drop-down list on the left and find and click **datetime**.



3. Click the textbox, and the time configuration interactive UI will pop up. On the UI, configure the time information.

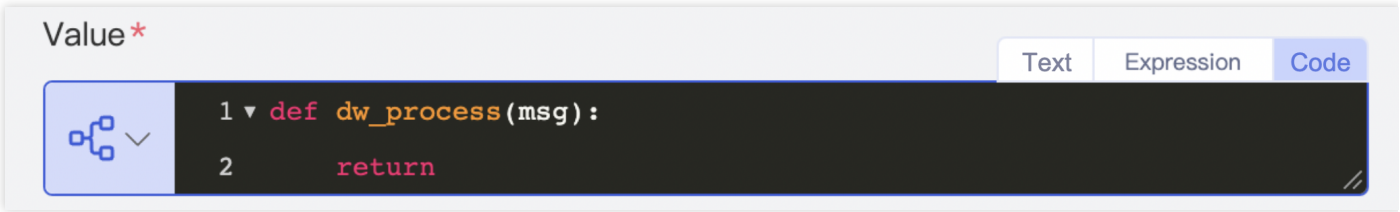
4. Then, click **Confirm**.



Script in Java code mode

In addition to Python syntax, Dataway also supports Java syntax. You can enter Java scripts in code mode.

1. Hover over the **Value** textbox, and the mode selection buttons will pop up. Click **Code** to enter the code mode.



2. Click the textbox to enter the code editing interactive UI. Click **Java** to edit your Java script.

3. Click **Confirm** to save the Java script.

Flow data panel and reference

Dataway allows you to reference the flow context data visually, so that data can flow between components without barriers. Currently, all Dataway modes support visual data reference on the flow data panel, including text, expression, Python code, and Java code modes.

The **Flow data** panel will pop up automatically when you edit the content in the Dataway textbox. You can click a data button on the panel to import the data, and the data will be displayed as tags in the textbox.

Development Guide
Basic Concepts of Dataway

Last updated：2023-08-04 10:10:59

This document describes the core concepts and features of Dataway. Dataway is a scripting engine for custom data conversion and processing in iPaaS. You can use it to write and execute powerful and complex data conversion scripts.

Dataway Toolset

Dataway provides three script toolset modes: text, expression, and code modes, which support distinctive semantics and syntax to meet the data requirements in different use cases. The textboxes of Dataway support all the three toolsets (certain toolsets are disabled for certain components due to the feature design). You can select one to enter the script based on your application needs and habits.

Mode	Purpose	Description
Text mode	Data creation and transfer.	You can generate the required data or reference the context data of a flow as instructed on the GUI.
Expression mode	Simple data conversion and processing as well as lightweight script execution.	You can enter an expression to get the required data.
Code mode	Complex data conversion and processing as well as complex script execution, formatting, and debugging.	You can write a complete Python 3 or Java JDK 8 script to get the required data.

The **flow data panel** embeds all the three modes to reference the context data of a flow. You can click data in the previous components to quickly reference it.

Dataway Type System

Dataway has the following core types, which can be used as the output results of Dataway scripts to be passed between components:

Type	Name	Description	Unique to Dataway	Example
None	Null	Null.	No	Python: None; Java: null
string	String	String.	No	"abc"
bytes	Byte array	Byte array.	No	Python: b"abc"; Java: "abc".getBytes()
bool	Boolean	Boolean.	No	Python: True/False; Java: true/false
float	Float	Float.	No	123.456
int/Long	Integer	Integer.	No	Python: 123; Java: 123L
list	List	Sequence container.	No	Python: [1,2,3]; Java: new java.util.ArrayList<>()
dict/Map	Dictionary	Key-value container.	No	Python: {1:1, 'key': 'value'}; Java: new java.util.HashMap<>()
decimal	Decimal	It is used for accurate decimal value calculation.	No	Python: decimal.Decimal(1); Java: new java.math.BigDecimal("1")
datetime	Date and time	Date and time.	No	Python: datetime.datetime.now(); Java: java.time.OffsetDateTime.now()
date	Date	Date.	No	Python: datetime.date.today(); Java: java.time.LocalDate.now()
time	Time	Time.	No	Python: datetime.datetime.now().time(); Java: java.time.OffsetTime.now()
Entity	Binary entity	Entity data in iPaaS, which represents a binary object, including `blob`, `mime_type`, and `encoding`.	Yes	`payload` in a message constructed by the HTTP Listener component

Type	Name	Description	Unique to Dataway	Example
MultiMap	Multi-value dictionary	Like `xml` but unlike `dict`, this type supports duplicate `key` values.	Yes	Object obtained after data in `application/www-form-urlencoded` format is parsed
FormDataParts	Form data	Array + list data structure, which is similar to `orderDict` in Python.	Yes	Object obtained after data in `multipart/form-data` format is parsed
Message	Message	Message in iPaaS, which carries flow data, including `payload`, `variables`, and `attributes`.	Yes	`msg` parameter in the `dw_process` entry function in a script in Python code mode
DataSet	Dataset	Dataset in iPaaS data integration, which can be manipulated through the data integration component.	Yes	Output of the Builder component
Record	Single data record	Single data record in iPaaS data integration, which contains the schema.	Yes	It can be obtained by using the Foreach component to traverse `DataSet`

Core types in Dataway are available in all the three script toolsets and have the corresponding data structures. Although the operation method of different data structures of the same type varies by script toolset, you can still convert data structures losslessly while ensuring that the core characteristics of different same-type data structures are the same.

If different script toolsets are used upstream and downstream of a flow, different data structures of the same type can be automatically mapped in an imperceptible manner. You can use different Dataway script toolsets in different Dataway textboxes, which will not affect the consistency and accuracy of processing of data of core types.

Note：

If the output result of a Dataway script is the final result returned by the flow, the types supported for the returned value will also be subject to the flow components. If an HTTP Listener component is used as a trigger in the flow, the final returned value must be of the `Entity` type.

In addition to the core types, each script toolset also supports certain unique types. However, **data of the unique types cannot be used as the output result of Dataway scripts**; otherwise, an error will occur. For more information, see Text Mode, Expression Mode, Python Code Mode, and Java Code Mode.

`Message` Type

In Dataway, the `Message` type carries an iPaaS message, which will be passed and updated during flow execution. The `Message` type contains attributes such as `payload` , `vars` , and `attrs` , which are collectively called **predefined attributes**. These attributes are generated by the system based on the current execution information and processed data and used to get the flow context information in a Dataway script.

> Note：
> The input of a Dataway script is the `msg` variable, which is of `Message` type and carries the context information of the flow for execution previous to the current component node.

The attributes in the `Message` type are as detailed below:

Attribute	How to get (Python code mode example)	Description	Type	Remarks
Variable set	msg.vars	Variable set in the context of the current message.	Dictionary type: The key is of the string type and represents the variable name; the value is of any core type and represents the variable value.	The set variables are shared among all subsequent component nodes in the flow. Therefore, this attribute can be used for data integration between different component nodes.

Attribute	How to get (Python code mode example)	Description	Type	Remarks
Payload	msg.payload	Payload data of the current message.	Any core type	`payload` is the payload data in an iPaaS message, which represents the execution result of a component node. It will be updated after a node is executed and can be configured through certain components such as **Set Payload**. For example, the HTTP Listener component will construct the payload content based on the received network request, so the payload will be of `Entity` type after being processed by an HTTP Listener component.
Attribute set	msg.attrs	A set of attributes of the current message, including message source and headers.	Dictionary type: The key is of the string type and represents the attribute name; the value is of any core type and represents the attribute value.	If the flow trigger is HTTP Listener, the network request headers will be stored in `msg.attrs`.
Unique ID	msg.id	The unique ID of the current message.	String type	The unique ID may change after the message passes a logical component.
Sequence number	msg.seq_id	The sequence number of the current message.	String type	The sequence number keeps unchanged when the message is transferred in a flow.

Attribute	How to get (Python code mode example)	Description	Type	Remarks
Error message	msg.error	Error message in the context of the currently processed context.	Dictionary type: The key is of the string type and represents the error attribute name; the value is of the string type and represents the attribute value.	It contains `code` (error type) and `desc` (error description string).

`Entity` Type

Overview

In Dataway, the `Entity` type carries the entity data of iPaaS and is an encapsulation object of binary data. It contains the **raw data (`blob`)**, **MIME type (`mime_type`)**, and **encoding type (`encoding`)**.

- **Raw data (`blob`)**: Raw binary data.
- **MIME type (`mime_type`)**: Content format of binary data, such as `application/json` , `application/www-form-urlencoded` , and `multipart/form-data` .
- **Encoding type (`encoding`)**: String encoding type of binary data, such as `utf8` and `gbk` .

You can access content in `Entity` as follows (taking the Python code mode as an example):

Access Method	Description
entity['^blob']	Gets the payload data of the binary object. A `bytes` object will be returned.
entity['^mime_type']	Gets the MIME type of the message object. A string object will be returned.
entity['^encoding']	Gets the encoding type of the message object. A string object will be returned.

For ease of use, Dataway also provides object methods such as `entity.get(attr, default=None)` and subscript-based selector syntax (for more information, see `Entity` selector) for quick access:

- entity['^value']: Parses the payload data based on the MIME and encoding types and returns the parsing result, which is of a core type.
- entity['xxxx']: Parses the payload data based on the MIME and encoding types and returns the value of the specified key, which is equivalent to `entity\['^value'\]['xxx']` .
- entity.get(attr, default=None): Parses the payload data based on the MIME and encoding types and returns the value of the specified key. If no value can be obtained, the default value (`None` by default) will be returned. This syntax is equivalent to `entity['^value'].get(attr, default=None)` .

When you use the quick access feature, the system will try parsing the binary payload data in `Entity` . If parsing fails, a runtime error will occur. For more information, see Supported MIME Types.

`Entity` selector

For common MIME and encoding types, Dataway allows you to use a selector to quickly access the content in an `Entity` object. The following operation types are supported:

Subscript Type	Description	Example
Number	Used to access the ith element in the current array.	entity[0]
String starting with `^`	Used to get the metadata such as `^mime_type` , `^encoding` , `^blob` (raw binary data), and `^value` .	entity['^mime_type']
Common character (letter, digit, underscore, hyphen, or dot)	A common character key, which is used to get a sub-element of the current element by name. If there are multiple sub-elements with the same name, only the first one will be returned.	entity['list']

Below are examples of the above selector types. Suppose `msg.payload` of the input message is an `Entity` object, and its raw data `blob` is parsed into a JSON array, MIME type is `application/json` , and encoding type is `utf-8` .

```
[{"a1":1},{"b1":1,"b2":2,"b3":3},{"c1":[1,2,3]}]
```

Example 1: Use a number subscript to get the data

You can use a number script to get an element in `msg.payload` . Below is a sample Dataway expression:

```
def dw_process(msg):
return msg.payload[1]
```

The expression output will be of `dict` type: `{"b1":1,"b2":2,"b3":3}` .

Example 2: Use the `^` symbol to get the metadata

You can use the `^` symbol to get the metadata in `msg.payload` . Below is a sample Dataway expression:

```
def dw_process(msg):
return {
"mimeType" : msg.payload["^mime_type"],
"encoding": msg.payload["^encoding"],
"blob": msg.payload["^blob"],
"value": msg.payload["^value"],
}
```

The expression output will be of `dict` type:

```
{
"mimeType" : "application/json",
"encoding": "utf-8",
"blob": b'[{"a1":1},{"b1":1,"b2":2,"b3":3},{"c1":[1,2,3]}]',
"value": [{"a1":1},{"b1":1,"b2":2,"b3":3},{"c1":[1,2,3]}]
}
```

Example 3: Use common characters to get elements

Suppose the value of `msg.payload` is still of `Entity` type, the MIME and encoding types are still `application/json` and `utf-8` respectively, but the payload data `blob` is parsed into the following:

```
{"a1":1,"b1":1,"b2":2,"b3":3,"c1":[1,2,3]}
```

If the following Dataway expression is used:

```
def dw_process(msg):
return {
"a1" : msg.payload["a1"],
"b2": msg.payload["b2"],
"c1": msg.payload['c1'],
}
```

The expression output will be of `dict` type:

```
{
"a1" : 1,
"b2": 2,
"c1": [1,2,3]
}
```

`Entity` object construction (Python code mode example)

1. Value-based constructor (`Entity.from_value`)

This method is used to encapsulate `data` into an `Entity` object and return it as follows:

```
Entity.from_value(data, mime_type=None, encoding="utf-8")
```

`Entity.from_value` serializes `data` based on the specified MIME and encoding types to get the raw data of `bytes` type, encapsulates it into an `Entity` object, and returns the object.

```
Here, the `mime_type` parameter is required. Currently, six MIME types are suppor
ted: `text/plain`, `application/json` (the alias is `text/json`), `application/x-
www-form-urlencoded`, `application/csv`, `application/xml` (the alias is `text/xm
l`), and `multipart/form-data`; the `encoding` parameter can be any valid encodin
g type and will be `utf-8` by default if it is left empty.
```

2. Raw data-based constructor (`Entity.from_bytes`)

This method is used to encapsulate a string or a `bytes` object into an `Entity` object and return it as follows:

```
Entity.from_bytes(data, mime_type=None, encoding="utf-8")
```

The verification rules of the `mime_type` and `encoding` parameters in `Entity.from_bytes` are similar to those in `Entity.from_value` but differ in that the `mime_type` value is not limited and can be any MIME type.

If the `data` parameter passed to the `Entity.from_bytes` method is of `bytes` type, the method will directly return an `Entity` object with the raw data of `data` , MIME type of `mime_type` , and encoding type of `encoding` .
If the passed `data` parameter is a string, it will be encoded as a `bytes` object based on the `encoding` parameter and constructed as an `Entity` object.

