



TIBCO Flogo[®] Enterprise

User Guide

Version 2.25.2 | January 2025

Contents

Contents	2
Introduction	6
Concepts	6
Creating Your First REST API	8
Procedure	9
App Development	28
Creating and Managing a Flogo App Using the UI	28
Creating an App	28
Validating your App	33
Editing an App	34
Auto-Upgrade of Activities, Triggers, and Connections	34
Renaming an App	35
Editing the Version of an App	35
Using App Tags	36
Using Notes	39
Switching Between Display Views On the App Page	41
Deleting an App	43
Exporting and Importing an App	43
App File Persistence	52
Creating Flows and Triggers	55
Flows	55
Triggers	94
Synchronizing a Schema Between Trigger and Flow	95
Data Mappings	95
Data Mappings Interface	95
Mapping Data from the Data Mappings Interface	98

Scopes in Data Mappings	101
Data Types	103
Reserved Keywords to be Avoided in Schemas	104
Mapping Different Types of Data	106
Mapping Data by Using if/else Conditions	138
Using Functions	142
Using Expressions	144
Supported Operators	145
Combining Schemas Using Keywords	146
Developing APIs	148
Using an OpenAPI Specification	149
Using GraphQL Schema	155
Using App Properties and Schemas	159
App Properties	159
App Schemas	177
Using Connectors	181
Creating Connections	182
Editing Connections	182
Deleting Connections	183
Using Extensions	183
Important Considerations	184
Creating Extensions	185
Uploading Extensions	192
Pulling Extensions from an Open Source Public Git Repository	196
Deleting Extensions or Extension Categories	198
Flow Tester	198
Testing Flows from the UI	199
Testing Flows from the CLI	215
Using the test command to test your flow from the CLI	216
Unit Testing	219
Creating and Running a Test Case	220
Creating and Running a Test Suite	229

Exporting and Importing a Unit Test	231
Enabling On-premises Services in Unit Testing	232
Unit Testing for the CI/CD	232
Deployment and Configuration	235
Building an App Executable	235
Building the App	235
App Configuration Management	240
Consul	240
AWS Systems Manager Parameter Store	248
AWS AppConfig	254
Environment Variables	259
Overriding Security Certificate Values	262
Encrypting Password Values	264
Container Deployments	265
Kubernetes	265
Amazon Elastic Container Service (ECS) and Fargate	272
Pivotal Cloud Foundry	273
Microsoft Azure Container Instances	279
Google Cloud Run	283
Red Hat OpenShift	286
Serverless Deployments	291
Developing for Lambda	291
Deploying a Flogo App to Microsoft Azure Functions	302
Deploying a Flogo App in Knative	308
Monitoring	313
About the TIBCO Flogo Enterprise Monitoring App	313
About TIBCO Flogo® Flow State Manager	327
Viewing Statistics by Using Flogo Enterprise Monitoring app	335
App Metrics	344
Distributed Tracing	358
Using APIs	373

CPU and Memory Profiling	376
Monitoring and Managing Enterprise Apps in TIBCO Cloud Integration	378
Environment Variables	378
Pushing Apps to TIBCO Cloud	384
Best Practices	387
Performance Tuning	392
Tuning Environment Variables	392
FLOGO_RUNNER_TYPE	393
FLOGO_LOG_LEVEL	398
GOGC	400
Flow Limit	402
CPU and Memory Monitoring	403
Top Command	403
Docker Stats Command	404
Runtime Statistics and Profiling	404
Samples	406
TIBCO Documentation and Support Services	407
Legal and Third-Party Notices	409

Introduction

TIBCO Flogo® Enterprise is an open-core product based on Project Flogo™, an open-source ecosystem for event-driven apps. Its ultra-light app engine offers you the flexibility to deploy your Flogo apps in containers, as serverless functions, or as static binaries on IoT edge devices. You can quickly implement microservices, serverless functions, event-driven apps, integrations, and APIs.

Flogo apps are created in TIBCO Cloud™ Integration, which provides a wizard-driven web-based tool to create integration apps without having to leave your browser. For more information on creating and using Flogo apps, see [TIBCO Cloud Integration](#) documentation.

To migrate an existing app (created using release 2.14.0 or a prior release) from Flogo Enterprise to TIBCO Cloud™ Integration, see *TIBCO Flogo® Enterprise Transition Guide*.

If you are entitled to TIBCO Flogo Enterprise and you would like to get access to TIBCO Cloud Integration to design your TIBCO Flogo apps, you can send a Support Request on <https://supportapps.tibco.com/> to provide the required Cloud contact details for us to provision Cloud access.

Concepts

This section describes the main concepts used in the Flogo Enterprise environment.

Apps

Flogo apps are developed as event-driven apps using triggers and actions and contain the logic to process incoming events. A Flogo app consists of one or more triggers and one or more flows.

Trigger

Triggers receive events from external sources such as Apache Kafka®, Salesforce, GraphQL. Handlers residing in the triggers, dispatch events to flows. Flogo Enterprise provides a set of out-of-the-box triggers. Also provides a range of connectors for receiving events from a variety of external systems.

Flow

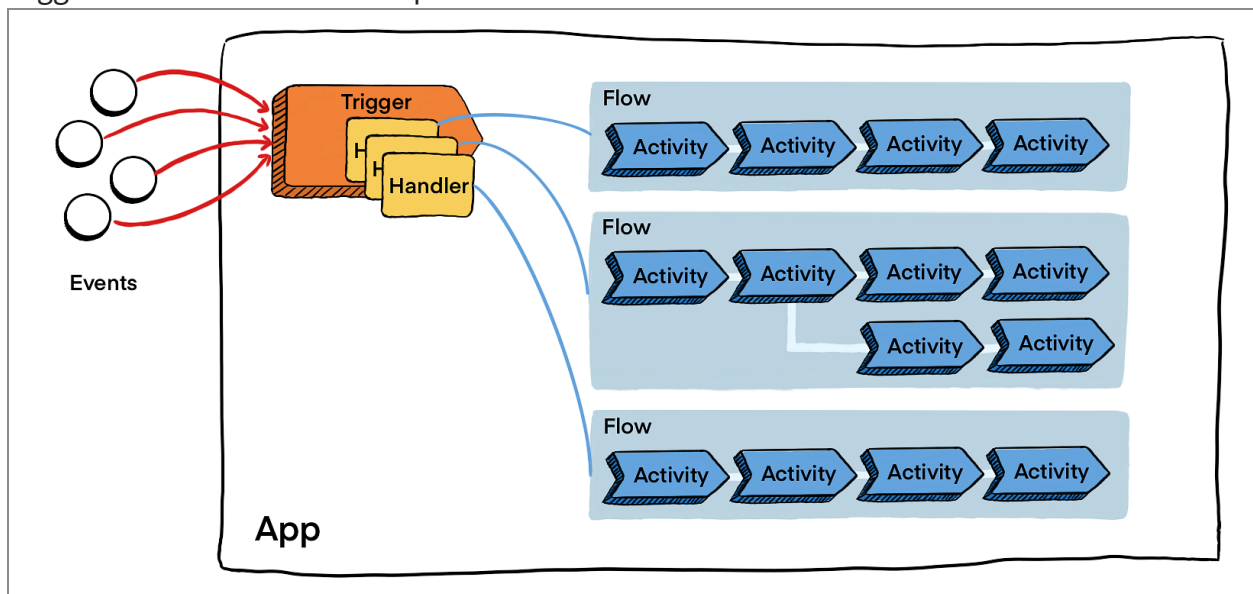
The Flogo provides a set of actions for processing events in a manner suitable to your implementation logic. The flow allows you to implement the business logic as a process. You can visually design and test the flows using the UI. A flow consists of one or more activities that perform a specific task. Activities are linked to facilitate the flow of data between them and contain conditional logic for branching. Each flow is also connected to a default error handler. A Flogo app can have one or more flows. A flow can be activated by one or more triggers within the app.

Activity

Activities perform specific tasks within the flow. A flow typically contains multiple activities.

How Flogo Works

The trigger consists of one or more handlers that serve as the means of communication between the trigger and the flow. When the trigger receives an event, it uses the respective flow handlers to pass the data from the event to the flow in the form of flow input. The business logic in the flow then can use the event data coming in through the flow input. When the trigger expects a reply from the flow, the data from the flow is passed on to the trigger in the form of flow output. A flow can contain one or more conditional branches.



Summary:

1. [Create an app.](#)

2. [Create a flow](#) in your app.
3. [Add one or more activities to the flow](#) and configure them.
4. Optionally, [add a trigger to your flow](#). You can add one or more triggers to a flow as and when you need them.
5. [Build your app](#).

Creating Your First REST API

This tutorial walks you through the steps to build a simple app with a REST service in Flogo Enterprise. It shows how to create a basic app that returns the booking details of a specific customer based on a query sent to the app. In this tutorial, the query sent to the app checks whether the passenger's family name is "Jones". The app then returns the booking details.

For the sake of this tutorial, the sample data used are: A passenger whose family name is "Jones" and travels by the "Business" class. All other customers travel by "Economy" class.

Overall Structure of the App

This app contains:

- **ReceiveHTTPMessage** trigger: This trigger listens for an HTTP GET request containing the family name of the passenger requesting flight booking details. After it receives a request, it triggers the flow attached to the trigger.
- **FlightBookings** flow: This flow is attached to the **ReceiveHTTPMessage** trigger. This flow handles the business logic of the app. In this flow, you must configure a **LogMessage** activity to log a custom message when a request is received successfully. The **LogMessage** activity has two success branches:
 - The first branch accepts requests with any family name and uses a condition to check if the family name in the request is "Jones". It runs a **Return** activity to return the information of a flight booked in "Business" class for Jones.
 - The second branch runs when the first branch runs as false (that is, the family name is not "Jones"). It runs a **Return1** activity to return the information of a flight booked in "Economy" class if the family name is not "Jones".

i Note: Each branch must have its **Return** activity as the last activity in the branch.

Procedure

The high-level steps for creating and configuring the app in this tutorial are as follows:

Procedure

1. [Create a new app](#).
2. [Create a JSON schema to reuse it across your app](#). The JSON schema describes the format of the JSON data used in the tutorial. In this tutorial, we use a simple JSON schema for the request that the REST service receives and the response that the service sends back. You can specify the JSON schema directly or specify JSON data, which is converted to JSON schema automatically.
3. [Create a flow and add a REST trigger \(Receive HTTP Message\)](#).
4. [Map trigger output to flow input](#). This is the bridge between the trigger and the flow where the trigger passes on the request data to the flow input.
5. [Map flow output to trigger reply](#). This is the bridge between the flow output and the response that the trigger sends back to the HTTP request it received. After the flow has finished running, the output of the flow execution is passed back to the trigger by the **Return** activity. Hence, we map the flow output to the trigger reply. This mapping is done in the trigger configuration.
6. [Add a LogMessage Activity to the flow](#) and configure a message that the activity must log in to the logs for the app as soon as it receives a request.
7. [Add the first branch to check whether the passenger's last name is Jones](#) to return the information of a flight booked in "Business" class for Jones.
8. [Add a second branch to process any other passengers](#) and return the information of a flight booked in "Economy" class if the family name is not Jones.
9. [Validate the app](#) to make sure that there are no errors or warnings in any flows or activities.
10. [Build the App](#).

11. [Test the app.](#)

Step 1: Create an app

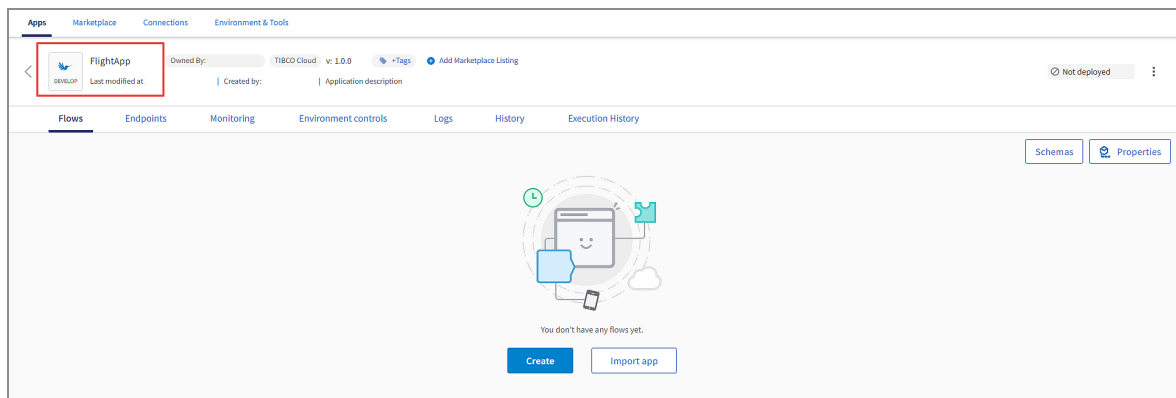
To create a Flogo app:

Procedure

1. Click **Apps**.
2. Click **Create/Import**. The **What do you want to build?** dialog opens.
3. To create a Flogo app:
 - Under **Quickstart > All app types;Apps**, click **Create a Flogo app**.
 - In the block that displays below your selection, click **Create Flogo app**.

A Flogo app is created with the default name in the `New_Flogo_App_<sequential_app_number>` format.

4. Click the default app name to make it editable. Change the app name to `FlightApp` and click anywhere outside the name to save the changes made to the name.



Step 2: Create a JSON schema

Procedure

1. Copy the following JSON sample to use in your app:

```
{
  "Class" : "string",
```

```

"Cost" : 0,
"DepartureDate" : "2017-05-27",
"DeparturePoint" : "string",
"Destination" : "string",
"FirstName" : "string",
"Id" : 0,
"LastName" : "string"
}

```

Note: Ensure that you use straight quotes when entering the schema elements and values.

2. On the **Apps** page, in the **Flows** section, click **Schemas**.
3. In the **Schemas** dialog that opens, click **Schema** to add a JSON schema.
4. Name your schema as `FlightResponse` and paste the copied schema into the text editor. Alternatively, if you enter JSON data in the editor, the JSON data is automatically converted to JSON schema.

The screenshot shows the 'Schemas' dialog box with the following elements:

- Search schema:** A search bar with a magnifying glass icon.
- + Schema:** A button to add a new schema.
- Notification:** A blue bar with an information icon stating 'JSON data will automatically be converted to JSON Schema'.
- Schema Name:** A text input field containing 'FlightResponse'.
- Schema Type:** A dropdown menu set to 'Json Schema'.
- Text Editor:** A code editor containing the JSON schema code:


```

1 {
2   "Class" : "string", "Cost" : 0,
3   "DepartureDate" : "2017-05-27",
4   "DeparturePoint" : "string",
5   "Destination" : "string",
6   "FirstName" : "string",
7   "Id" : 0,
8   "LastName" : "string"
9 }

```
- Buttons:** 'Cancel' and 'Save' buttons at the bottom right.

5. Click **Save**.

Step 3: Create a flow and add a REST trigger

Every app must have at least one flow and, in most cases, a trigger that initiates the flow. Create a flow with the REST trigger. The **ReceiveHTTPMessage** REST trigger listens for an incoming REST request that contains the details of a passenger who wants to book a flight. Specify the fields for the request in the REST trigger in JSON schema format.

To create a flow:

Procedure

1. On the **Flows** page, click **Create**.

The **Add triggers and flows** dialog is displayed. The **Flow** option is selected by default.

2. In the **Flow details** section, provide the following details and click **Create**:

Name: FlightBookings.

Description: Optional description of the flow.

3. On the FlightBookings flow page, click the **Triggers** icon. The trigger palette opens.
4. From the **Triggers** palette, drag the **Receive HTTP Message** trigger to the **Triggers** area on the left. The Configure trigger: ReceiveHTTPMessage dialog opens.

Configure trigger: Receive HTTP Message Step 1 of 1

Step 1

9999

Configure Using API Specs i

True False

Method i 1

GET

Resource Path i 2

/flightbookings

Response Schema i FlightResponse [Change](#) 3

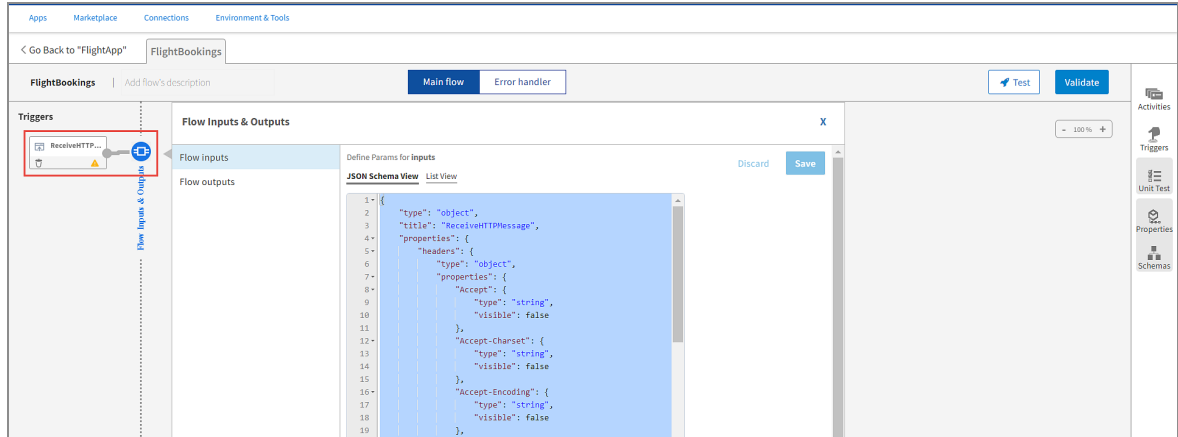
← Back Cancel Continue ✓

- a. Select **GET** as the **Method**.
 - b. Enter `/flightbookings` in the **Resource path** box.
 - c. Enable the **Use App Level Schema** toggle next to **Response Schema** to open the **Schemas** dialog and select the **FlightResponse** schema you defined earlier. The selected schema is automatically displayed in the **Response Schema** box.
 - d. Click **Continue**.
5. Next, select **Copy Schema** when prompted.

The schema that you entered when creating the trigger is automatically copied to the **Flow Inputs & Outputs** tab to match the input and output of the trigger.

A new flow is created and attached to a REST trigger.

Your flow must look similar to the following image:



6. Lastly, close the **Flow Inputs & Outputs** tab.

Step 4: Map trigger output to flow input

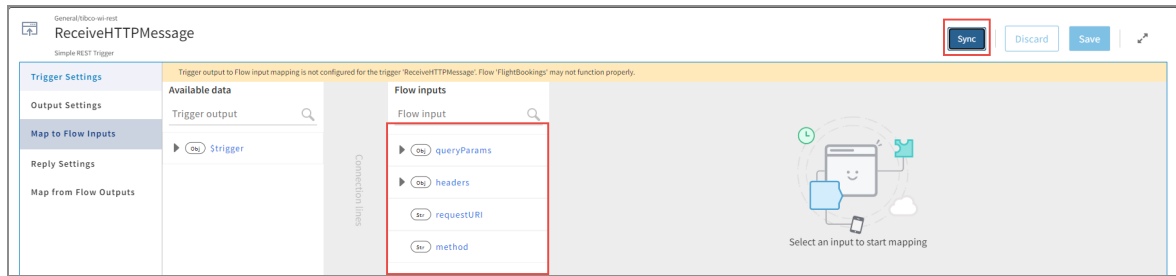
When REST trigger receives a request from a passenger (a REST request), the data from the request is produced by the **ReceiveHTTPMessage** REST trigger. For the request to be processed, this output must be used by the flow in the form of flow input. Hence, you must map the trigger output to the flow input.

To do this:

Procedure

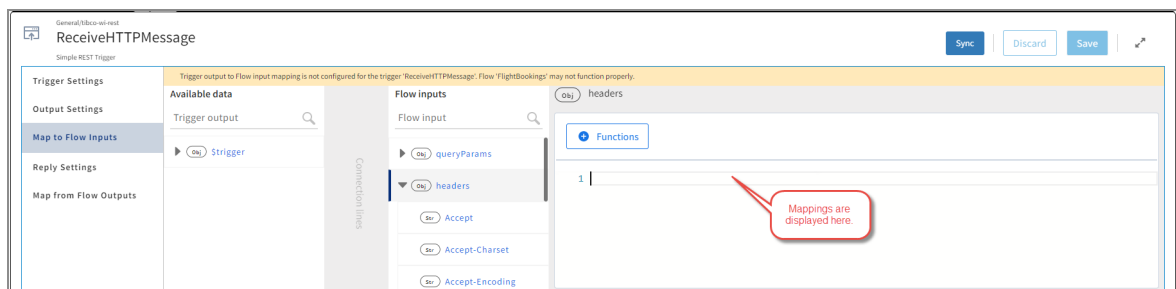
1. Click the **REST Trigger** icon to open its configuration dialog.
In the **Configuration** dialog, multiple tabs are displayed in a column on the left. **Trigger Settings** is selected by default.
2. Click **Output Settings** to add the query parameter.
3. Click **Add row** to add a query parameter.
4. In the new row in the **Query Parameters** table, enter the value of **ParameterName** as `lastname` and click **Save** in the same row (in the **Actions** column).
5. To start the mapping, click the **Map to Flow Inputs** tab and configure the mapping of the trigger output. On the **Map to Flow Inputs** tab, the **Available data** and **Flow inputs** panes are displayed. **Flow inputs** is the list of flow inputs that can be mapped to the trigger outputs in the **Available data** pane. Only headers are displayed in the flow inputs. The new query parameter is not visible yet.
6. **Save** the trigger configuration and click **Sync** to display the new values. Now,

queryParams must open in the **Flow inputs** column.