Supported MIME Types

Dataway uses the `Entity` type to support various data types, including JSON, CSV, and XML. You can specify the MIME and encoding types in the value or raw data-based constructor to get an `Entity` object encapsulating different data types. You can use an `Entity` selector to read the parsed structure data.

Different MIME types have different data formats in `Entity` . The mappings are as listed below:

MIME Type	Data Format
application/json	JSON format
application/x-www-form-urlencoded	HTTP form format
text/plain	Text format
application/xml	XML format
application/csv	CSV form format
multipart/form-data	HTTP Form-Data form format
Other MIME types	Other formats

The encoding rules, data structures, and specific `Entity` selector syntax vary by data format. Different data formats are as detailed below:

JSON format

JSON data is the serialized data of an `Entity` object with the MIME type of `application/json` .

- If the raw data-based constructor is used, Dataway will parse the input `str` or `bytes` object into a dictionary object.
- If the value-based constructor is used, the input data can be of the list, dictionary, or multi-value dictionary type and will be parsed into a dictionary object.

Below are two examples of how to use JSON data:

- Example 1: Construct an `Entity` object in JSON format
- Example 2: Use a complex JSON structure

This example shows how to construct an `Entity` object in JSON format by using an `Entity` constructor and use an `Entity` selector to get the attributes and data of the `Entity` object.

- **Input**
  The Dataway runtime environment depends on component execution. Suppose a **Transform** component previous

to a **Set Payload** component has set `payload` in the flow execution message `msg` and `msg.payload` is a dictionary object with the following internal structure:

```
{
"name": "zhangsan",
"age": 10,
"male": True,
"brothers": ["lisi", "zhaowu"]
}
```

- **Dataway script**

  The following Dataway script uses the value-based constructor to convert `msg.payload` of dictionary type into an `Entity` object, uses a selector to get the metadata and elements of the object, and returns the obtained content.

```
def dw_process(msg):
entity = Entity.from_value(msg.payload, mime_type='application/json', encoding=
'utf-8')
return {
'blob': entity['^blob'],
'mimeType': entity['^mime_type'],
'name': entity['name'],
'brother': entity['brothers'][0],
'male': entity['^value']['age'],
'other': entity.get('other', 'other_default')
}
```

- **Output**

  The Dataway script output is a dictionary object:

```
{
"blob": b'{"name":"zhangsan","age":10,"male":true,"brothers":["lisi","zhaow
u"]}',
"mimeType": "application/json",
"name": "zhangsan",
"brother": "lisi",
"male": 10,
"other": "other_default"
}
```

HTTP form format

HTTP form data is the parsed data of an `Entity` object with the MIME type of `application/x-www-form-urlencoded`.

- If the raw data-based constructor is used, Dataway will parse the input string or `bytes` object into a dictionary object.
- If the value-based constructor is used, the input data can be of the dictionary or multi-value dictionary type and will be parsed into a multi-value dictionary object.

As its internal implementation is of the multi-value dictionary type, HTTP form data also supports special selector syntax in addition to general `Entity` selector syntax.

Selector	Description
['*key']	Returns all elements of `key` in the dictionary.
['key']	Returns the first element of `key` in the dictionary.

Example: Construct and use an `Entity` object in HTTP form format

This example shows how to construct an `Entity` object in HTTP form format by using an `Entity` constructor and use `Entity` selector syntax to get the attributes and data of the `Entity` object.

- **Input**

  `msg.payload` stores the HTTP form data `k1=123&k2=helloworld&k3=2&k3=abc`, which is represented as a multi-value dictionary object in Dataway and can be converted into a dictionary object.

```
{
"k1": 123,
"k2": "helloworld",
"k3": [2, "abc"]
}
```

- **Dataway script**

  The following Dataway script uses the value-based constructor to convert a dictionary object into an `Entity` object and uses a selector to get the metadata and elements of the object.

```
def dw_process(msg):
entity = Entity.from_value(msg.payload, mime_type='application/x-www-form-urlencoded', encoding='utf-8')
return {
'blob': entity['^blob'],
'mimeType': entity['^mime_type'],
'k1': entity['k1'],
```

```
'k3': entity['^value']['k3'],
'k3multi_selector': entity['^value']['*k3'],
'k5': entity.get('k5', 'default_value')
}
```

- **Output**

  The Dataway script output is a dictionary object:

```
{
"blob": b'k1=123&k2=helloworld&k3=2&k3=abc',
"mimeType": "application/x-www-form-urlencoded",
"k1": 123,
"k3": 2,
"k3multi_selector": [2, "abc"],
"k5": "default_value"
}
```

Text format

Text data is the parsed data of an `Entity` object with the MIME type of `text/plain`. In both the value-based constructor and raw data-based constructor, the `data` parameter is a string or a `bytes` object, and the `Entity` object is a string.

Example: Construct an `Entity` object in text format

This example shows how to construct an `Entity` object in text format by using an `Entity` constructor and use `Entity` selector syntax to get the attributes and data of the `Entity` object.

- **Input**

  `msg.payload` stores text data: "This is a text plain message".

- **Dataway script**

  The following Dataway script uses the value-based constructor to convert a string into an `Entity` object and uses a selector to get the metadata and elements of the object.

```
def dw_process(msg):
entity = Entity.from_value(msg.payload, mime_type='text/plain', encoding='utf-8')
return entity['^value']
```

- **Output**
  The Dataway script output is a string: "This is a text plain message".

XML format

XML data is the serialized data of an `Entity` object with the MIME type of `application/xml`.

- If the raw data-based constructor is used, Dataway will parse the input string or `bytes` object into a dictionary object.
- If the value-based constructor is used, the input data can be of only the dictionary type and will be parsed into a dictionary object. The input dictionary only contains a default key `root`, and the value is the built-in `MultiMap`, where you can manipulate the `msg` attributes as needed.

XML data also supports special selector syntax in addition to general `Entity` selector syntax.

Selector	Description
['*key']	Returns all elements of `key` on an XML node.
['*key']	Returns the first element of `key` on an XML node.
['#text']	Returns the text value of an XML node.
['@attr']	Returns the `attr` value of an XML node.

Below are two examples of how to use XML data:

Example 1: Construct an `Entity` object in XML format

This example shows how to construct an `Entity` object in XML format by using an `Entity` constructor and use `Entity` selector syntax to get the attributes and data of the `Entity` object.

- **Input**
  The value of `payload` in the Dataway input parameter `msg` is constant `1`, and `msg.vars` contains a key-value pair with the key of `abc` and the value of `123`.

```
{
"payload": 1,
"vars": {
"abc": "123"
}
}
```

- **Dataway script**

  The following Dataway script uses the value-based method to convert a dictionary object into an `Entity` object and returns it:

```
def dw_process(msg):
a = math.floor(1.4)
return Entity.from_value({
'root': {
'k1': msg.vars['abc'],
'k2': json.dumps('Haha', ensure_ascii=False),
'k3': a + 1,
'@id': "hello",
'#text': "<a>dwad</a>",
'k4': ['abc', 'def', None],
}
}, mime_type = 'application/xml')
```

- **Output**

  The output result of the Dataway script is an `Entity` object. Here, the raw data (`blob`) is a binary object in XML format:

```
{
"mime_type": "application/xml",
"encoding": "utf-8",
"blob": b'<?xml version="1.0" encoding="utf-8"?>
<root id="hello"><k1>123</k1><k2>"Haha"</k2><k3>2</k3><k4>abc</k4><k4>def</k4><
k4></k4><a>dwad</a></root>'
}
```

Example 2: Use a specific XML selector

This example shows how to use a selector of the specific syntax in XML data.

- **Input**

  The value of `payload` in the Dataway input parameter `msg` is constant `1` , and `msg.vars` contains a key-value pair with the key of `abc` and the value of `123` .

```
{
"payload": 1,
"vars": {
"abc": "123"
}
}
```

- **Dataway script**

  The following Dataway script uses the value-based constructor to convert a dictionary object into an `Entity` object and uses a selector to get the object data.

```
def dw_process(msg):
a = math.floor(1.4)
entity = Entity.from_value({
'root': {
'k1': msg.vars['abc'],
'k2': json.dumps('Haha', ensure_ascii=False),
'k3': a + 1,
'@id': "hello",
'#text': "<a>dwad</a>",
'k4': ['abc', 'def', None],
}
}, mime_type = 'application/xml')
return {
'k1': entity['root']['#text'] + entity['root']['@id'],
'k2': entity['root']['k1'],
'k3': entity['root']['*k4']['^value'],
'k4': entity['root']['k4'],
'k5': entity['^mime_type']
}
```

- **Output**

  The Dataway script output is a dictionary object:

```
{
"k1": "<a>dwad</a>hello",
"k2": "123",
"k3": ['abc', 'def', None],
"k4": "abc",
"k5": "application/xml"
}
```

Note：
In XML data, the `root` node is the default node, and its attributes are specified in the format of `@id=123` and text in the format of `#text`. The `root` value is of `MultiMap` type, where the key is the name of each child node and the value is the child node value.

CSV format

CSV data is the serialized data of an `Entity` object with the MIME type of `application/csv`.

- If the raw data-based constructor is used, Dataway will parse the input string or `bytes` object into a list data structure, where each element is a dictionary object.
- If the value-based constructor is used, the input data can be of the list type and will be parsed into a list data structure, where each element is a dictionary object.

Below is an example of how to use CSV data:

Example: Construct an `Entity` object in CSV format

This example shows how to construct an `Entity` object in CSV format by using an `Entity` constructor and use `Entity` selector syntax to get the attributes and data of the `Entity` object.

- **Input**
  The value of `payload` in the Dataway input parameter `msg` is constant `1`, and `msg.vars` contains a key-value pair with the key of `abc` and the value of `123`.

```
{
"payload": 1,
"vars": {
"abc": "123"
}
}
```

- **Dataway script**
  The following Dataway script uses the value-based constructor to convert a dictionary object into an `Entity` object and uses selector syntax to get the object attribute values.

```
def dw_process(msg):
entity = Entity.from_value([
{'k1':'abcd','k2':123.0,'k3':True},
{'k1':'defs','k2':'dwdw,2e','k3':10},
], mime_type = 'application/csv')
return {
'var1': entity['^blob'],
'var2': entity['^mime_type'],
'var3': entity['^encoding'],
'var4': entity[0]['k2'] + entity[1]['k3'],
'var5': entity[1]['k2']
}
```

- **Output**

  The Dataway script output is a dictionary object:

```
{
"var1": b'k1,k2,k3\r\nabcd,123.0,True\r\ndefs,"dwdw,2e",10\r\n',
"var2": "application/csv",
"var3": "utf-8",
"var4": "133",
"var5": "dwdw,2e"
}
```

Note：
For CSV data format, each element in the received list is a dictionary object. As the title line of the CSV text, the key in each element must be the same; the value of each element represents the value of the line, and multiple values are separated by comma.

HTTP Form-Data form

HTTP Form-Data form data is the serialized data of an `Entity` object with the `mime-type` of `multipart/form-data` .

`multipart/form-data` in the browser

When a browser sends a request with the `Content-Type` of `multipart/form-data` , the actually transferred byte array is converted into a string as follows:

Each parameter starts with a boundary indicating the start of the parameter, such as `--34b21` . Note that `--` is the fixed start, and `34b21` is a random string of up to 70 characters generated by the browser.

The next two lines are the fixed headers of `Content-Disposition` and `Content-Type` . `Content-Disposition` contains two fields: `name` and `filename` . `Content-Type` is the `Content-Type` of the input content. `name` is the parameter name, and `filename` is the filename. If `filename` is `*.txt` or empty, `Content-Type` will be `text/plain` by default. If `filename` is another value, `Content-Type` will be set automatically based on the file extension. In addition, other extended headers are also supported.

The next line is the actual content. If `Content-Type` is `text/plain` , it will be a general string such as `Book` . If `Content-Type` is `application/json` , it will be a JSON string, such as the JSON structure of `file1` .

On the last line, `--34b21--` is used to mark the end of the request.

```
--34b21
Content-Disposition: form-data; name="text"
Content-Type: text/plain

Book
--34b21
Content-Disposition: form-data; name="file1"; filename="a.json"
Content-Type: application/json

{
"title": "Java 8 in Action",
"author": "Mario Fusco",
"year": 2014
}
--34b21
Content-Disposition: form-data; name="file2"; filename="a.html"
Content-Type: text/html

<!DOCTYPE html>
<title>
Available for download!
</title>
--34b21--
```

Form-Data construction and usage

- If the raw data-based constructor is used, Dataway will parse the input string or `bytes` object into a FormDataParts data structure.
- If the value-based constructor is used, the input data can be of `Entity`, `Formdata`, list, or dictionary type and will be parsed into a FormDataParts data structure.

Note：

- If an `Entity` object is input, the system first checks whether its MIME type is `multipart/form-data`, and if so, the `Entity` object will be directly returned; otherwise, an error will be reported.
- If a `FormdataParts` object is input (the list/dictionary structure mentioned above), its value will be directly assigned to an `Entity` object.
- If a list object is input, the data structure must be as follows: The first element is the parameter name; if the second element is a list, the first, second, third, and fourth items in the second element must be a filename, the actual content, `Content-Type`, and a dictionary (representing `extra_headers`) respectively. If the second element is a string, the filename is empty, the actual content is the string content, and `Content-Type` is `text/plain` by default.

HTTP Form-Data data also supports special selector syntax in addition to general `Entity` selector syntax.

Selector	Description
['parts']	Returns data of the custom `FormDataParts` type.
['parts'][0]	Returns the zeroth item of `FormData` .
['parts']['a']['content']	Returns the `content` value in the value with the key of `a` in `FormDataParts` .
['boundary']	Returns the separator of `FormDataParts` .

Examples

Below are two examples of how to use HTTP Form-Data:

- Example 1: Construct an `Entity` object in HTTP Form-Data format
- Example 2: Use a Form-Data selector

This example shows how to construct an `Entity` object in HTTP Form-Data format by using an `Entity` constructor.