7. In the **Flow inputs** column, click **headers**.

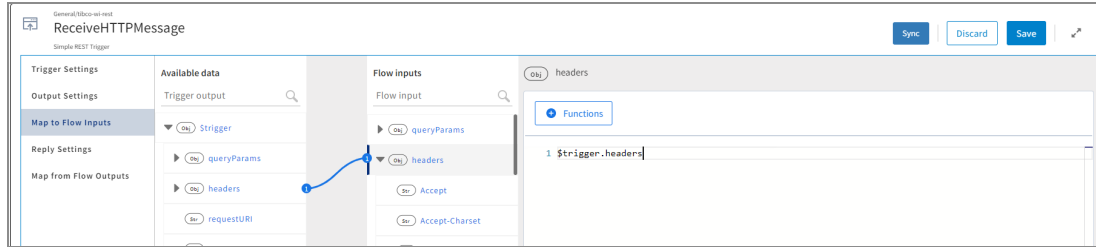
The **headers** text editor on the right of **Flow inputs** is initially empty.



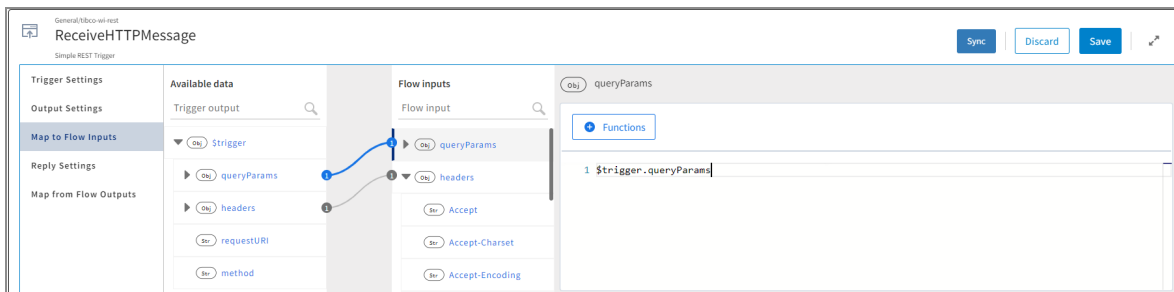
8. To map the trigger output headers to the flow input header:

- a. Expand **\$trigger** to see all the trigger outputs available. This displays the headers and body.
- b. Drag **headers** from the **Available data** pane to **headers** in the **Flow inputs** pane. Alternatively, click **headers** from the **Flow inputs** pane, drag **headers** from the **Available data** pane into the text editor.

The text editor must now display **\$trigger.headers** and a connection line between the two panes. This indicates that you have successfully mapped the trigger output headers to the flow input header. The numbers at the end of the connection line indicate the total number of mappings for the selected element.



- To map the flow input, in the **Flow inputs** column, click **queryParams**. The data mapper view is the same as the one while mapping headers. The **queryParams** text editor is initially empty. Drag **queryParams** from the **Available data** pane and drop it on **queryParams** in the **Flow inputs** pane. The text editor must now display **\$trigger.queryParams**. This indicates that you have successfully mapped the trigger output **queryParams** to the flow input **queryParams**.



- To save your progress, click **Save**.
This completes the mapping of flow inputs.

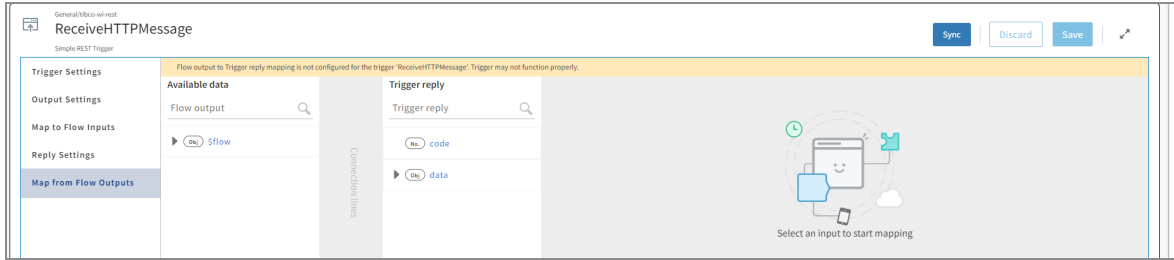
Step 5: Map the flow output to trigger reply

When the execution of the flow is completed, the output must be sent back to the trigger for the trigger to send a reply to the REST request initiator. Hence, the flow output data must be mapped to the trigger reply, which then returns the result of the flow execution to the REST request initiator.

To map the flow output to the trigger reply:

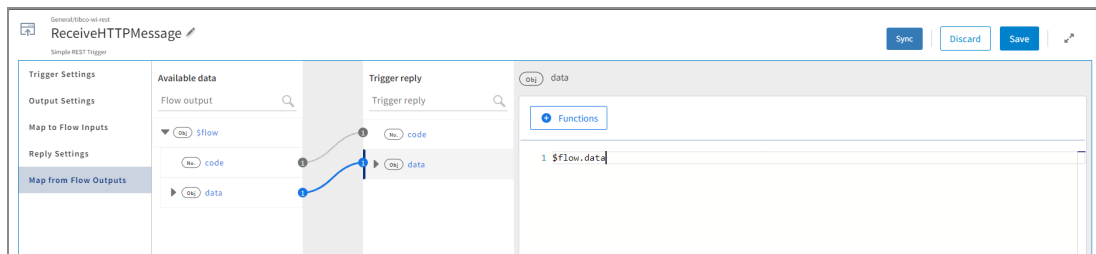
Procedure

- In the left pane, click the **Map from Flow Outputs** tab to configure the mapping of the trigger reply. The **Available data** and **Trigger reply** panes are displayed. You can map the following trigger replies to the flow outputs - **code** and **data**.



2. In the **Map from Flow Outputs** section:

- a. The **Available data** pane displays the data available for the mapping. **\$flow** is displayed in this pane. To see all the flow outputs available for the mapping, expand **\$flow**. This displays **code** and **data**.
- b. Drag **code** from **Available data** and drop it on **code** in the **Trigger reply** pane. **\$flow.code** is displayed in the **code** text editor. You have successfully mapped the **code** in **Trigger reply** to the **code** in **Available data**.
- c. Repeat the same steps to map **data** from **Trigger reply** with **data** from **Available data**.



Note: You can expand **data** in both the **Trigger reply** pane and the **Available data** pane to see the tree structure of the data you have defined in the schema.

3. Click **Save** and close the trigger dialog.

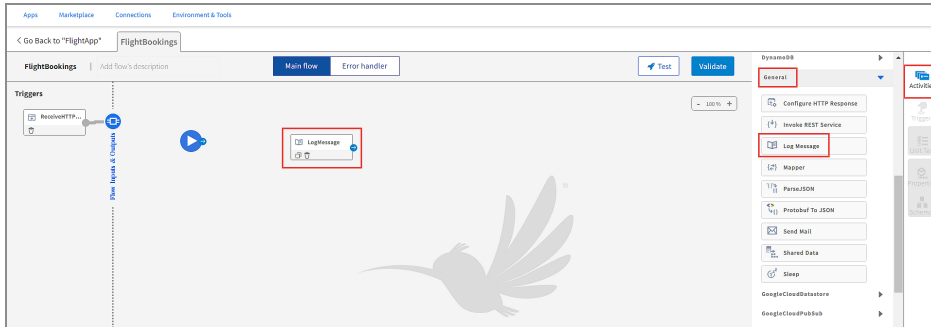
Step 6: Add a Log Message Activity to the flow

The flow uses the **LogMessage** activity to log an entry in the app logs when the trigger receives a request from the passenger that reaches the trigger in the form of a REST request.

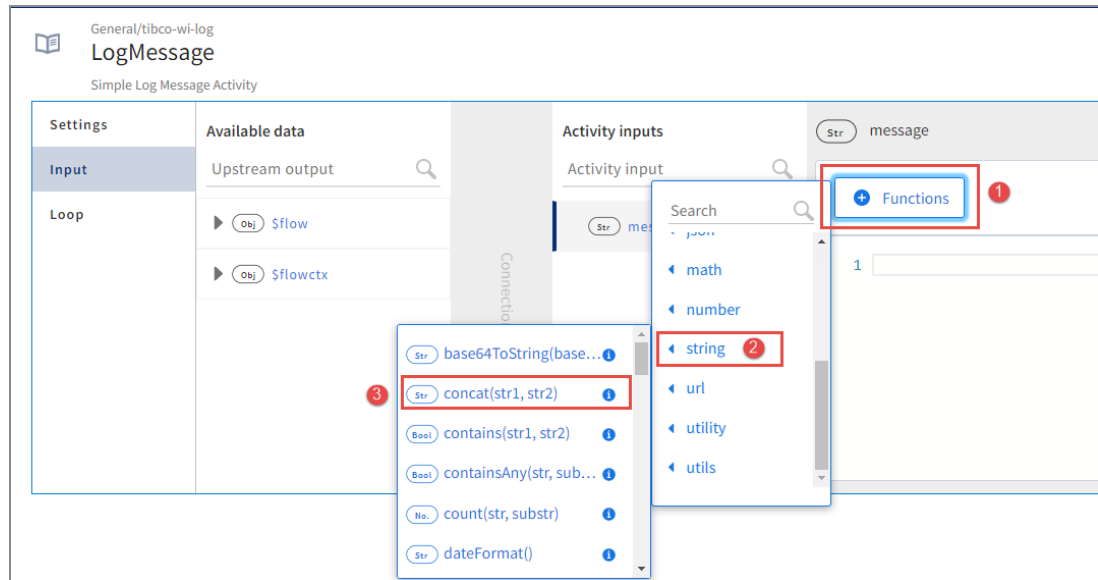
To add a **LogMessage** activity:

Procedure

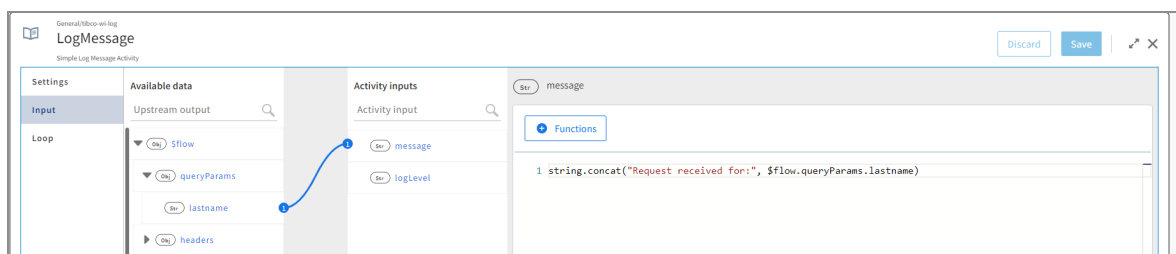
1. On the FLightBookings flow page, click **Activities**, the activities palette opens.
2. In the **Activities** palette, under **General** tab, select **Log Message** and drag it to the activities area.