- **Input**
  The variables (`vars`) in the Dataway input parameter `msg` contain a key-value pair with the key of `abc` and value of `123` .

```
{
"vars": {
"abc": "123"
}
}
```

- **Dataway script**
  The following Dataway script uses the value-based constructor to convert a dictionary object into an `Entity` object and returns it:

```
def dw_process(msg):
a = math.floor(1.4)
c = Entity.from_value({
```

```
'k1': msg.vars['abc'],
'k2': json.dumps('Haha', ensure_ascii=False),
'k3': a + 1,
}, mime_type = 'application/json')
return Entity.from_value(
[
('a', ('test.json', '{"a": a}', 'application/json', {'Test111': 1})),
('b', '333'),
('c', ('c.json', c, c['^mime_type']))
],
mime_type='multipart/form-data; boundary=123333333'
)
```

- **Output**

  The output result of the Dataway script is an `Entity` object. Here, the raw data (`blob`) is a binary object with the `multipart/form-data` structure:

```
{
"mime_type": "multipart/form-data; boundary=12345",
"encoding": "utf-8",
"blob": b'''--123333333
Content-Disposition: form-data; name="a"; filename="test.json"
Content-Type: application/json
Test111: 1

{"a": a}
--123333333
Content-Disposition: form-data; name="b"


333
--123333333
Content-Disposition: form-data; name="c"; filename="c.json"
Content-Type: application/json


{"k1":"123","k2":"&amp;quot;Haha&amp;quot;","k3":2}
--123333333--
'''
}
```

Other types

For data of other MIME types, Dataway cannot directly create it through the value-based constructor but can read it from the upstream components or use the raw data-based constructor to construct an encapsulated `Entity` object.

Suppose the input data is a binary `bytes` flow and a Dataway expression is used in the **Set Payload** component to encapsulates the `bytes` flow into `msg.payload` :

Dataway expression

```
def dw_process(msg):
b = msg.payload
return Entity.from_bytes(b, mime_type='application/octet-stream')
```

You can use `Entity` selector syntax for subsequent operations in downstream components.

Flow Data Panel

Dataway supports visual data reference. On the **Flow data** panel, you can click a data tag to reference the target data in the flow context, such as variable and previous component output. This lets you interconnect components more quickly and easily. Currently, all Dataway modes support visual data reference, including text, expression, Python code, and Java code modes.

1. The **Flow data** panel will pop up automatically when you click the Dataway textbox.
2. On the **Flow data** panel, click a data tag to select it, and the Dataway textbox will directly reference the flow context data and insert the data tag where the cursor is.

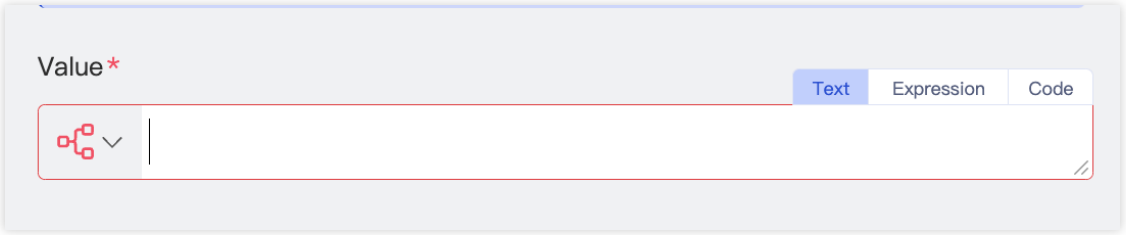
Text Mode

Last updated：2024-12-24 09:35:16

In text mode, you can generate the required data through simple operations.

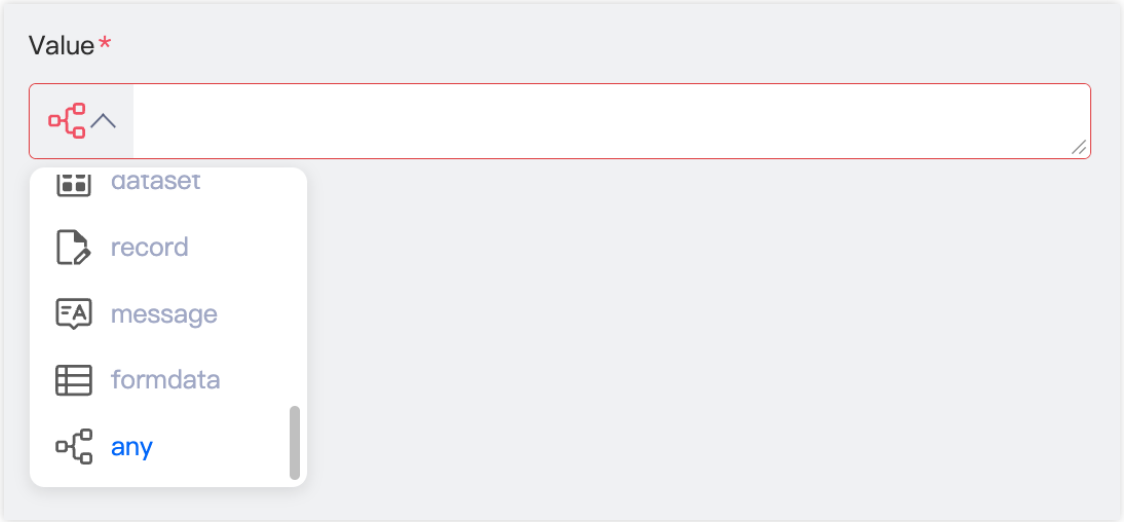
Use of IDE

1. Hover over any Dataway textbox, and the mode selection buttons will pop up automatically. Click **Text** to enter the text mode.



>

2. Click the data type drop-down list on the left and select the target data type, and Dataway will display a dedicated input interactive UI.



Data types

Show All

Any type (`any` , which is the default type)

展开&收起

You can directly enter text to generate a string or reference the flow context data on the flow data panel. If there are multiple items of text data or data referenced on the flow data panel, such data items will be used as elements to form a list.

Integer (`int`), float (`float`), string (`string`), decimal (`decimal`), and Boolean (`bool`)

展开&收起

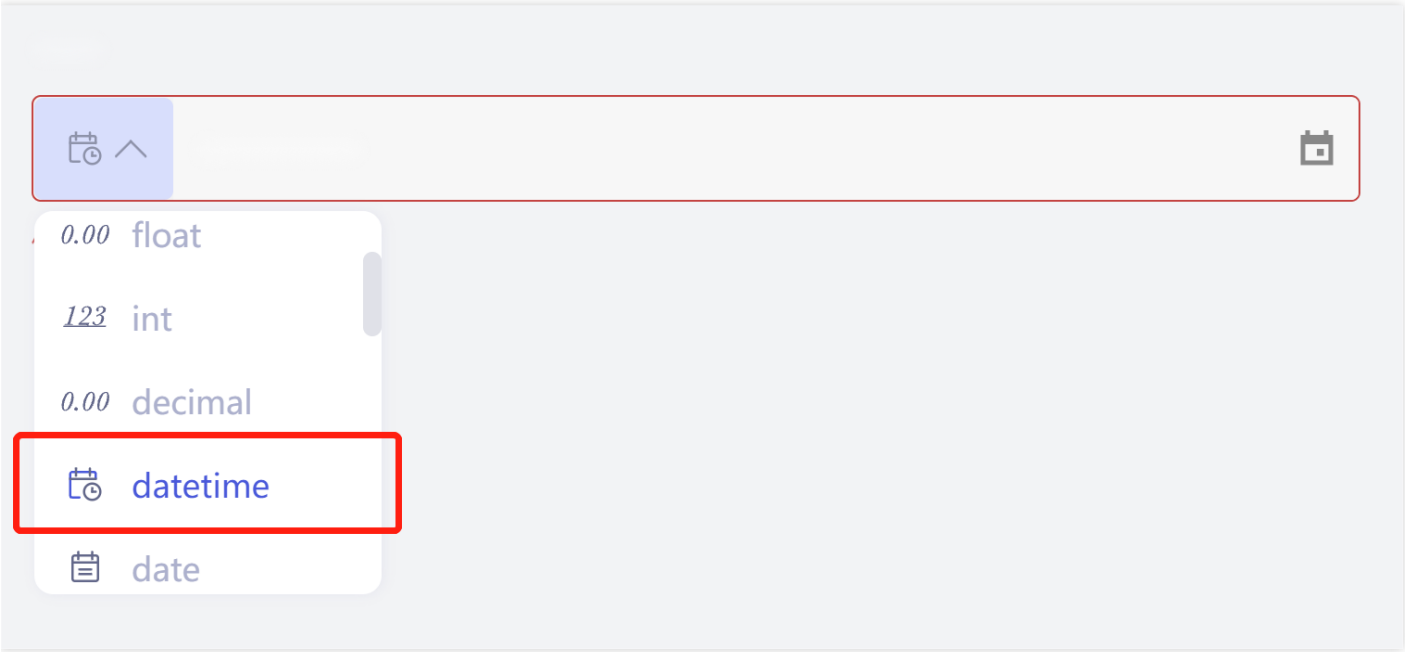
You can directly enter a literal to generate the data or reference the flow context data on the flow data panel. The data referenced on the flow data panel will be converted into a string and added to the literal.

Date and time (`datetime`), date (`date`), and time (`time`)

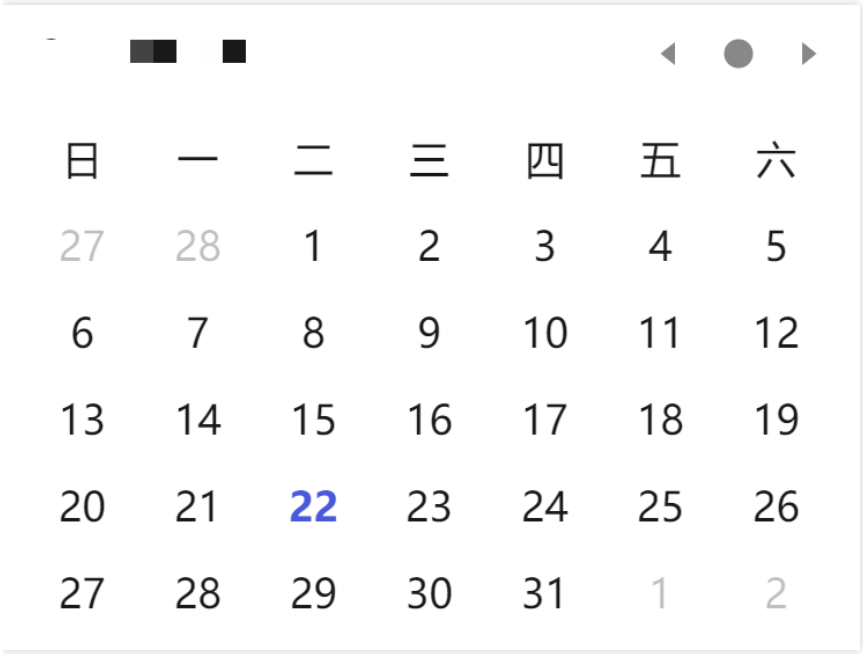
展开&收起

Click the textbox, and the corresponding visual interactive UI will be displayed. You can click the target time data to select it on the UI.

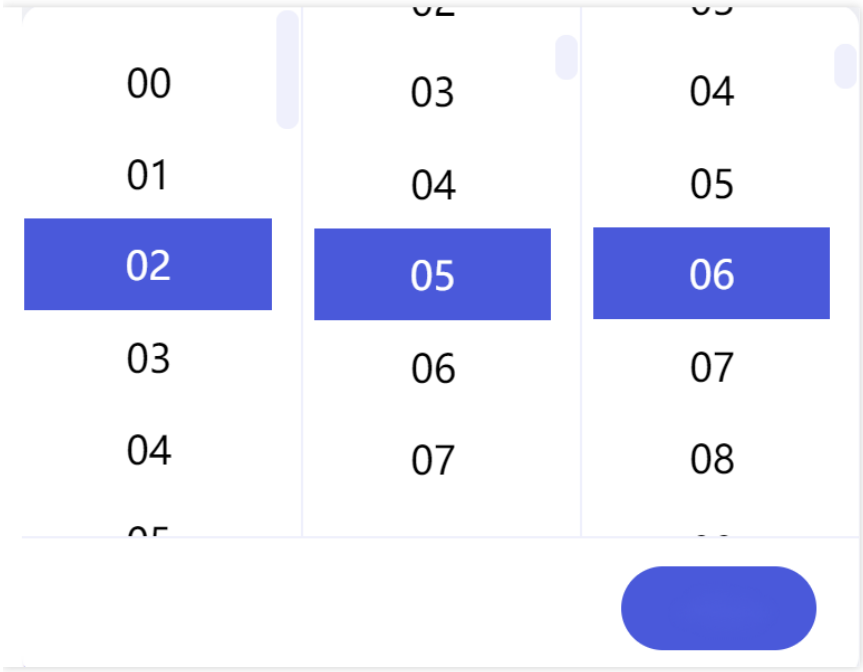
Date and time:



Date:



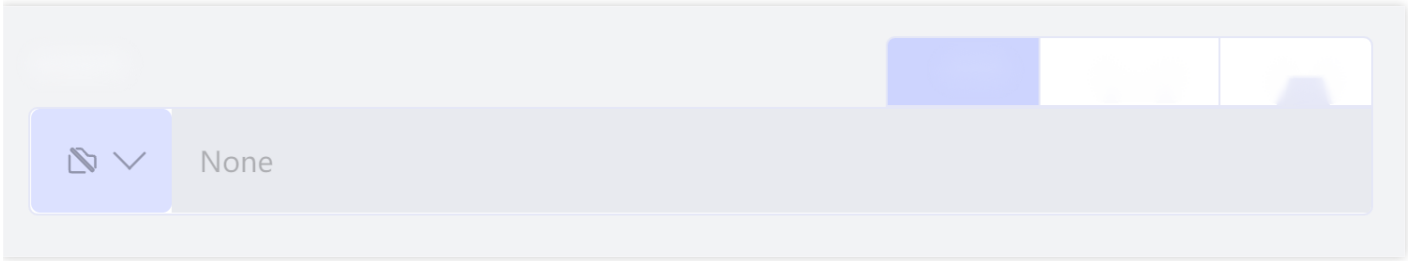
Time:



Null (`None`)

展开&收起

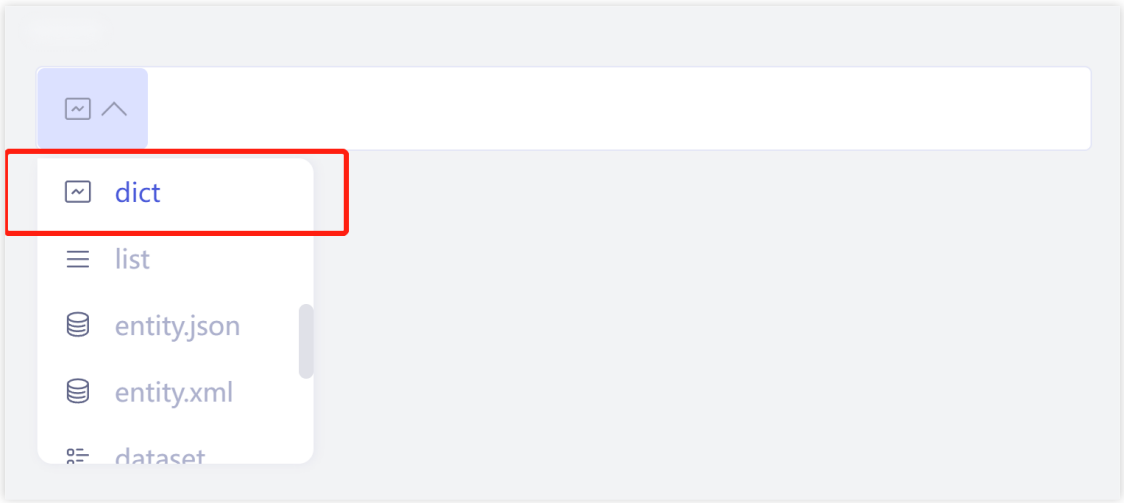
This type indicates that the input data is null, for which the textbox is grayed out.



Dictionary (`dict`), list (`list`), message (`Message`), and form (`FormData`)

展开&收起

1. Click the textbox, and the corresponding interactive UI will be displayed.

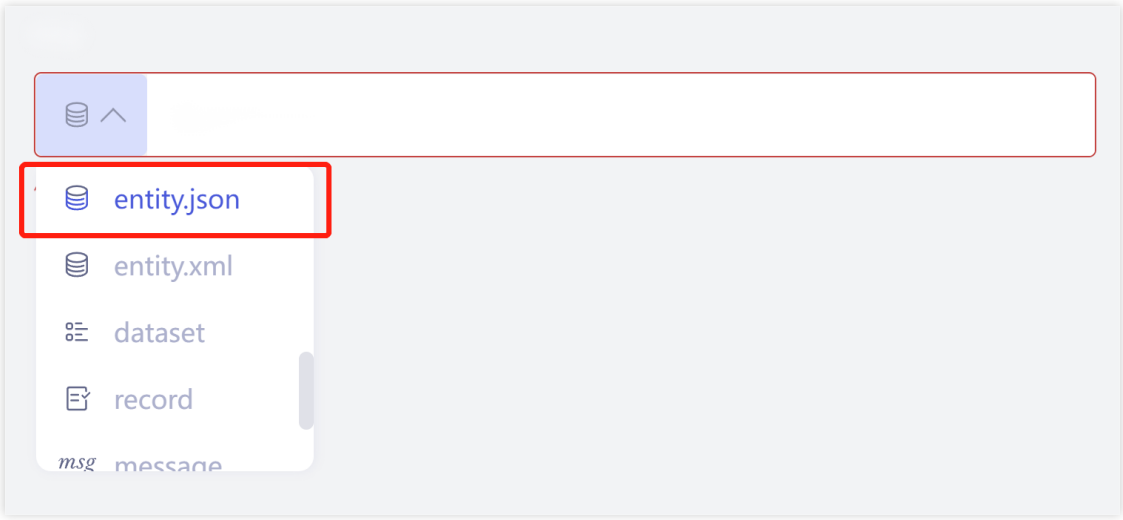


2. On the interactive UI, add data items one by one and confirm the content.

Binary JSON (`Entity.json`) and binary XML (`Entity.xml`)

展开&收起

1. Click the textbox, and the corresponding text input UI will be displayed.



2. On the text input UI, enter the text and click **Confirm**, and a UTF-8 encoded `Entity` object of the specified MIME type (`json` or `xml`) will be generated automatically.

Data set (`DataSet`) and single data record (`Record`)

展开&收起

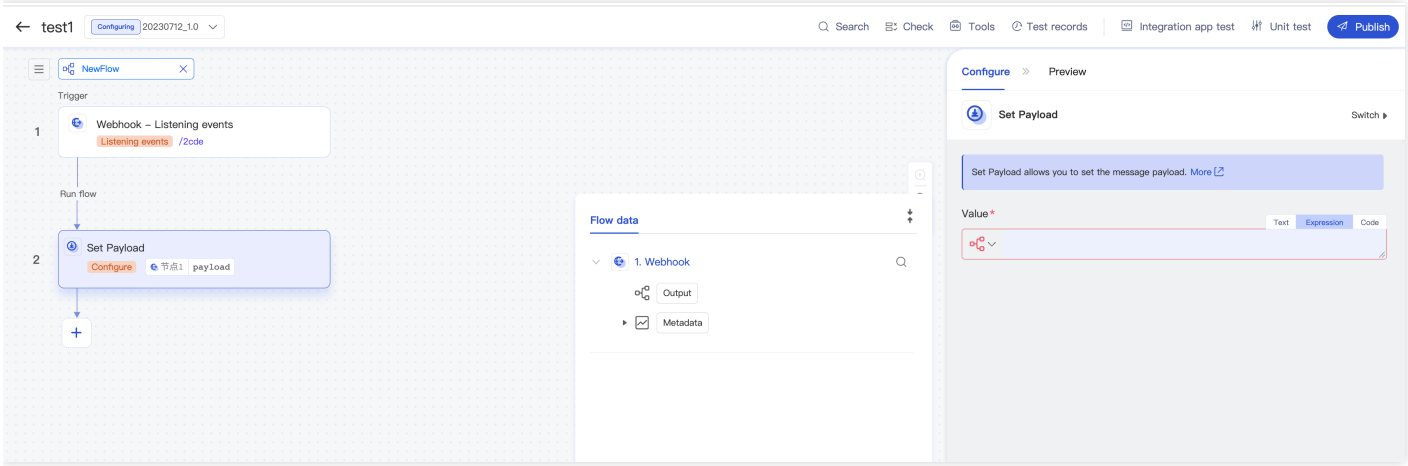
In text mode, you can reference data integration data in the flow context on the flow data panel: data set (`DataSet`) and single data record (`Record`). As `DataSet` and `Record` are data types unique to data integration, you need to generate data of such types through certain components such as **RecordSet Encoder**, and Dataway doesn't provide any generation method.

Expression Mode

Last updated：2023-08-03 17:51:33

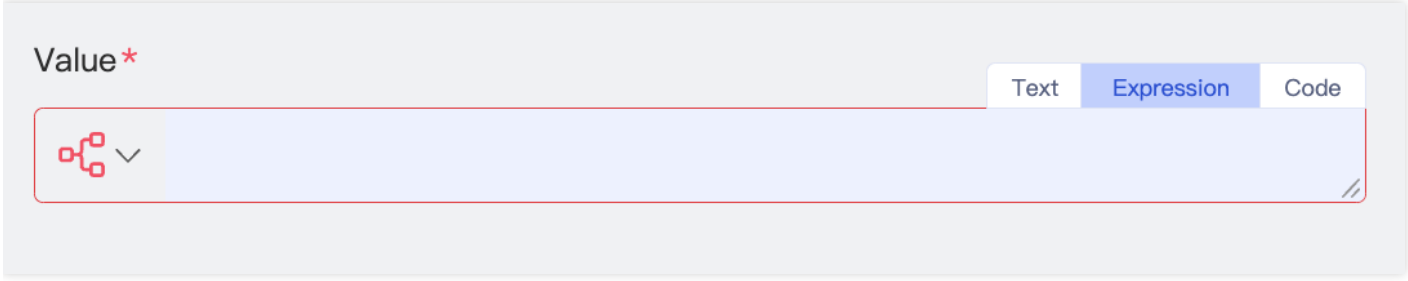
Overview

In expression mode, you can enter a valid Python expression to return the required data. The expression mode is specifically optimized to provide a smart prompt feature which simplifies your input and outperform the code mode.



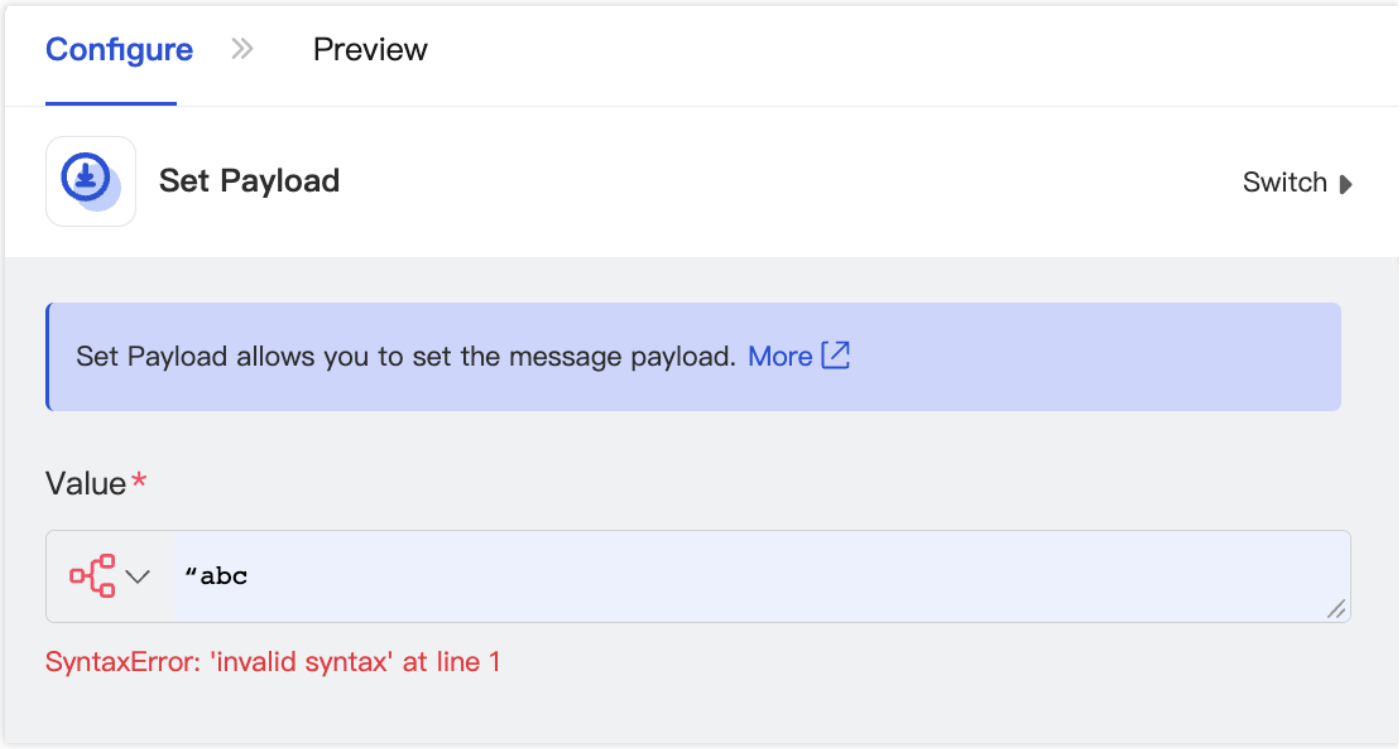
Use of IDE

Hover over any Dataway textbox, and the mode selection buttons will pop up automatically. Click **Expression** and then click the textbox to enter an expression.