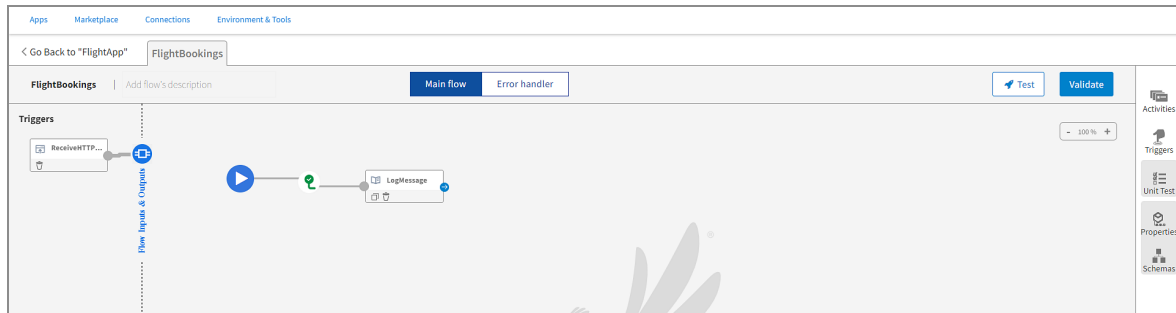
3. Drag a connection line from **StartActivity** to the **LogMessage** activity that you have created.
4. Now, to configure the **LogMessage** activity with a message to log when it receives an incoming request from the **ReceiveHTTPMessage** trigger:
 - a. Click the **LogMessage** activity to open the configurations dialog.
 - b. Click the **Input** tab. The **Available data** and **Activity inputs** columns are displayed on the right side of the **LogMessage** activity tabs.
 - c. Click the **message** to open the mapper to the right. Configure a message to be logged by the **LogMessage** activity when the input from the request that the trigger received is passed on to and received by the flow.
 - d. To configure the message, click **Functions**, and expand the **string**. Click **concat (str, str2)** to add the function to the **message** box.



- e. Select **str** in the box and replace it by entering "**Request received for:** " (include the quotes too): `string.concat("Request received for: ", str2)`.
5. Replace **str2** with the family name of the passenger who booked the flight. (The family name of the passenger is passed on from the trigger to the flow. We had mapped this trigger output to flow input previously. Hence it is now available for mapping under **\$flow** in **Available data**.)
 - a. In the **Available data** pane, expand **\$flow** and expand **queryParams**.
 - b. Drag **lastname** and drop it in place of **str2**.
 - c. Click **Save**.



6. Close the **LogMessage** dialog.
Your flow must now look like this:

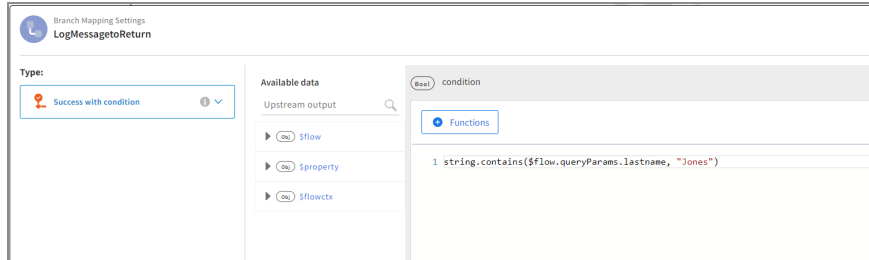


Step 7: Add the first Return Activity branch

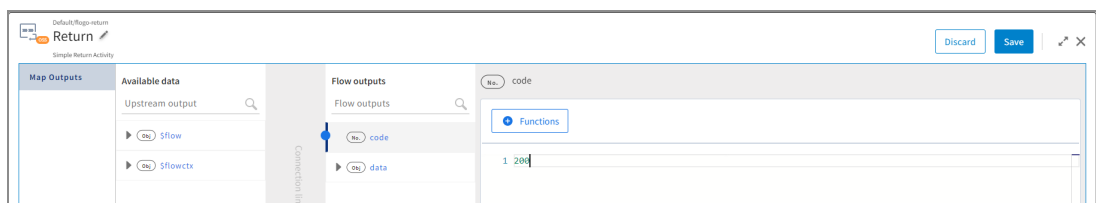
To add a **Return** activity and the branch to configure its condition to look for the family name "Jones":

Procedure

1. From the **Activities** palette, drag **Return** activity available under **Default** category to the activity area.
2. Now, to configure a connection line between a **LogMessage** activity to the **Return** activity. Configure the branch with a condition to read the family name of the passenger.
3. Drag a highlighted arrow from the **LogMessage** activity to the **Return** activity.
4. Hover over and click the branch label on the connection line you just created. The configuration window for branch condition opens.
5. In the **Branch Mapping Settings** dialog that opens, select the **Success with condition** branch condition.
 - a. Click **Functions**. Select the **string>>contains(str1, str2)**. The selected function is added to the **condition** text editor.
 - b. Configure **str1** in the expression to take the value of the family name that the user enters. In the **Available data**, expand **\$flow > queryParams**. Drag **lastname** to **str1**. This family name is the name entered by the user in the search query.
 - c. Replace **str2** in the condition by manually typing "Jones".

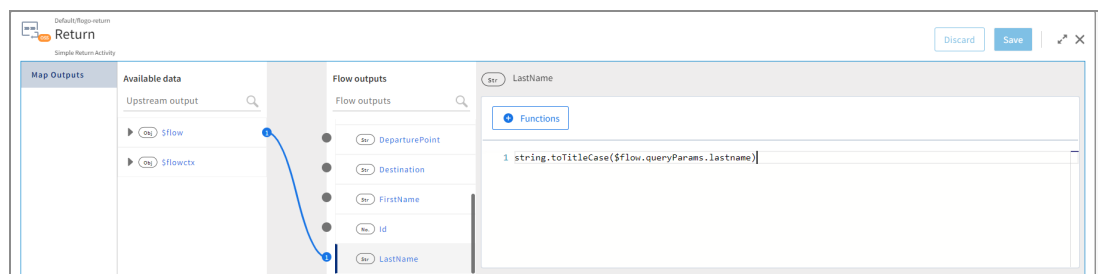


- d. Click **Save**. This branch runs when the name entered as a query parameter is Jones.
6. Now, Configure the **Return** activity for the branch to produce the flow results if this branch runs (when the passenger's family name is anything but Jones):
 - a. Click the return activity to open the configuration dialog.
 - b. Click **code** under **Flow outputs** to open the mapper and type **200** in the **code** box, which is the HTTP success code.

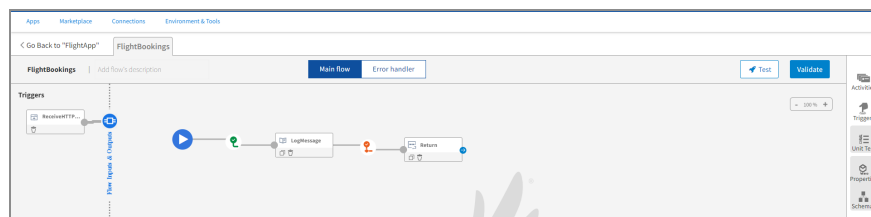


- c. Expand the next flow output **data**. All the different elements under **data** that are returned by this activity are displayed. Assign a value to each field under **data**.
- d. Start by clicking **Class** under **data** and type "Business" as Jones is traveling by "Business" class.
- e. Click **Cost** to type a number of your choice. You can also use a function to randomize the value. To do so, in the **Functions** section, expand the **number** category and click **random()**. Enter **5000** as an input parameter to the **random()** function.
- f. Click **DepartureDate** to enter the departure date in any format of your choice. Use quotation marks as the date needs to be specified as a string. For example, "01/01/21" or "January 1, 2021" are valid values.
- g. Click **DeparturePoint** to enter the departure airport name of your choice. Use quotation marks as the departure point needs to be specified as a string. For

- example, "LAX" or "LHR" are valid values.
- Click **Destination** to enter a string for this field. For example, "Paris" or "JFK" are valid values.
 - Click **FirstName** to enter the first name associated with the family name Jones. For example, "Brian" or "Paul" are valid values.
 - Click **Id** to enter a number of your choice. You can also use a function to randomize the value. To do so, in the **Functions** section, expand the **number** category and click **random()**. Enter **999999** as an input parameter to the **random()** function.
 - Click **LastName** to map this field to the query parameter **lastname**. Before doing so, we can use a string function to capitalize the family name that is returned by our app. To do so, under **Functions**, expand the **string** and click **toTitleCase(str)**. Once **string.toTitleCase(str)** is added to your box, select **str** to replace it with the query parameter. Expand **\$flow** and then **queryParams** under **Available data**. Drag **lastname** and drop it in place of **str**. The text editor must look like this:



- Click **Save** and then close the **Return Activity Configuration** dialog. Your flow must look like this:




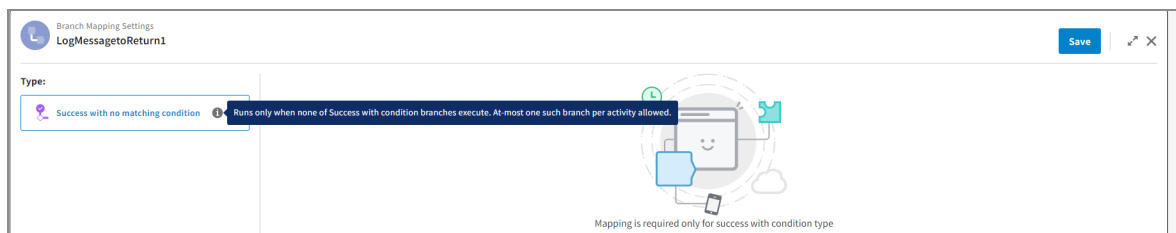
Step 8: Add a second Return Activity branch

The second branch that you add from the **LogMessage** activity runs when the success condition of the first branch is not matched. If the passenger's family name is not "Jones", the passenger's seat is in "Economy" class.