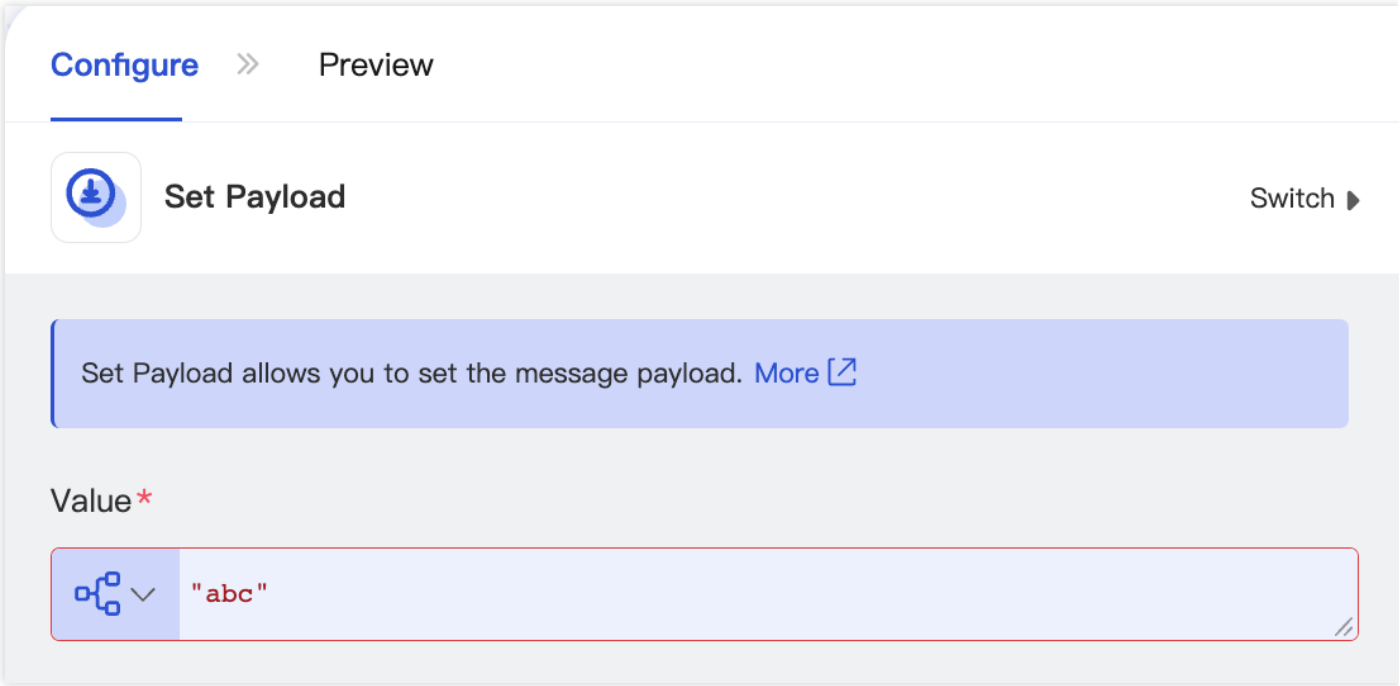
- **Syntax check**
  On the Dataway interactive UI, you can check the syntax of the expression in real time. If an error occurs, the border of the textbox will turn red, and an error message will be displayed below the textbox.
  You can modify the expression based on the error message. A flow can be published only after all its expressions

pass syntax check.



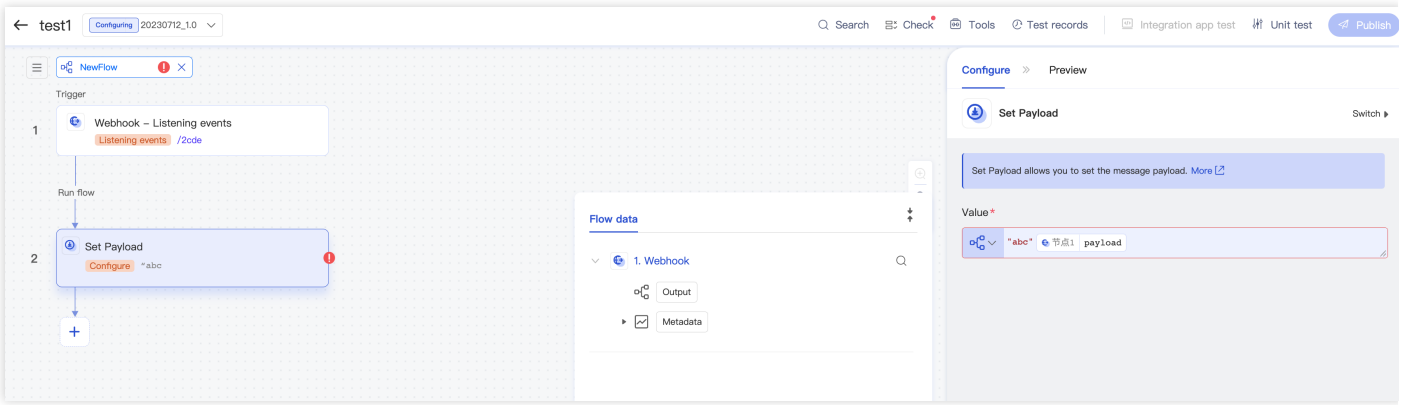
- **Autocomplete**
  When you input content in the textbox, the Dataway interactive UI will automatically display the syntax prompts and viable completion options based on the current context below or above the textbox. You can select an appropriate tag to quickly complete your expression. The syntax prompts include attributes, methods, built-in functions, and

third-party modules.



- **Reference on the flow data panel**

  In expression mode, you can reference data on the flow data panel. For more information, see Flow Data Panel.



- **Type conversion**

  Other than certain components with special requirements, the expression mode supports convenient type conversion for the Dataway interactive UI. You can click the target data type in the drop-down list on the left of the textbox to convert the type of the expression output result explicitly, so as to forcibly change the data type. The

default type is `any`, i.e., no type conversion.

Syntax

The expression mode is based on the `eval()` function in Python 3 syntax to make it easier to use. Like the code mode, the expression mode allows you to use `msg` (`Message` type) to reference the message of the current flow. In addition, in expression mode, you can quickly reference the output data of previous components quickly on the flow data panel.

The expression mode supports the following syntax structures:

Expression Group	Expression Type	Description	Example	Remarks
Atomic expression	literal	Literal	"abc", 123, True, b'abc'	Literal of a data type such as `string`, `int`, `float`, `bool`, or `bytes`.
	name	Identifier	abc	A variable used to read the specified name from global context.
	tuple	Tuple	('a', 'b', 'c'), (str(1))	Returns a tuple if there is at least one comma; otherwise, returns the value of a single expression.
	list	List	[1,2,3]	Enumerates elements to construct a

				list.
	list-comp	List comprehension	[i for i in 'abc']	Constructs a list through comprehension.
	set	Set	{1,2,'a'}	Enumerates elements to construct a set.
	set-comp	Set comprehension	{i for in 'abc'}	Constructs a set through comprehension.
	dict	Dictionary	{1:2, 'a':'b', 3.0:True}	Enumerates elements to construct a dictionary.
	dict-comp	Dictionary comprehension	{i:i*i for i in range(10)}	Constructs a dictionary through comprehension.
	generator	Generator	(k*k for k in range(10))	Returns a generator.
Primary expression	attr	Attribute reference	msg.payload	Returns an attribute.
	index	Container subscript value	msg.payload[1], msg.vars['a']	Gets the value by subscript.
	slice	Slice subscript	msg.payload[1:3]	Gets the value by subscript.
	call	Call	str('a')	Calls a function.
Mathematical expression	binop	Binary operator	3**3, 3+3, 'a' is not in msg.vars	Exponentiation (**), arithmetic operations (+, -, *, /, //, %), shift operations (>>, <<), bitwise operations (&, ^, |), and comparison operations (>, >=, <, <=, ==, !=, is, is not, in, not in) are supported.
	uniop	Unary operator	not msg.vars, ~msg.vars['no']	`+` (keeps the value of the operand), `-` (produces the negative value), `~` (indicates bitwise negation), and `not` (indicates logical NOT).
Conditional expression	if-expr	Conditional expression	'a' if 's' in msg.vars else 'b'	xx if True else xx.
	logical	Logical expression	a and b, not True	Boolean operations (`and`, `not`, `or`) are supported.

Special expression	dataref	Data reference	Tags generated automatically after you click data in the drop-down list	References the data in the specified path from the context.

Data types

The expression mode fully supports the core types in iPaaS based on native types in Python. Data of types bound to the core types in iPaaS can freely flow between components and Dataway.

Data Type	Core Type in iPaaS	Feature	Feature Type	Feature Functionality	Outpu
int	Integer	+	Operator	Addition.	int
		-	Operator	Subtraction.	int
		*	Operator	Multiplication.	int
		/	Operator	Division.	float
		//	Operator	Floor division.	int
		%	Operator	Modulus.	int
		-x	Operator	Negative value.	int
		&	Operator	Bitwise AND.	int
		|	Operator	Bitwise OR.	int
		^	Operator	Bitwise XOR.	int
		~	Operator	Bitwise negation.	int
		<<	Operator	Left shift.	int
		>>	Operator	Right shift.	int
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool

		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
str	String	+	Operator	Concatenation.	str
		*	Operator	Repetition.	str
		[index]	Subscript operation	Gets the value of the specified index.	str
		[index1:index2]	Subscript operation	Splices the data.	str
		[index1:index2:step]	Subscript operation	Splices the data by step.	str
		in	Operator	Returns whether the data is a substring.	bool
		%	Operator	Formatting.	str
		==	Operator	Equal to	bool
		!=	Operator	Not equal to	bool
bool	Boolean	or	Operator	OR	bool
		and	Operator	AND	bool
		not	Operator	NOT	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
float	Float	+	Operator	Addition.	float
		-	Operator	Subtraction.	float

		*	Operator	Multiplication.	float
		/	Operator	Division.	float
		//	Operator	Floor division.	float
		%	Operator	Modulus.	float
		-x	Operator	Negative value.	float
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool
		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
bytes	Non-core types	+	Operator	Concatenation.	bytes
		*	Operator	Repetition.	bytes
		[index]	Subscript operation	Gets the value of the specified index.	int
		[index1:index2]	Subscript operation	Splices the data.	bytes
		[index1:index2:step]	Subscript operation	Splices the data by step.	bytes
		in	Operator	Returns whether the data is a substring.	bool
		%	Operator	Formatting.	bytes
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool

list	List	+	Operator	Concatenation.	list
		*	Operator	Repetition.	list
		[index]	Subscript operation	Gets the value of the specified index.	any
		[index1:index2]	Subscript operation	Splices the data.	list
		[index1:index2:step]	Subscript operation	Splices the data by step.	list
		in	Operator	Returns whether the data is an element in the list.	bool
		<	Operator	Returns whether each element in the former list is less than the corresponding element in the latter list.	bool
		>	Operator	Returns whether each element in the former list is greater than the corresponding element in the latter list.	bool
		<=	Operator	Returns whether each element in the former list is less than or equal to the corresponding	bool

				element in the latter list.	
		>=	Operator	Returns whether each element in the former list is greater than or equal to the corresponding element in the latter list.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
dict	Dictionary	[key]	Subscript operation	Gets the value of the specified `key`.	any
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
set	Non-core types	&	Operator	Intersection.	set
		|	Operator	Union.	set
		-	Operator	Subtraction.	set
		<	Operator	Returns whether the data is a proper subset.	bool
		>	Operator	Returns whether the data is a proper superset.	bool
		<=	Operator	Returns whether the data is a subset.	bool
		>=	Operator	Returns	bool

				whether the data is a superset.	
		in	Operator	Returns whether the data is an element in the list.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
decimal.Decimal	Decimal	+	Operator	Addition.	decim
		-	Operator	Subtraction.	decim
		*	Operator	Multiplication.	decim
		/	Operator	Division.	decim
		%	Operator	Modulus.	decim
		-x	Operator	Negative value.	decim
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool
		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
datetime.datetime	Date and time	year	Attribute	Year.	int
		month	Attribute	Month.	int
		day	Attribute	Day.	int
		hour	Attribute	Hour.	int

		minute	Attribute	Minute.	int
		second	Attribute	Second.	int
		microsecond	Attribute	Microsecond.	int
		+	Operator	Moves forward the date.	datetir
		-	Operator	Calculates the time interval.	datetir
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool
		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
datetime.date	Date	year	Attribute	Year.	int
		month	Attribute	Month.	int
		day	Attribute	Day.	int
		strftime(format)	Method	Formatting.	str
		+	Operator	Moves forward the date.	datetir
		-	Operator	Calculates the time interval.	datetir
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool
		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool

		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
datetime.time	Time	hour	Attribute	Hour.	int
		minute	Attribute	Minute.	int
		second	Attribute	Second.	int
		microsecond	Attribute	Microsecond.	int
		<	Operator	Less than.	bool
		>	Operator	Greater than.	bool
		<=	Operator	Less than or equal to.	bool
		>=	Operator	Greater than or equal to.	bool
		==	Operator	Equal to.	bool
		!=	Operator	Not equal to.	bool
Entity	Binary entity	from_bytes(bs,mime_type=None, encoding="utf-8")	Static method	Constructs an `Entity` object based on the binary data.	Entity
		from_value(obj,mime_type=None, encoding="utf-8")	Static method	Constructs an `Entity` object based on the data.	Entity
		get(key,dafault=None)	Method	Gets the data.	any
		[key]	Subscript operation	Gets the value of the specified `key`.	any
		[^value]	Subscript operation	Gets the parsed value.	any
		[^blob]	Subscript operation	Gets the binary raw data.	bytes

RecordSet	Data set	schema()	Method	Gets the schema.	dict
Record	Single data record	[key]	Subscript operation	Gets the value of the specified `key`.	any
Message	Message	payload	Attribute	Returns the output.	any
		attrs	Attribute	Returns an attribute.	dict
		vars	Attribute	Returns a variable.	dict
		id	Attribute	Returns the unique identifier of the message.	str
		seq_id	Attribute	Returns the flow sequence number.	str
		error	Attribute	Returns an error.	dict
		isthrowing	Attribute	Specifies whether to throw an error.	bool

Other support

The expression mode provides various type methods, built-in functions, and third-party modules for you to choose as needed to quickly implement predefined features. For more information, see Expression Mode Appendix.

Expression Mode Appendix

Last updated：2023-08-03 17:51:33

Methods of native types in Python

To help you manipulate native types in Python, Dataway supports the following common Python methods:

Data Type	Method	Method Type	Feature	Output Type
str	endswith(suffix[, start[, end]])	Method	Compares suffixes.	bool
	split(sep=None, maxsplit=-1)	Method	Splits data.	list
	startswith(prefix[, start[, end]])	Method	Compares the prefixes.	bool
	count(sub[, start[, end]])	Method	Counts substrings.	int
	find(sub[, start[, end]])	Method	Searches for matched substrings.	int
	format(*args, **kwargs)	Method	Formatting.	str
	index(sub[, start[, end]])	Method	Indexes matched substrings.	int
	isascii()	Method	Returns whether the data has only ASCII characters.	bool
	isspace()	Method	Returns whether the data is non-empty and has only whitespace characters.	bool
	encode(encoding="utf-8", errors="strict")	Method	Encodes the data.	bytes
	join(iterable)	Method	Concatenation.	str
	lower()	Method	Converts the data into	str

			lowercase letters.	
	replace(old, new[, count])	Method	Replaces data.	str
	strip([chars])	Method	Removes the prefix and suffix consisting of the specified characters.	str
	upper()	Method	Converts the data into uppercase letters.	str
bytes	count(sub[, start[, end]])	Method	Counts substrings.	int
	find(sub[, start[, end]])	Method	Searches for matched substrings.	int
	index(sub[, start[, end]])	Method	Indexes matched substrings.	int
	decode(encoding="utf-8", errors="strict")	Method	Decodes the data.	str
	replace(old, new[, count])	Method	Replaces the data.	bytes
	rstrip([chars])	Method	Removes the suffix consisting of the specified characters.	bytes
	strip([chars])	Method	Removes the prefix and suffix consisting of the specified characters.	bytes
	split(sep=None, maxsplit=-1)	Method	Splits data.	list
	startswith(prefix[, start[, end]])	Method	Compares the prefixes.	bool
	endswith(prefix[, start[, end]])	Method	Compares the suffixes.	bool
float	is_integer()	Method	Returns whether the data is an integer.	bool
list	count(x)	Method	Counts elements.	int
	index(sub[, start[, end]])	Method	Indexes elements.	int
tuple	count(x)	Method	Counts elements.	int

	index(sub[, start[, end]])	Method	Indexes elements.	int
dict	get(key[, default])	Method	Gets the `key` value.	any
	items	Method	Gets the key-value pair list.	list
set	union(*others)	Method	Returns the new set, which is a union of the specified sets.	set
datetime.datetime	today()	Class method	Gets the current time without a time zone.	datetime.datetime
	fromtimestamp(timestamp, tz=None)	Class method	Constructs the time based on a timestamp.	datetime.datetime
	now()	Class method	Gets the current time with a time zone.	datetime.datetime
	strptime(date_string, format)	Class method	Constructs the formatted time.	datetime.datetime
	time()	Method	Converts the data into a time point.	datetime.time
	date()	Method	Converts the data into a date.	datetime.date
	strftime(format)	Method	Formatting.	str
datetime.date	today()	Class method	Gets the current date.	datetime.date
	strftime(format)	Method	Formatting.	str
datetime.time	strftime(format)	Method	Formatting.	str
DataSet	id()	Method	Gets the data set ID.	int
	partitions()	Method	Gets the number of data set partitions.	int
	schema()	Method	Gets the data set schema.	Schema
Record	data()	Method	Returns the elements as a list.	list

	get(name, default=None)	Method	Gets the data of the specified field name.	any
	schema()	Method	Gets the data set schema.	Schema

Built-in constants and functions

The expression mode supports constants `None` , `True` , and `False` . In addition, for ease of use, the expression mode has many built-in functions, which you can call to quickly implement the corresponding features and get the required data.

Built-in Function	Description
abs(x)	Returns the absolute value of an integer or floating-point number.
all(iterable)	Returns whether all elements are `True` or `None` .
any(iterable)	Returns whether any elements are `True` .
ascii(object)	Prints the `object` without processing non-ASCII characters.
bool([x])	Converts the data into a boolean value.
bytes([source[,encoding[,errors]]])	Converts the data into a `bytes` object.
chr(i)	Returns the Unicode of an integer.
dict(kwarg)/dict(mapping,kwarg)/dict(iterable,kwarg)	Converts the data into a `dict` object.
float([x])	Converts the data into a float.
int(x)/int(x,base)	Converts the data into an integer.
len(s)	Returns the length.
list([iterable])	Converts the data into a list.
max(iterable, [,key,default])/max(arg1,arg2,args[,key])	Returns the maximum value.
min(iterable,[,key,default])/min(arg1,arg2,args[,key])	Returns the minimum value.

Built-in Function	Description
ord(c)	Returns the code of a `char` object.
pow(x,y[,z])	Returns the value of `x` to the power of `y`, modulus `z`.
range(stop)/(start,stop[,step])	Returns an immutable list.
repr(object)	Returns the information of the specified printable object.
round(number[,ndigits])	Returns the number rounded to `ndigits` precision after the decimal point.
set([iterable])	Converts the data into a set.
str(object)/(object,encoding,errors)	Converts the data into a string.
sum(iterable[,start])	Calculates the sum.
tuple([iterable])	Converts the data into a tuple.
type(object)	Returns the data type.

Other third-party modules

The expression mode supports some common third-party Python modules.

Module	Feature	Feature Type	Description	Feature Output
time	asctime([t])	Function	Formats `struct_time`.	str
	ctime([secs])	Function	Formats a timestamp.	str
	gmtime([secs])	Function	Generates the `struct_time` with a UTC time zone.	struct_time
	localtime([secs])	Function	Generates the local `struct_time`.	struct_time

	mktime(t)	Function	Generates a timestamp for `struct_time`.	float
	strftime(format[,t])	Function	Converts `struct_time` into a custom format.	str
	strptime(string[,format])	Function	Converts `struct_time` into a string.	struct_time
	time()	Function	Generates the current timestamp.	float
	time_ns()	Function	Generates the current timestamp in nanoseconds.	int
math	e	Constant	Value of Euler's number.	float
	pi	Constant	Value of Pi.	float
	sqrt(x)	Function	Returns the square root.	float
	log([x,base])	Function	Returns the logarithm.	float
	ceil(x)	Function	Returns the smallest integer greater than or equal to `x`.	int
	floor(x)	Function	Returns the greatest integer less than or equal to `x`.	int
	cos(x)	Function	Returns the	float

			cosine.	
	fabs(x)	Function	Returns the absolute value.	float
	log2(x)	Function	Returns the base-2 logarithm of `x`.	float
	log10(x)	Function	Returns the base-10 logarithm of `x`.	float
	pow(x,y)	Function	Returns the value of `x` raised to the power `y`.	float
	sin(x)	Function	Returns the sine.	float
	tan(x)	Function	Returns the tangent.	float
json	dumps(obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)	Function	Encodes the data into JSON format.	str
	loads(s, *, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)	Function	Decodes the JSON data.	any
base64	b64encode(s, altchars=None)	Function	Encodes the data into Base64 format.	bytes
	b64decode(s, altchars=None, validate=False)	Function	Decodes the Base64 data.	bytes
random	randint(a, b)	Function	Returns a random	int

			integer in the range of [a,b].	
	random()	Function	Returns a random floating-point number in the range of [0,1).	float
urllib	parse.quote(string, safe='/', encoding=None, errors=None)	Function	Transcodes symbols.	str
	parse.urlencode(query, doseq=False, safe='', encoding=None, errors=None)	Function	Encodes the URL.	float

Python Code Mode

Last updated：2023-08-04 10:44:00

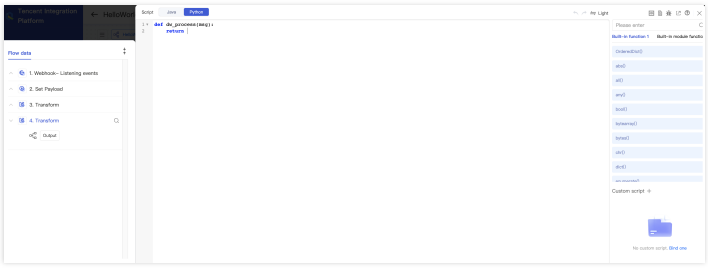
The Python code mode extends the expression mode to support more complex syntax and more powerful features. It is more difficult to use and requires a basic knowledge of Python programming.

Use of IDE

Directions

1. Hover over any Dataway textbox, and the mode selection buttons will pop up automatically. Click **Code** to enter the code mode.

2. Click the textbox, and the code editor will pop up. The Python script editor is selected by default, where you can edit a Python script.



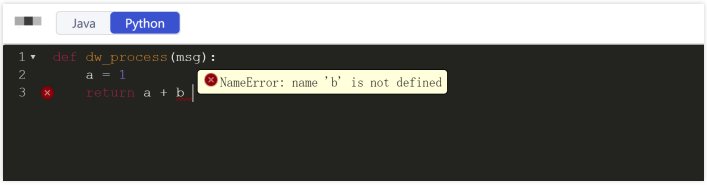
3. After editing the script, click **Confirm**.

Feature description

The editor provides various IDE-like features, including syntax check, formatting, script debugging, autocomplete, and code highlighting.

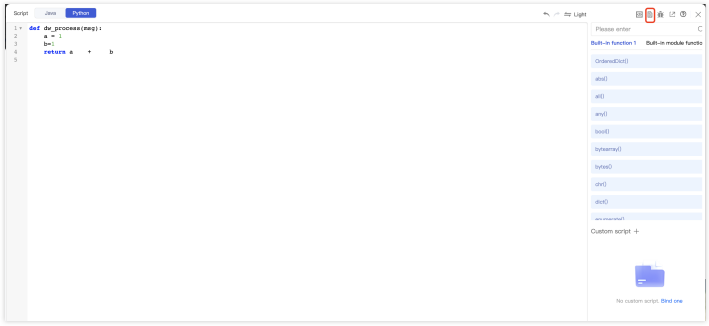
- **Syntax check**
  The Python script editor can check the syntax of the Python script in real time and display conspicuous prompts on the left of the script editing window and below the incorrect code. When you hover over an error prompt, the detailed error message will be displayed.

You can modify the Python script based on the syntax prompts. Only code that passes the syntax check can be saved successfully.

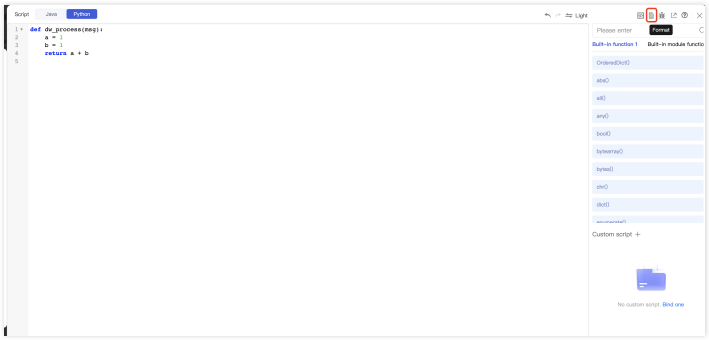


- **Formatting**

  The Python script editor offers a formatting feature button. You can click the formatting icon in the top-right corner to quickly format the Python script, making the code simpler and more standard.



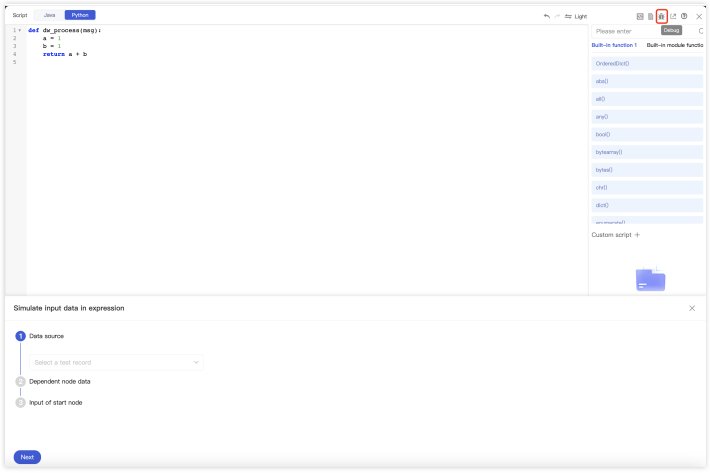
After you click the formatting icon in the top-right corner, the formatted code is as shown below:



- **Script debugging**

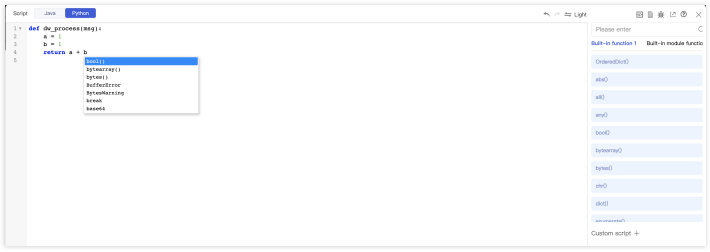
  The Python script editor offers a debugging feature button. You can click the **Debug** icon in the top-right corner to

debug the Python script online as instructed in Script debugging.



- **Autocomplete**

  When you input content in the textbox, the Python script editor will automatically display the syntax prompts based on the current context below the cursor. Generally, syntax prompts include built-in functions, keywords, and third-party modules.



- **Reference on the flow data panel**

  In code mode, you can reference data on the flow data panel. For more information, see Flow Data Panel.

- **Code highlighting**

  The Python script editor highlights the Dataway code and completes parentheses and brackets by default.

Script Structure

A complete script in Python code mode must be in compliance with Python 3 syntax and contain the entry function definition `def dw_process(msg)` , such as:

```
def dw_process(msg):
sq = func(3)
val = {
'square': sq,
'data': msg.payload['realData'] + 1
```

```
}
return Entity.from_value(val, mime_type='application/json')

def func(x):
return x*x
```

The `dw_process` entry function only accepts a parameter `msg`, which represents the message that Dataway needs to process currently. The returned value of the function is also the returned value of the script.

Enter the above expression in the **Set Payload** component. If the input message of the component is in JSON structure `{"realData": 123}`, the calculation output result of the Python script will be as follows:

```
{
"square": 9,
"data": 124
}
```

Basic Dataway Syntax Description

The Python code mode is implemented based on Python 3 syntax. This section describes the basic syntax of the Python code mode.

Keyword

The Python code mode supports the following keywords. As reserved words in Python code mode, keywords won't be treated as any identifier name.