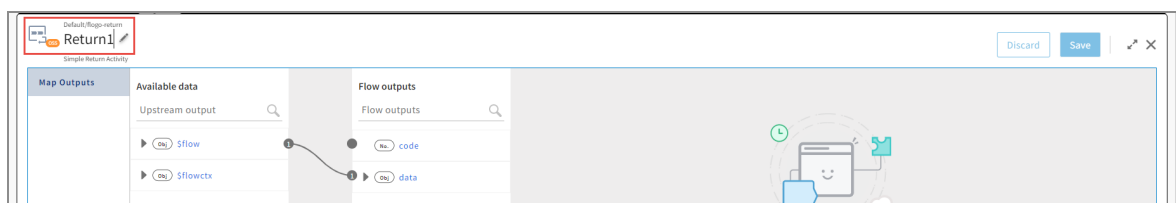
To add a second branch from the **LogMessage** activity:

Procedure

1. Duplicate the **Return** activity from the first branch instead of manually adding another **Return** activity. You can copy the activity by clicking . The copied activity is displayed next to your original **Return** activity:
2. Click the **CopyOfReturn** activity to configure the response this branch return.
3. First, to create a connection between the **LogMessage** activity and the **Return1** activity, hover over to the LogMessage activity, you see that an arrow highlighted. Drag the arrow to the **Return** activity.
4. Select the **Success with no matching condition** branch condition. If the conditions of all the other **Success with condition** branches are not true, this branch is run. This means, if the family name entered as a query parameter is not Jones, this second branch is run.

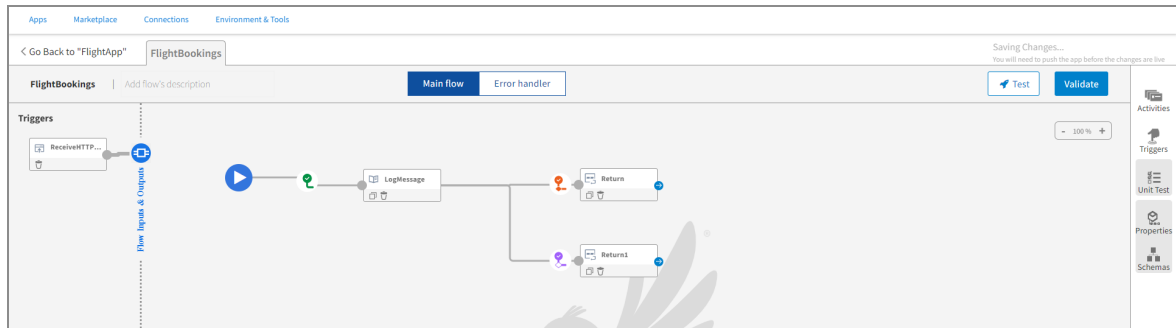


5. Now, in the configuration window click the name of the activity to make it editable and rename the activity.



6. In the **Flow outputs** section, expand **data**, select **Class**, and type "Economy" as this branch must return "Economy" class bookings.
7. Click **Save** and close the dialog.

Your flow must look like this:



Step 9: Validate the app

Your app is now ready. Before you push the app to the Cloud, validate all the flows for any errors or warnings. To do so, click **Validate**. Flogo validates each flow and activity within the flow. For any errors or warnings, you see the respective icons next to the flow name or activity tab, which contains the error or warning.

On successful validation, you get the following message:

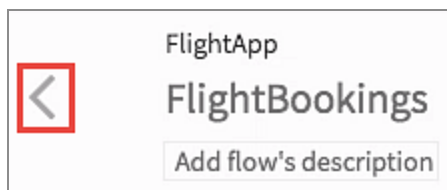


Step 10: Build the App

Your app is now ready to be built. You can build a Flogo app using a s an executable file.

Procedure

1. Click the left arrow next to the flow name to open the FlightApp page.



2. Click **Build**.
3. Select your target platform from the **Build** drop-down list. Select **Windows/amd64** on Windows, **Darwin/amd64** on Macintosh or **Linux/amd64**, or **Linux/86** on Linux from the list.

You see a build log with the progress of the build command. When the build completes, you see an executable file called `FlightApp-darwin_<processor>` in your `/Downloads` directory.

Step 11: Test the app

Now that the app has been built successfully, you run the app. Once it runs successfully, you can test your API in a REST client.

On Macintosh and Linux platforms:

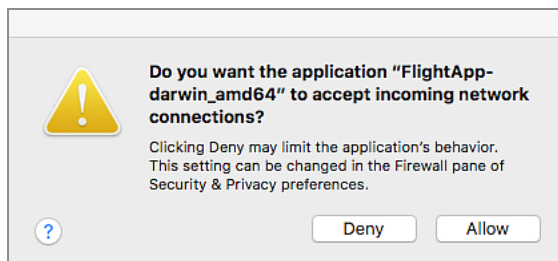
To test the app:

Procedure

1. Open a terminal and change the directory to the location of `FlightApp-darwin_amd64`, `FlightApp-linux_amd64`, or `FlightApp-linux_86` file depending on your platform.
2. Run the following commands:
 - `chmod +x <FlightApp-darwin_amd64>`
 - `./FlightApp-darwin_amd64`

Note: In the commands, use the file name specific to your platform - `FlightApp-linux_amd64` or `FlightApp-linux_86` in the case of Linux.

3. Click **Allow** in the following dialog:



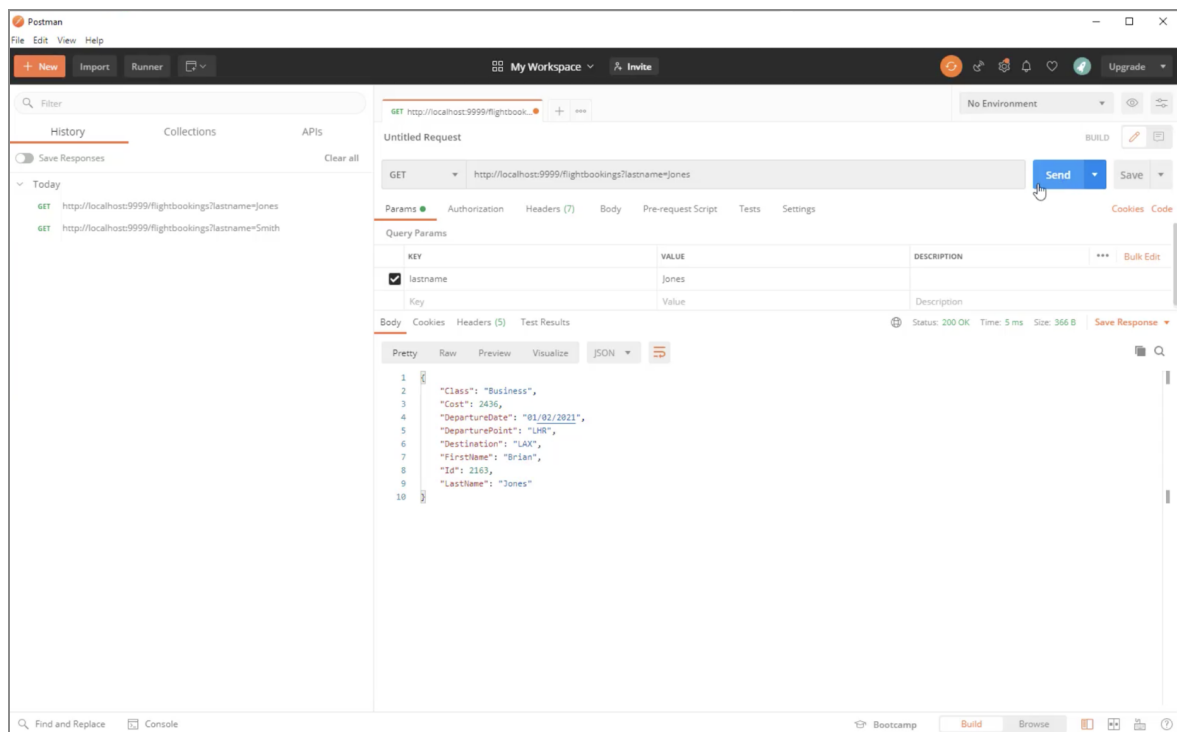
The following messages are displayed in the console:

```

TIBCO Flogo® Runtime - 2.11.0 (Powered by Project Flogo™ - v1.2.0)
TIBCO Flogo® connector for General - 1.2.0.455
Starting TIBCO Flogo® Runtime
2021-01-07T11:07:11.587Z WARN [Flogo] - unable to create child logger named: ReceiveHTTPMessage - unable to create child logger
2021-01-07T11:07:11.587Z INFO [general-trigger-rest] - Name: ReceiveHTTPMessage, Port: 9999
2021-01-07T11:07:11.587Z INFO [Flogo.engine] - Starting app [ FlightApp ] with version [ 1.0.0 ]
2021-01-07T11:07:11.587Z INFO [Flogo.engine] - Engine Starting...
2021-01-07T11:07:11.587Z INFO [Flogo.engine] - Starting Services...
2021-01-07T11:07:11.590Z INFO [Flogo] - ActionRunner Service: Started
2021-01-07T11:07:11.590Z INFO [Flogo.engine] - Started Services
2021-01-07T11:07:11.590Z INFO [Flogo.engine] - Starting Application...
2021-01-07T11:07:11.590Z INFO [Flogo] - Starting Triggers...
2021-01-07T11:07:11.590Z INFO [general-trigger-rest] - Starting ReceiveHTTPMessage...
2021-01-07T11:07:11.628Z INFO [general-trigger-rest] - Started ReceiveHTTPMessage
2021-01-07T11:07:11.628Z INFO [Flogo] - Trigger [ ReceiveHTTPMessage ]: Started
2021-01-07T11:07:11.628Z INFO [Flogo] - Triggers Started
2021-01-07T11:07:11.628Z INFO [Flogo.engine] - Application Started
2021-01-07T11:07:11.628Z INFO [Flogo.engine] - Engine Started
Runtime started in 68.5802ms

```

4. Make a note of the port number 9999 and path /flightbookings in the logs.
5. You can test your API in a REST client such as Postman by entering the port number 9999, path /flightbookings, and query parameter lastname. For example, `http://localhost:9999/flightbookings?lastname=jones`.



In the above example, note that since the query parameter sent has the family name as "Jones", the **Class** in the response has been automatically set to "Business" class.

6. Go back to your terminal. You must see the logs you configured with the **Log** activity.

```
2021-01-07T11:08:14.459Z INFO [general-trigger-rest] - REST Trigger: Received request for id 'ReceiveHTTPMessage'  
2021-01-07T11:08:14.459Z INFO [FlightBookings/LogMessage] - Request Recieved for: Jones  
2021-01-07T11:08:14.459Z INFO [flogo.flow] - Instance [89a00c06f94e0e9b9dcd5918d363ce7e] Done
```

App Development

Flogo Enterprise offers a wizard-driven approach to app development. You can create apps in Flogo Enterprise using only a browser. It is powered by Project Flogo™, a lightweight integration engine.