Keyword	Description
True	Boolean value. `True` indicates true, which is opposite to `False`.
True	Boolean value. `False` indicates false, which is opposite to `True`.
None	Null.
and	Logical AND.
or	Logical OR.
not	Logical NOT.
as	Creates an alias.
assert	Uses an assertion to test an expression.

Keyword	Description
break	Stops a loop statement.
continue	Jumps out of the current loop.
def	Defines a function.
if/else/elif	Forms a conditional statement.
for	Forms a loop statement.
global	Declares a global variable.
in	Checks whether a value is present in an object.
is	Checks whether two variables refer to the same object.
lambda	Creates an anonymous function. A function can be implemented in a single line.
nonlocal	Declares a nonlocal variable in a nested function. This keyword can modify variables defined externally.
pass	Null statement, which can be used as a placeholder.
raise	Throws an exception.
return	Returns the value of a function.

Line and indentation

The Python code mode identifies code blocks based on indentation. Different numbers of indentation spaces indicate different code levels. The number of indentation spaces at the same level must be the same.

Operator

The Python code mode supports the following common operators, including arithmetic, comparison, assignment, logical, and bitwise operators. Suppose variable `a` is `5` and `b` is `3`, the examples are as listed below:

Operator	Description	Example
=	Assignment.	c = 3
+	Addition.	a + b = 3
-	Subtraction.	a - b = 2
*	Multiplication.	a * b = 15

Operator	Description	Example
/	Division.	15 / a = b
%	Modulus, which returns the remainder of a division.	16 % b = 1
**	Exponentiation.	a ** b = 125
//	Returns the greatest integer less than or equal to the specified value.	a // b = 1
+=	Addition assignment.	`c += a` is equivalent to `c = c + a`.
-=	Subtraction assignment.	`c -= a` is equivalent to `c = c - a`.
*=	Multiplication assignment.	`c *= a` is equivalent to `c = c * a`.
/=	Division assignment.	`c /= a` is equivalent to `c = c / a`.
==	Checks whether two values are equal.	`a == b` returns `False`.
!=	Checks whether two values are not equal.	`a != b` returns `True`.
>	Checks whether a value is greater than another.	`a > b` returns `True`.
<	Checks whether a value is less than another.	`a < b` returns `False`.
>=	Checks whether a value is greater than or equal to another.	`a >= b` returns `True`.
<=	Checks whether a value is less than or equal to another.	`a <= b` returns `False`.
&	Bitwise AND.	a & b = 1(0101 & 0011 = 0001).
|	Bitwise OR.	a | b = 7 (0101 & 0011 = 0111).
^	Bitwise XOR.	a ^ b = 6 (0101 ^ 0011 = 0110).
~	Bitwise negation.	~a = -6.
<<	Left shift.	a << 3 = 20 (0000 0101 << 3 = 0001 0100).
<<	Right shift.	a >> 1 = 2 (0101 >> 1 = 0010).

Conditional and loop control statements

- The Python code mode uses `if` , `elif` , and `else` statements for conditional control. For example, the value of `a` is checked to return different strings:

```
def dw_process(msg):
a = 100
if a < 10:
return 'a is lower than 10'
elif a <= 100 and a >= 10:
return 'a is between 10 and 100'
else:
return 'a is bigger than 100'
```

The execution result of the Dataway expression is `a is between 10 and 100` .

- The Python code mode uses the `for` loop for loop control. For example, a `for` loop is used to get the product of elements in `a` :

```
def dw_process(msg):
a = [1, 2, 3, 4]
num = 1
for i in a:
num *= i
return num
```

The execution result of the Dataway expression is `24` .

Function definition

In Python code mode, you can use the `def` keyword to define a function as follows: the `def` keyword is followed by a function name and parameter name list, the function definition line ends with a colon `:` , the next line is indented by default, and the entire function ends with a `return` statement; if no `return` statement is used, `None` will be returned.

After defining a function, you can call and execute it in another function. In Python code mode, the default entry function `dw_process` doesn't need to be declared manually. To customize a function, directly define it below the `dw_process` entry function. For example, a function `test()` is defined to calculate the sum of the list elements and is called in the `dw_process` function, and a `return` statement is used at the end to return the result:

```
def dw_process(msg):
a = [1, 2, 3, 4]
return add_list(a)
```

```
def add_list(alist):
sum = 0
for i in reversed(alist):
sum += i
return sum
```

The final output result is `10` .

Module call

The Python code mode has various built-in third-party modules, including `time` , `json` , `math` , `base64` , `hmac` , `random` , `hashlib` , `Crypto` , `socket` , `struct` , `decimal` , and `datetime` . When using a module, you can directly reference the module name without using the `import` keyword. For the specific function description, see Expression Mode Appendix. For example, a JSON string is received and converted into a dictionary:

```
def dw_process(msg):
jsonStr = '{"a": 1, "b": 2, "c": 3}'
jsonDict = json.loads(jsonStr) # Convert into a `dict` object
num = 1
for k, v in jsonDict.items(): # Traverse the `dict` object
num += math.pow(v, 2)
return num
```

The final output result is `15.0` .

Comment

In Python code mode, single-line comments start with `#` , and you can use multiple `#` symbols, `'''` , or `"""` for multi-line comments. For example, execute the following code:

```
# Dataway comment

'''
Dataway comment
Dataway comment
'''

"""
Dataway comment
Dataway comment
"""
def dw_process(msg):
return 'Dataway Hello World!'
```

The output result is:

```
Dataway Hello World!
```

Note：
The Python code mode provides the syntax check feature to check the syntax in real time and prompt errors when you write code. For the detailed syntax description, see The Python Language Reference.

`dw_process` Entry Function

`dw_process` is the main entry function in Python code mode, which acts like the `main` function in C or C++.

`dw_process` only accepts a parameter of the Message type, and its returned value is the output value of the script in Python code mode.

As a stage in the data processing process in iPaaS, the `dw_process` function currently supports core types for its returned value.

For more information on the data types and returned value in Python code mode, see Data Type System.

Data Type System

Type	Description	Unique to Dataway	Example
str	String, i.e., native `str` type in Python.	No	"abc"
None	`None` in Python.	No	None
bool	Boolean, i.e., native `bool` type in Python.	No	True/False
float	Float, i.e., native `float` type in Python.	No	123.123
int	Integer, i.e., native `int` type in Python.	No	123
bytes	Byte array, i.e., `bytes` type in Python.	No	b'this_is_a_bytes'
set	Set, i.e., `set` type in Python.	No	{1,2,3}

Type	Description	Unique to Dataway	Example
list	List (a sequence container), i.e., native `list` type in Python.	No	[1,2,3]
dict	Dictionary (a key-value pair container), i.e., native `dict` type in Python.	No	{1:1, 'key': 'value'}
Entity	Entity data in iPaaS, which represents a binary object and is accessed as an `Entity` object in Dataway. It contains information such as `blob`, `mime_type`, and `encoding`.	Yes	`payload` in a message constructed by the HTTP Listener component, such as `msg.payload`
MultiMap	Multi-value map. Like `xml` but unlike `dict`, this type supports duplicate `key` values.	Yes	Object obtained after data in `application/www-form-urlencoded` format is parsed
FormDataParts	Array + list data structure, which is similar to `orderDict` in Python.	Yes	Object obtained after data in `multipart/form-data` format is parsed
Message	Message in iPaaS, which is accessed as a `Message` object in Dataway.	Yes	`msg` parameter in the `dw_process` entry function

Note：

- The above types can be used in the Python code mode, but **the data type of the returned value of the `dw_process` function must be a core type**.
- If the output value of a Dataway expression is the final result returned by the flow, the types supported for the returned value will also be subject to the flow components. If an HTTP Listener component is used as the first component in the flow, the final `payload` value must be of `Entity` type.

Script Debugging

The Python code mode supports script debugging to help troubleshoot problems and verify the result. With this feature, you can manually define the input parameter `msg` and click **Test** to directly view the script execution result, debugging log, and error message.

1. Enter an expression in the script textbox.
   In Dataway debugging mode, you can use the `print()` function in the expression to print the information to be observed. The printed message will be displayed on the UI after the script is executed.

2. Click the **Debug** icon in the top-right corner of the script editing window. In the simulated data configuration pop-up window, you can set the payload, attributes, and variables of `msg` . After completing the configuration, click **Start test**, and the system will automatically assemble a `msg` parameter and pass it to the `dw_process` function as the script input.

3. After the `dw_process` function is executed, the execution result and printed debugging log will pop up at the bottom of the editing window. If errors occur during execution, error messages will be displayed.

   - **Output**: It displays the execution result of the Dataway expression.

   - **Error**: It displays the script execution error message. If no errors occur, a green tick will be displayed.

Other Support

The Python code mode provides various built-in functions and third-party modules for you to choose as needed to quickly implement predefined features. For more information, see Python Appendix.

Python Appendix

Last updated：2023-08-04 11:08:20

Built-in Functions

Currently, the Python code mode supports the following built-in functions:

No.	Built-in Function	Description
1	abs()	Calculates the absolute value.
2	all()	Checks whether all elements in a sequence (set, list, tuple, or dictionary) meet the specified condition.
3	any()	Checks whether any elements in a set meet the specified condition.
4	bool()	Constructs a Boolean value.
5	bytearray()	Constructs a byte array.
6	bytes()	Constructs an empty `bytes` object.
7	chr()	Returns the ASCII character of an integer within the range of 0–256.
8	dict()	Creates a dictionary.
9	enumerate()	Lists the elements and element subscripts in a traversable data object. This function is generally used in a `for` loop.
10	filter()	Filters a set. For example, the result of `list(filter(lambda x:x>=100, [1,3,4,100,102]))` is `[100,102]`.
11	float()	Constructs a floating-point number.
12	getattr()	Calculates the attribute value of an object.
13	hasattr()	Checks whether an object has an attribute.
14	hash()	Calculates the hash value.
15	id()	Returns the unique identifier of an object.
16	int()	Constructs an integer.

No.	Built-in Function	Description
17	isinstance()	Checks whether an object is of a certain type.
18	iter()	Generates an iterator.
19	len()	Gets the number of elements in a set.
20	list()	Constructs a list.
21	map()	Maps the specified sequence according to the function. For example, the result of `list(map(lambda x: x * 2, [1, 2, 3, 4, 5]))` is `[2, 4, 6, 8, 10]`.
22	max()	Gets the maximum value.
23	min()	Gets the minimum value.
24	next()	Returns the next item of an iterator. This function is used together with `iter()`.
25	objects()	Returns an empty object.
26	ord()	Returns the integer value of an ASCII character.
27	pow()	Calculates the power of a number.
28	print()	Prints the relevant information during debugging in Python code mode (it takes effect only when you use the debugging feature when editing a Dataway expression).
29	range()	Creates an iterable object. For example, the result of `list(range(5))` is `[0, 1, 2, 3, 4]`.
30	reversed()	Creates a revere iterator. For example, the result of `list(reversed('abcdefg'))` is `['g', 'f', 'e', 'd', 'c', 'b', 'a']`.
31	round()	Returns the nearest integer of a value.
32	set()	Creates a set.
33	slice()	Sets a slice of elements.
34	sorted()	Sorts.
35	str()	Constructs a string.
36	sum()	Gets the sum of values.

No.	Built-in Function	Description
37	tuple()	Constructs a tuple.
38	type()	Returns the data type of an object.
39	zip()	Zips elements in an iterable object into multiple tuples. For example, the result of `list(zip([1,2,3], [4,5,6]))` is `[(1, 4), (2, 5), (3, 6)]`.

Third-Party Module

time

`time` is a library for time processing. For more information, see 16.3. time - Time access and conversions. It has been built in the Python code mode and can be referenced directly.

Currently, the Python code mode supports the following library functions/types:

No.	Library Function/Type	Description
1	altzone	Returns the offset of the local DST time zone from UTC in seconds.
2	asctime	Converts a `struct_time` object into a time string.
3	ctime	Converts a timestamp into a time string.
4	mktime()	Converts a `struct_time` object into a timestamp.
5	strftime()	Formats a `struct_time` object.
6	strptime()	Parses an event string in the specified format and returns a structured `struct_time` object.
7	timezone	Returns the current time zone.
8	tzname	Returns the name of the current time zone.
9	time()	Returns the current time.
10	localtime	Converts a timestamp into the local time of the local time zone and returns a `struct_time` object.

json

`json` is a library for JSON data processing. For more information, see 19.2. json - JSON encoder and decoder. It has been built in the Python code mode and can be referenced directly.

Currently, the Python code mode supports the following `json` functions:

No.	`json` Function	Description
1	dumps()	Encodes a Python object into a JSON string.
2	loads()	Parses a JSON string into a Python object.

math

`math` is a library for arithmetic operations. For more information, see 9.2. math - Mathematical functions. It has been built in the Python code mode and can be referenced directly.

Currently, the Python code mode supports the following `math` functions:

No.	`math` Function	Description
1	math.ceil(x)	Returns the ceiling of `x`, i.e., the smallest integer greater than or equal to `x`. If `x` is not a floating-point number, delegates to `x.ceil()`, which should return an integer value.
2	math.floor(x)	Returns the floor of `x`, i.e., the largest integer less than or equal to `x`. If `x` is not a floating-point number, delegates to `x.floor()`, which should return an integer value.
3	math.fabs(x)	Returns the absolute value of `x`.
4	math.pow(x,y)	Returns `x` raised to the power `y`.
5	math.sqrt(x)	Returns the square root of `x`.

The following constants are supported:

No.	Constant	Description
1	math.pi	Mathematical constant π = 3.141592..., to available precision.
2	math.e	Mathematical constant e = 2.718281..., to available precision.