For more information about Project Flogo™, navigate to the [Project Flogo](#) website.

Creating and Managing a Flogo App Using the UI

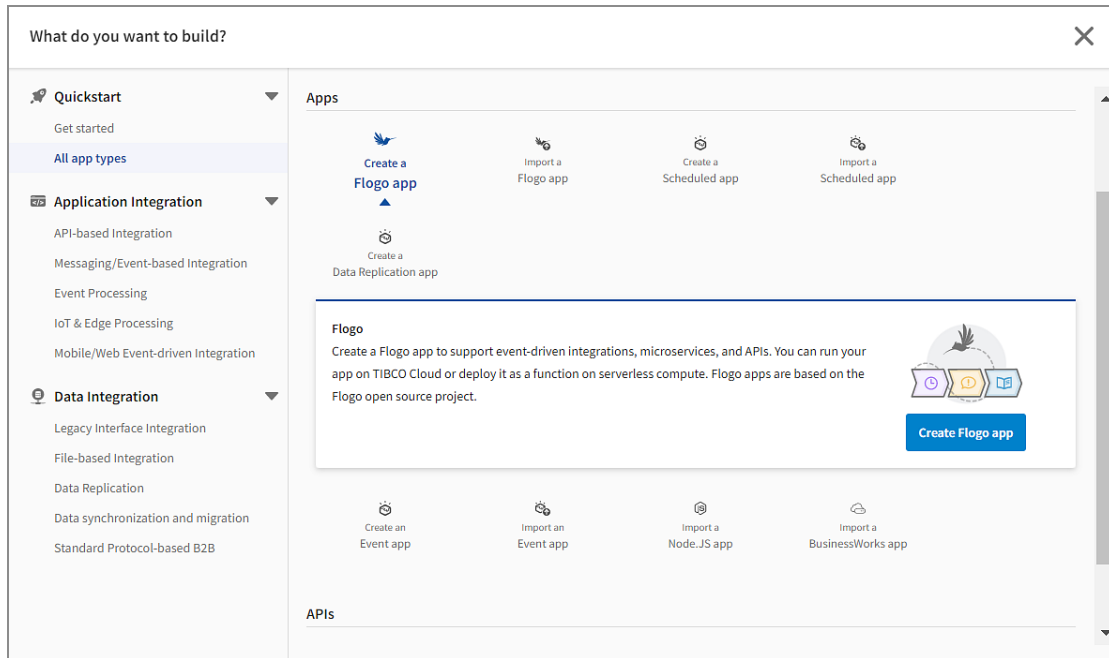
This section describes how to create and manage Flogo® apps.

Creating an App

You can create a Flogo® app from the **Apps** page.

Procedure

1. Log in to **TIBCO Cloud™ Integration**.
2. On the **Apps** page, click **Create/Import**.
The **What do you want to build?** dialog is displayed.



3. To create a Flogo app:

- Under **Quickstart > All app types > Apps**, click **Create a Flogo app**.
- On the left, select a category that identifies the type of integration you need. On the right, click **Create a Flogo app**. In the block that displays below your selection, click **Create Flogo app**.

The app is created and the **App Details Page** is displayed for the new app. By default, the app is named in sequential order in the format `New_Flogo_App_<sequential_number>`. For example, if you created three apps without renaming them, then the first one has a default name of `New_Flogo_App_1`, the second one is called `New_Flogo_App_2` and, the third one is called `New_Flogo_App_3`. The version of a newly created app is 1.0.0 and is displayed as `v: 1.0.0` beside the name of the app. You can edit the version of the app. For more information, refer to [Editing the Version of an App](#).

4. Edit the app name to a meaningful string. To do so, click anywhere within the app name and edit it, then click anywhere outside the text box to persist your change.

i Note: The app name must not contain any spaces. It must start with a letter or underscore. The app name can contain letters, digits, periods, dashes, and underscores.

5. Click **Create**.

Result

The **Add triggers and flows** dialog is displayed. You can now create one or more flows for the app. See the [Creating a Flow](#) topic and its subtopics for details on creating a flow. When the app is created, the following files are generated in the <FLOGO_HOME>/apps/<app>/ directory:

- `flogo.json`: contains the app itself.
- `manifest.json`: contains the manifest details such as the endpoints, memory resource details. The `manifest.json` file is automatically updated whenever you modify the app.

Creating an App from a Saved Specification

If you have an existing specification saved in either the TIBCO Cloud™ Integration - API Modeler or on your local machine, you can use the specification to create a Flogo app. Currently, Flogo Enterprise supports app creation using a Swagger Specification 2.0, OpenAPI Specification 3.0, and GraphQL Schema.

The specification must exist before creating the Flogo app.

For more information on creating an app using a specification, see the following topics:

- [Creating a New App Using an OpenAPI Specification](#)
- [Creating a New App Using GraphQL Schema](#)

Also, see the appropriate topics under the [Building APIs](#) section for information on how to create a Flogo app using the specification.

Creating a New App Using an OpenAPI Specification

You can create a Flogo app by uploading an API specification file or importing an existing file stored in the API Modeler. You can simply drag the specification file to the UI or

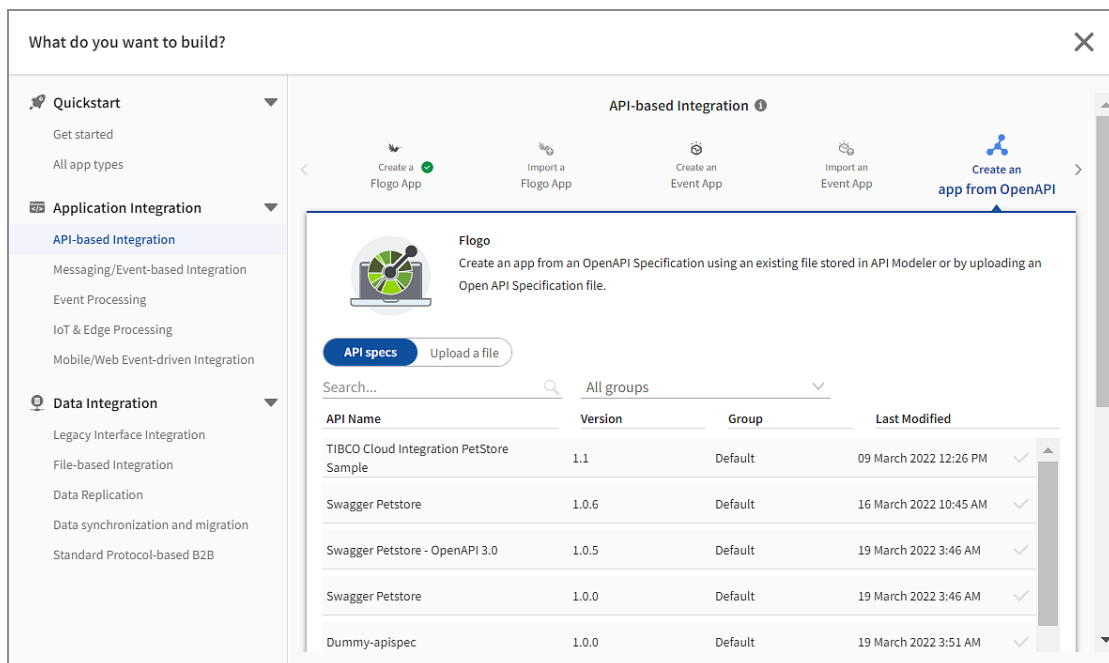
navigate to it.

Before you begin

For requirements and considerations, see [Using an OpenAPI Specification](#). For details about the OpenAPI specification, see [OpenAPI Specification](#).

Procedure


1. Log in to TIBCO Cloud™ Integration.
2. On the **Apps** page, select **Create/Import**. The **What do you want to build?** dialog is displayed.



3. In the block that displays below your selection, select one of the following options:
 - Create a flow using an API specification that exists in the TIBCO Cloud™ Integration-API Modeler. To do this, on the **API Specs** tab, select the specification that you want to use.
 - Use an API specification saved locally on your computer by uploading it to Flugo Enterprise. To do this, click the **Upload file** tab. Browse to the saved API specification on your local machine or drag your saved API specification into the dialog.
4. Click **Import OpenAPI spec**.

The app is created and the **App Details** Page is displayed for the new app. Your app is running but has zero instances. To start and scale your app, see **Starting, Stopping, and Scaling** apps.

i Note: While creating an app with a **REST Trigger - ReceiveHTTPMessage**, If the API specification changes, you can merge the changes into an existing app by either uploading the updated specification file again. Or, click **Refresh** beside the **Browse** tab under **API Spec** in the REST trigger configuration window.

To **Refresh** the API spec file, you must make the **API spec** editable by clicking  in the trigger configuration window.

For any flows that are already implemented, adding, or deleting any method in TCAM does not impact the flows in the app.

If there are any changes made in the *Spec file* in **TCAM** UI while your trigger configuration window is also open in Flogo, **Refresh** does not appear in the trigger configuration window. Close and reopen the trigger configuration window.

Creating a New App Using GraphQL Schema

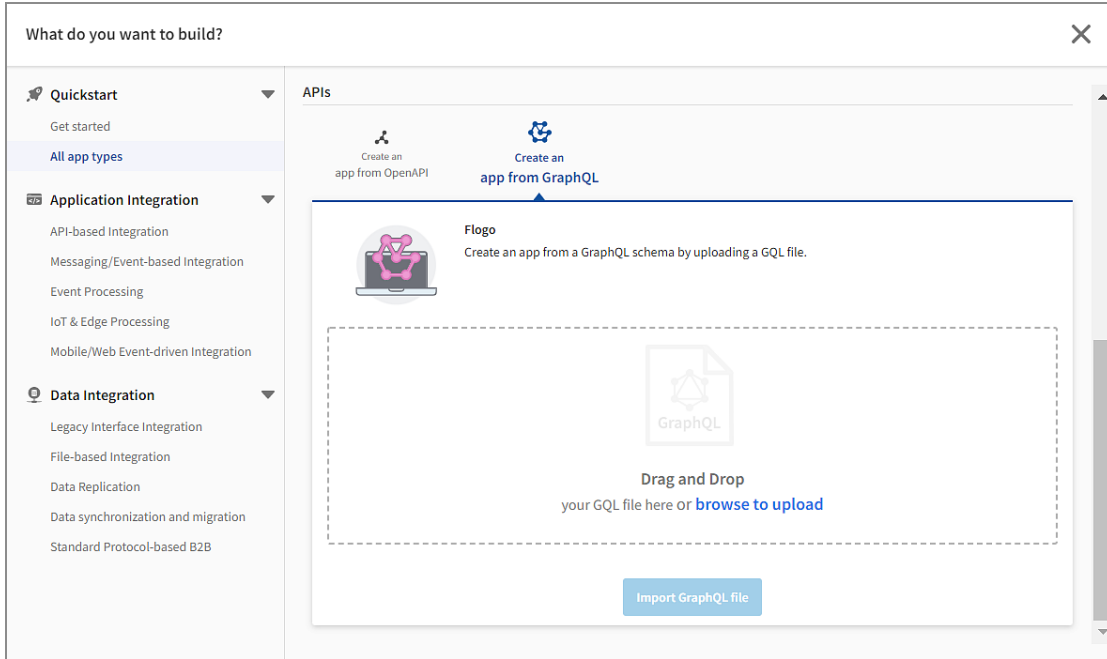
You can create GraphQL triggers by dragging and dropping your GraphQL schema file into the UI or by navigating to the file.

Before you begin

For requirements and considerations, see [Using GraphQL Schema](#).

Procedure

1. Log in to TIBCO Cloud™ Integration.
2. On the **Apps** page, select **Create/Import**. The **What do you want to build?** dialog is displayed.



3. In the block that displays below your selection, browse to the GraphQL schema file or drag the file to the dialog.
4. Click **Import GraphQL file**.

What to do next

The app is created and the **App Details Page** is displayed for the new app. Your app is running but has zero instances.

Validating your App

After you have created the flows in your app, you must validate the app before you push it to the cloud.

To validate your app, click **Validate** on the app details page. This validates each flow and activity. If a flow or activity has an error, it displays an error or warning icon on the top-right corner of the flow or activity.

Important Considerations

- When you open any flow for the first time or switch to a flow for the first time, the

validation is auto-triggered for that instance only. After that, for any change in the canvas, you must do a manual validation check by clicking **Validate**.


- If a flow is already a part of open tabs and no unsaved changes exist for that flow, while switching to that flow from any other flow, validation is not triggered.
- After a flow's validation is completed, the validation details are cached and remains present till you move out of the flow to the Flow List page or you refresh the page.
- If you add or change triggers and activities in a flow or any change in canvas, no validation is triggered. To observe the latest validation, click **Validate**.
- If a new tab is opened, validation is triggered.

If a sub-flow is appended, validation is triggered when the subflow is clicked. If you call a subflow already present on the Flows tab, validation is not triggered.

For more information, see [Viewing Errors and Warnings](#).

Editing an App

You can edit your Flogo app from the **Apps** page. Click any app to edit flows, triggers, and so on.

 **Warning:** Editing the same app in two browser tabs is not supported.

When you modify the app, the `flogo.json` and `manifest.json` files in `<FLOGO_HOME>/apps/<app>/` are updated automatically. For example, if you add a flow and add a trigger to it, the `flogo.json` and `manifest.json` files are updated automatically to include the details of the flow and the trigger.

Auto-Upgrade of Activities, Triggers, and Connections

Flogo supports the automatic upgrade of activities, triggers, and connections. Thus, you can view the newly added fields without exporting and reimporting apps. In case of updates to the activities and triggers, open the app to upgrade it automatically. To view updates for connections, you must open the connection from the **Connections** page.

Considerations for Auto-Upgrade

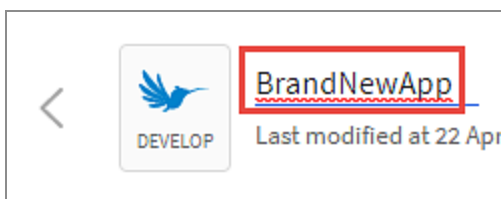
- Contributions from activities, triggers, and connections are auto-upgraded only if the contribution version is also updated.
- Field values that you enter previously are auto-populated after the upgrade. The new fields have default values, if the field is mandatory. You need not reconfigure the app.
- When you open the connection from the **Connections** page and cancel or close the dialog, the newly added connection properties appear in the **Properties** dialog. The same happens on the **Environment Control** tab of the app along with the existing connection properties. If you save or log in to the connector (wherever applicable), then only the connection properties used in the connection appear in the **Properties** dialog and on the **Environment Control** tab of the app.
- New connection-related fields at the Activity or trigger level are populated only when you save or log in to the existing connection.

Renaming an App

To rename an existing app:

Procedure

1. Open the app details page by clicking the app name.
2. Click anywhere in the app name and edit the name.



3. Click away from the app name to see your changes.

Editing the Version of an App

When you create an app, the default version of the app is 1.0.0. You can edit the version of an app.

The format of a valid app version is:

```
xxx.xxx.xxx
```

i Note: Alphabets or special characters are not allowed in an app version.

Some examples of valid app versions are:

```
1.1.1  
11.22.13  
111.222.333
```

Procedure

1. Open the app details page.

Besides the name of the app, the version of the app is displayed as follows:

```
New_Flogo_App_<sequential_number> v: 1.0.0
```

For a newly created app, the version is 1.0.0.

2. To edit the version of the app, click the version number and specify the new version.

The new version of the app is reflected everywhere. For example, in runtime logs.

Using App Tags

You can use app tags to provide additional information and organize your apps. For example, you use it to specify whether it is a REST app, or whether it is running in Kubernetes. One or more tags can be added to an app. You can view and filter the tags from the list of apps on the **Apps** page. The tags are preserved after exporting a Flogo app.

Adding Tags

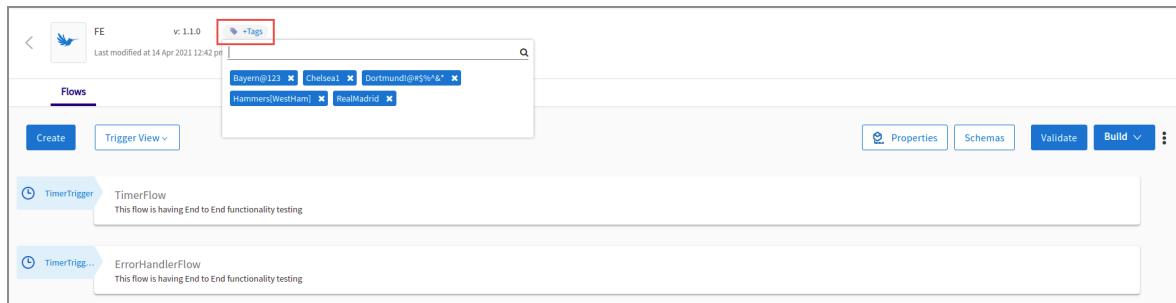
To add or change tags in an app:

Procedure

1. Click the **Apps** tab.
2. Click the app that you want to modify. The **App Details** page opens.

3. Click **Tags** or **+Tags** (if the app has no tags). Tags that have already been applied to this app are shown.

For example, the following screenshot shows that the app **FE** has no tags. Click **+Tags** to view the tags in the organization.



- To add a tag, enter a name in the search control, then click **Create New**. Tags are case-sensitive.
 - If you enter text in the search box, all matching tags in your organization are shown. This search is case-insensitive. Click a tag to add it.
 - Click close (✕) next to a tag to remove it from the app.
4. Click outside the dialog to save the changes.

The same set of tags is used across your organization. A tag is deleted from your organization when it is removed from the last app using it.

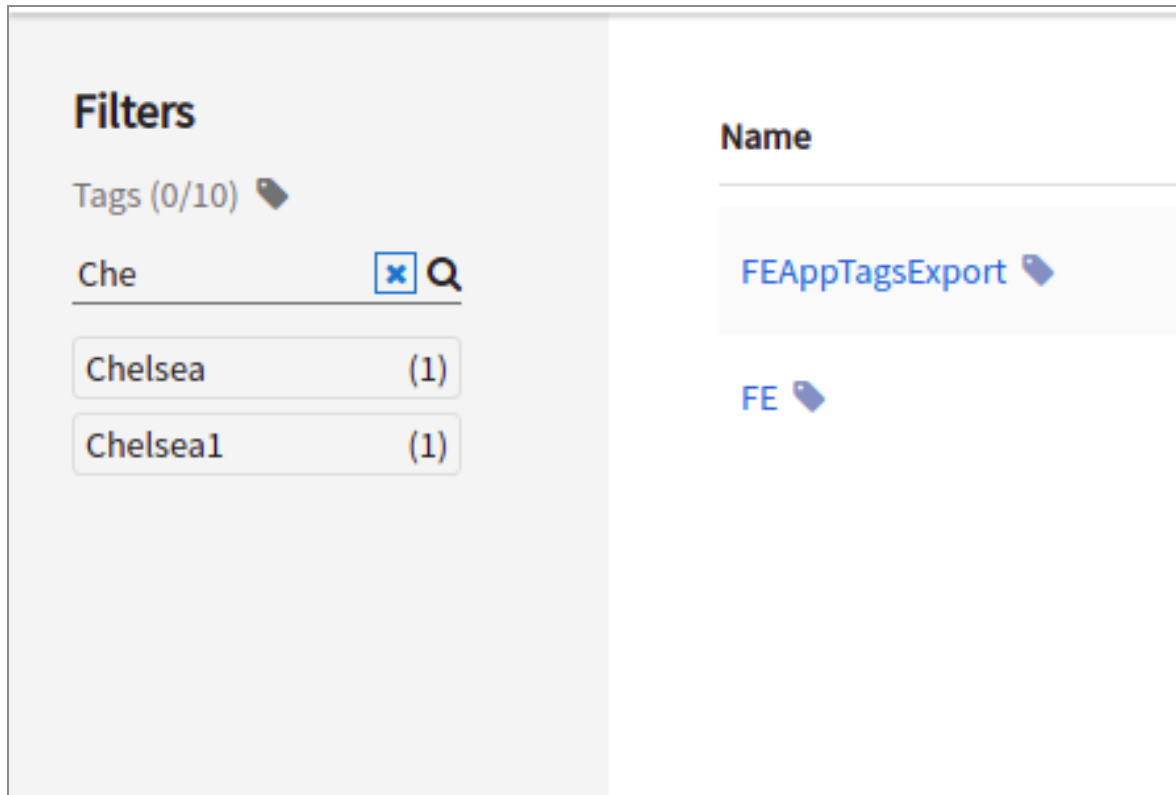
Filtering Tags

To filter the apps list on the **Apps** page:


Procedure

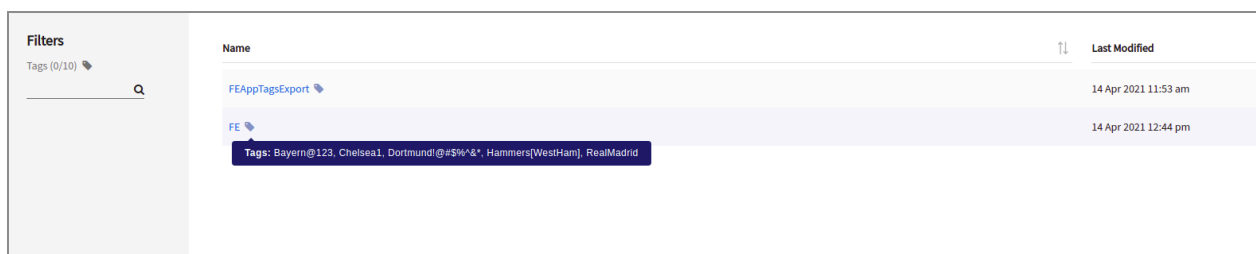
1. In the filters shown on the left side of the apps list, the **Tags** filter shows the total number of tags and search control. Enter text in the search filter to show buttons for each tag containing the text. This search is case-insensitive.