No.	Constant	Description
3	Floating-point positive infinity (for negative infinity, use `-math.inf`), which is equivalent to the output of `float('inf')`.	
4	math.nan	Floating-point "not a number" (NaN) value, which is equivalent to the output of `float('nan')`.

base64

`base64` is a library for Base64 encoding/decoding. For more information, see 19.6. base64 - Base16, Base32, Base64, Base85 Data Encodings. It has been built in the Python code mode and can be referenced directly. The following functions are supported:

No.	Supported Function	Description
1	base64.b64encode(s)	Encodes the bytes-like object `s` using Base64 and returns the encoded bytes.
2	base64.b64decode(s)	Decodes the Base64 encoded bytes-like object or string `s` and returns the decoded bytes.

hmac

`hmac` is a library for HMAC encoding/decoding. For more information, see 15.2. hmac - Keyed-Hashing for Message Authentication. It has been built in the Python code mode and can be referenced directly. The following functions are supported:

No.	Supported Function	Description
1	hmac.new(key)	Return a new `hmac` object. `key` is a `bytes` or `bytearray` object giving the secret key.

random

`random` is a library for random number generation. For more information, see 9.6. random - Generate pseudo-random numbers. It has been built in the Python code mode and can be referenced directly. The following functions are supported:

No.	Supported Function	Description

No.	Supported Function	Description
1	random.randint(a,b)	Returns a random integer `N` such that `a <= N <= b`.

hashlib

`hashlib` is a library for hash value generation. For more information, see 15.1. hashlib - Secure hashes and message digests. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	hashlib.sha256()	Creates a SHA-256 hash object.
2	hashlib.md5()	Creates an MD5 hash object.
3	hashlib.sha1()	Creates a SHA-1 hash object.

datetime

`datetime` is a library for time and date processing. For more information, see 8.1. datetime - Basic date and time types. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	datetime.date	Idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Valid attributes: `year`, `month`, `day`.
2	datetime.time	Idealized time, independent of any particular day, assuming that every day has exactly 24 * 60 * 60 seconds. Valid attributes: `hour`, `minute`, `second`, `microsecond`, `tzinfo`.
3	datetime.datetime	Combination of a date and a time. Valid attributes: `year`, `month`, `day`, `hour`, `minute`, `second`, `microsecond`, `tzinfo`.
4	datetime.timedelta	Duration expressing the difference between two `date`, `time`, or `datetime` objects to microsecond resolution.
5	datetime.timezone	Offset from UTC.
6	datetime.tzinfo	Time zone information objects. These are used by the `datetime` and `time` classes to provide a customizable notion of time adjustment (for example, to account for time zone and/or DST).

decimal

`decimal` is a library for floating-point number processing. For more information, see 9.4. decimal - Decimal fixed point and floating point arithmetic. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	decimal.Decimal	Constructs a decimal floating-point object.

socket

`socket` is the underlying implementation of TCP sockets in Python. For more information, see 18.1. socket - Low-level networking interface. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	socket.htonl()	Converts 32-bit positive integers from host to network byte order.
2	socket.ntohl()	Converts 32-bit positive integers from network to host byte order.

pycryptodome

`pycryptodome` is a dedicated third-party encryption tool library. For more information, see Welcome to PyCryptodome's documentation. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	Crypto.Util.Padding	Provides minimal support for adding and removing standard padding from data. This module provides the `pad()` and `unpad()` methods.
2	Crypto.Cipher.AES	Implements AES encryption. This module has a fixed data block size of 16 bytes. Its keys can be 128, 192, or 256 bits long. It provides the `new()` method.

struct

`struct` is a library for binary file packing. For more information, see 7.1. struct - Interpret bytes as packed binary data. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	struct.pack(format, v1, v2, ...)	Returns a `bytes` object containing the values `v1` , `v2` , ... packed according to the format string `format` . The arguments must match the values required by the format exactly.
2	struct.unpack(format, buffer)	Unpacks from the buffer `buffer` according to the format string `format` and returns a tuple.

urllib

`urllib` is a library for URL processing. For more information, see 21.5. urllib - URL handling modules. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	urllib.parse.urlparse()	Gets URL parameters and parses the URL into a tuple of six strings: protocol, location, path, parameters, query, and fragment identifier.
2	urllib.parse.unquote()	Decodes the encoded URL.

csv

`csv` is a library for CSV file read/write. For more information, see 14.1. csv - CSV File Reading and Writing. It has been built in the Python code mode and can be referenced directly. The following functions/attributes are supported:

No.	Supported Function/Attribute	Description
1	csv.reader()	Creates a reader object, which will traverse over lines in a CSV file object.

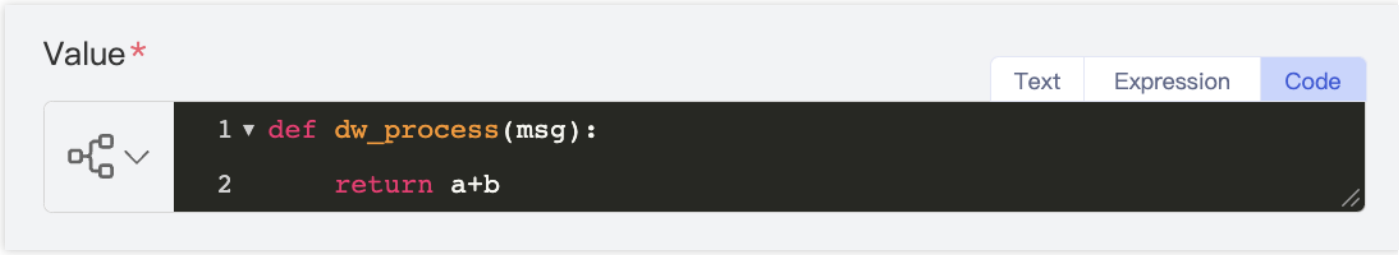
Java Code Mode

Last updated：2023-08-03 17:51:33

To make it easier for users experienced in Java to connect to iPaaS, the Dataway code mode supports Java scripts.

Use of IDE

1. Hover over any Dataway textbox, and the mode selection buttons will pop up automatically. Click **Code** to enter the code mode.



2. Click the textbox, and the code editor will pop up. Click **Java** to enter the Java script editor.

3. After editing the script, click **Confirm**.

The Java code mode supports data reference on the flow data panel.

Script Structure

A Java script must be in compliance with JDK 8 syntax.

The class name must be `Handler`, and you must define a function with the signature `Object eval(Message msg)` as the entry function.

```
import com.tencent.ipaas.dataway.common.message.Message;
import com.tencent.ipaas.dataway.common.message.DataRef;
```

You can add other `import` statements based on required data types.

```
// The following `import` statements are fixed and cannot be deleted.
import com.tencent.ipaas.dataway.common.message.Message;
import com.tencent.ipaas.dataway.common.message.DataRef;

/**
* Entry class of `dataway-java`, which must be named `Handler`
*/
public class Handler {
/**
* Entry function, whose signature must be `Object eval(Message msg)`
* @param msg Input a `Message` object
* @return Any data object supported by Dataway
*/
public Object eval(Message msg) {
return msg.getPayload();
}
}
```

Data Types

The Java code mode supports various data types, making it easy for you to manipulate different data.

Type	Description	Corresponding Python Type
null	`null` in Java.	None
String	String, i.e., native `String` type in Java.	str
Boolean	Boolean, i.e, native `bool` type in Java.	bool
float/Float	Float, i.e., native `float` type in Java.	float
int/Integer	Integer, i.e., native `int` type in Java.	int
long/Long	Long integer, i.e., native `Long` type in Java.	
short/Short	Short integer, i.e., native `Short` type in Java.	

byte[]	Byte array, i.e., `byte[]` type in Java.	bytes
java.util.List	List (a sequence container), i.e., native `List` type in Java.	list
java.util.Map	Dictionary (a key-value pair container), i.e., native `Map` type in Java.	dict
java.time.OffsetDateTime	Time, i.e., native `OffsetDateTime` type in Java.	datetime.datetime
java.time.LocalDate	Date, i.e., native `LocalDate` type in Java.	datetime.date
java.time.OffsetTime	Time, i.e., native `OffsetTime` type in Java.	datetime.time
java.math.BigDecimal	Decimal number, i.e., native `BigDecimal` type in Java.	decimal.Decimal
com.tencent.ipaas.dataway.common.message.Entity (data type unique to iPaaS)	Entity data in iPaaS, which represents a binary object and is accessed as an `Entity` object. It contains information such as `blob`, `mimeType`, and `encoding`.	Entity
com.tencent.ipaas.dataway.common.message.Multimap (data type unique to iPaaS)	Multi-value map. Like `xml` but unlike `dict`, this type supports duplicate `key` values. It is inherited from `HashMap<String, List>`.	MultiMap

com.tencent.ipaas.dataway.common.message.FormDataParts (data type unique to iPaaS)	Array + list data structure, which is similar to `orderDict` in Python. It is inherited from `LinkedHashMap<String, Object>`.	FormDataParts
com.tencent.ipaas.dataway.common.message.Message (data type unique to iPaaS, which cannot be constructed in Dataway)	Flow message in iPaaS, which is accessed as a `Message` object.	Message
com.tencent.ipaas.dataway.common.message.DataSet (data type unique to iPaaS, which cannot be constructed in Dataway)	Data set in data integration, which is generated by the data integration component.	RecordSet
com.tencent.ipaas.dataway.common.message.Record (data type unique to iPaaS, which cannot be constructed in Dataway)	Single data record in data integration, which contains the schema.	Record

com.tencent.ipaas.dataway.common.message.Schema (data type unique to iPaaS, which cannot be constructed in Dataway)	Data dictionary in data integration, which describes the metadata.	Schema
com.tencent.ipaas.dataway.common.message.RecordField (data type unique to iPaaS, which cannot be constructed in Dataway)	Field information in data integration, which describes the metadata of a single field.	Schema

Using an `Entity` Object

Basic methods

In Java code mode of iPaaS, the `Entity` type is used to represent the entity data in flows. It is an encapsulation object of binary data and contains `blob` , `mimeType` , and `encoding` .

Field	Description
blob	Raw binary data.
mimeType	Content format of binary data, such as `application/json` , `application/www-form-urlencoded` , and `multipart/form-data` .
encoding	Character encoding type of binary data, such as `utf-8` and `gbk` .

You can access content in `Entity` as follows:

Access Method	Description
byte[] getBlob()	Gets the payload data of the message object. A `byte[]` object will be returned.
String getMimeType()	Gets the MIME type of the message object. A `String` object will be returned.
String getEncoding()	Gets the encoding type of the message object. A `String` object will be returned.
Object getValue()	Deserializes `blob` in the payload based on the MIME and encoding types and returns the result. This type is defined in the type system in Java code mode.
Object get(Object key)	Deserializes the content in `message` based on the MIME and encoding types and gets the value of the specified key.

Currently, the MIME types supported for deserialization and types of the deserialized value are as listed below:

- text/plain → String
- application/json → Object, which is the same as JSON
- application/x-www-form-urlencoded → Multimap
- application/xml → Map
- application/csv → List<Map<String,String>>, i.e., list of mappings between field names and values
- multipart/form-data → FormDataParts

Constructor

`Entity.fromValue` static method

This method is used to encapsulate the value type `data` into an `Entity` object and return it as follows:

```
Entity.fromValue(Object value, String mimeType, String encoding)
```

The `fromValue` function tries serializing `value` based on the specified MIME and encoding types to get the data of `byte[]` type, encapsulates it into an `Entity` object, and returns the object.

- The `mimeType` parameter is required. Currently, six MIME types are supported: `text/plain`, `application/json`, `application/x-www-form-urlencoded`, `application/csv`, `application/xml`, and `multipart/form-data`.
- The `encoding` parameter is required and can be any valid encoding type.

`Entity.fromBytes` static method

This method is used to encapsulate a `String` or a `byte[]` object into an `Entity` object and return it as follows:

```
Entity.fromBytes(Object data, String mimeType, String encoding)
```

The verification rules of the MIME type and encoding type parameters in `fromBytes` are similar to those in the `fromValue` function but differ in that the value of the MIME type parameter is not limited and can be any MIME type.

- If the `data` parameter passed to the `fromBytes` function is of `byte[]` type, the function will directly return an `Entity` object consisting of parameters `data`, `mimeType`, and `encoding`.
- If the passed `data` parameter is of `String` type, it will be encoded as a `byte[]` object based on the `encoding` parameter and constructed as an `Entity` object.

Use limits

An `Entity` object is essentially an encapsulation object of binary data. For ease of use, object content access methods like `entity.get()` are provided. Before using these features, note that for some special operations, the system will try deserializing the binary data in the `Entity` object, and runtime errors will occur if parsing fails. Such special operations include:

- Using `entity.getValue()` to get the structured result after parsing.
- Using `entity.get(key)` to get the content of an element in the structured result.

If you perform the above special operations on a non-compliant `Entity` object, runtime errors will occur.

FAQs

Last updated：2023-08-03 17:51:33

What is the relationship between Dataway Python expressions and Python scripts?

Dataway Python expressions are encapsulated based on Python 3. Their features are also tailored based on Python, and you must define a valid entry function `dw_process(msg)`. Therefore, all Dataway expressions have a returned value after being executed normally, but Python scripts don't necessarily have a specific returned value.

Where can I use Dataway expressions?

Currently, almost all core iPaaS components can calculate the value of expressions. Dataway plays a key role in implementing the dynamic value calculation capabilities of iPaaS.

How do I troubleshoot a Dataway script execution error?

1. Check to make sure that the Dataway script contains no syntax errors and has passed the syntax check during editing.
2. View the execution error log to locate the cause of the Dataway script error and the line of the script which caused the error for further problem analysis and locating.
3. You can also debug the Dataway script for troubleshooting.

How do I use a third-party module?

By default, Dataway supports only the following built-in modules: `time`, `json`, `math`, `base64`, `hmac`, `random`, `hashlib`, `Crypto`, `socket`, `struct`, `decimal`, `urllib`, `csv`, and `datetime`.

Generally, such modules are enough for Dataway scripts. If you do need to use more third-party modules, please submit a ticket to request support for an additional third-party module.