For example, the following screenshot shows the tags starting with the letters "Che".



2. Click a tag to limit the list of displayed apps to only apps with that tag. Click a selected tag to clear it. You can select multiple tags.

A tag icon  is next to each name in the **Name** column on the **Apps** page. Hover over the tag icon to see a list of all the tags in that app. The following screenshot displays the tags for the app **FE**.



i Note:

- When you export a Flogo app, its app tags are not retained in the Flogo JSON app archive file.
- When you create tags, they are case-sensitive, but the tag filter search is case-insensitive. For example, you can create unique app tags for abc and ABC, but when you search for an *a* in the search control, both are shown.

Role Requirements

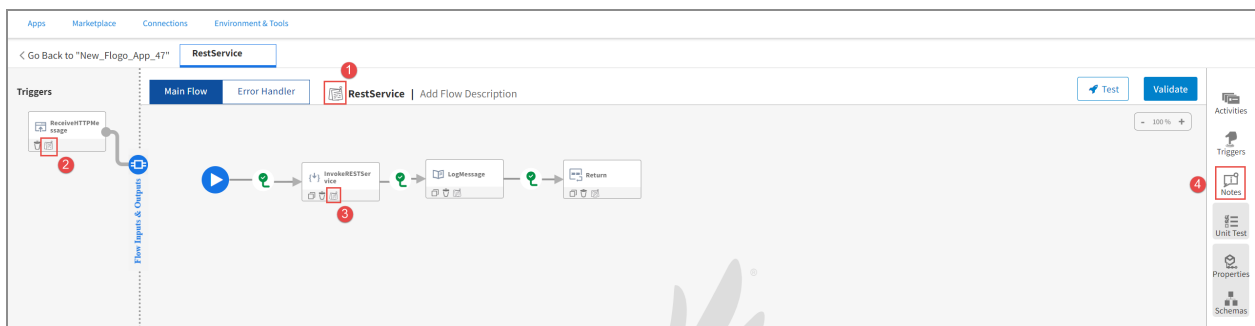
- Admins can edit tags for any app in their organization.
- Users can only edit tags on apps they own.
- Read-only users cannot change tags on any app.

Using Notes

You can use Notes to keep a track of information about any flow, activity, or trigger. It helps you in keeping updates and important references, especially when the flow is very large and complex. This feature is available in the Error Handler tab also.

Understanding Notes with an Example

Let us take an example of a Flogo app that invokes a REST service. You use the **ReceiveHTTPMessage** Trigger and **InvokeRESTService**, **LogMessage** and **Return** Activities to create the app.



Here, you can add Notes in the following manner:

1. Flow Note

This note gives information about the flow.

In the above case, the Flow Note can be - "This app invokes a Rest service and generates a log message that shows the status of the invoked Rest service".

2. Trigger Note

This note is used to add information about a respective Trigger.

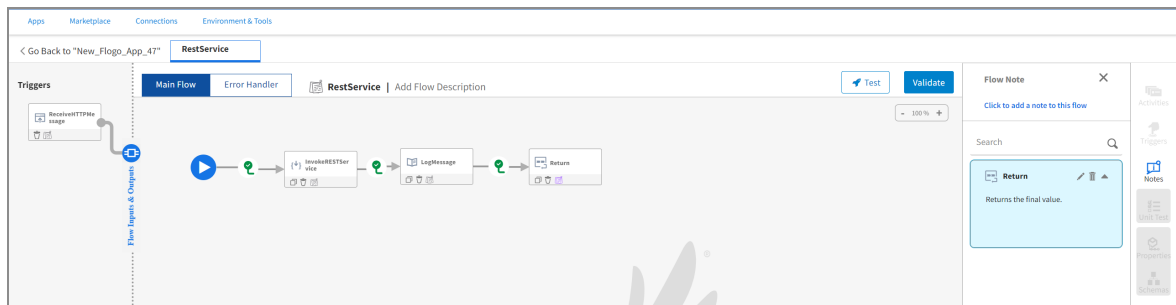
For the above example, add the Trigger Note that says - "This trigger listens to incoming REST requests."

3. Activity Note





This Note displays information about a respective Activity.

For the above example, the note for **InvokeRESTService** Activity can be - "This Activity invokes an external service".

4. To view all the Notes, click the icon on the right-hand sidebar.



Note:

- Use the  icon on the Activities and Triggers to add the respective Notes.
- Use the  icon next to the Error Handler tab to add Flow notes.
- In all cases,
 -  - Notes are not added,
 -  - Notes are added.
- The **Save** option on any note is enabled only after some content is added in it.
- If you are a Read-Only user, you cannot add, delete, or edit a note.

Switching Between Display Views On the App Page

On clicking an app name on the **Apps** page, the app details page opens. The flows in the app are listed on the app details page. You have the option to view this page in the **Trigger View** or **Flow View**. By default, it opens in the **Trigger View**. Click **Trigger View** dropdown and select **Flow View** from the menu to switch to the flow view. When you are in the flow view, click **Flow View** and select **Trigger View** from the dropdown menu to go back to the trigger view.

Trigger View

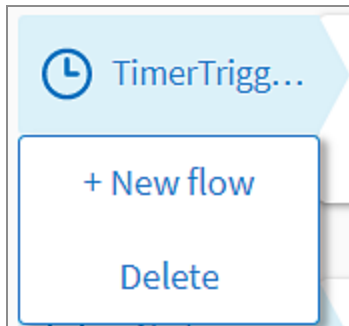
In this view, the flows are displayed attached to one or more triggers that they use. If a flow is attached to multiple triggers, it is attached to each trigger separately. You can see a single flow multiple times on the page but attached to different triggers. Flow that is not attached to any triggers display **No trigger** in place of the trigger name.

The following image shows the **Trigger View**:

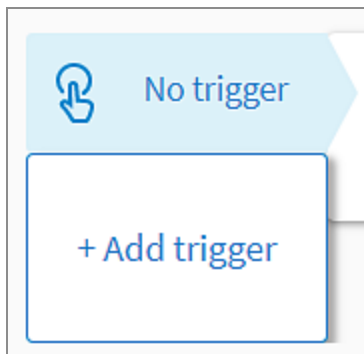


In the image above, **MyRESTFlow2** is attached to both **TimerTrigger** and **ReceiveHTTPMessage** trigger, hence it appears twice. The **MyTimerFlow** was created with a new timer trigger hence it is not attached to the first timer trigger and the **TimerTrigger** appears twice on the page.

Hovering on a trigger displays the **New flow** option. Click the **New flow** option to create a flow to attach the newly created flow to that trigger.



Hovering over **No trigger** displays the **Add trigger** option, which takes you to the triggers catalog.



Flow View

In this view, each flow is shown separately with a trigger attached to it on the extreme left side. Shown below is a **Flow View** representation of the **Trigger View** image above:

The following image shows the **Flow View**:




Notice that **MyRESTFlow2** shows two triggers. That is because this flow is attached to two triggers as you can see in the **Trigger View**. A blank flow shows **0** triggers against it as it is not attached to any triggers.

Deleting an App

You can delete an app using the **Delete app** icon, which appears when you hover your mouse cursor at the end of the app row.

To delete an app:

Procedure

1. On the **Apps** page, hover your mouse cursor to the end of the app row until the **Delete app** icon () appears.
2. Click the **Delete app** icon.
3. On the confirmation dialog, click **Delete app**.

Result

The selected app is deleted. The `<FLOGO_HOME>/apps/<app>/manifest.json` file is also deleted.

Exporting and Importing an App

You can export and import apps and use them as templates for development. Or, simply put them in a version control system such as GitHub.

- ✔ **Tip:** When an app is created, the `flogo.json` file and `manifest.json` file are automatically created in the `<FLOGO_HOME>/apps/<app>/` directory. Instead of using the **Export** option, you can use the files from the `<FLOGO_HOME>/apps/<app>/` directory to push the app directly using the TIBCO Cloud™ CLI. For more information see, [Creating an App](#) and [Editing an App](#).

Exporting an App

Here are a few things to keep in mind before you export an app:

- When you export an app, all the flows in your app get exported. You cannot choose the flows to export.

- Passwords that are configured in any activities within any flow or connection in the app to be exported are removed in the exported app. Manually configure the credentials in the flows after importing such apps.
- Some apps created in Project Flogo™ use the any data type. The any data type is not supported in Flogo Enterprise. Such apps get imported successfully, but the element of type any gets converted into an empty object. Explicitly use the mapper to populate the empty object with member elements.



Warning: When exporting an app, if the app contains open configurations to hold its test data, the open configurations are not exported with the app. Open **Configurations** in an app must be exported independently of the app export.

To export an app:

Procedure

1. On the **Apps** page, click the app to open the app details page.
2. Click the shortcut menu (⋮).
3. Click **Export**.

Using the **Export** option dropdown menu, download the following:

- **App** - exports a single `<appname>.json` file. You can use this option to download an app that you plan to import into TIBCO Cloud Integration using the drag-and-drop method.
- **TIBCO Cloud Integration artifacts** - downloads two files, `manifest.json`, and `flogo.json`. The `manifest.json` contains the manifest details such as the endpoints, memory resource details. The `flogo.json` contains the app itself. These artifacts are needed to push the app directly using the TIBCO Cloud™ CLI. You must have the TIBCO Cloud™ CLI installed on your local machine to do so. Use this option to push a Flogo app to TIBCO Cloud Integration without having to import it into TIBCO Cloud Integration. See the section [Pushing Apps to TIBCO Cloud](#) for details on how to do this.

Exporting an App's JSON File

When an app's binary is built, the `.json` file is embedded within the binary file. To export the `.json` file from the binary file to the disk, use the following command:


```
./<app-binary-name> --export app
```

The `.json` is exported as `<app-binary-name>.json`.

✔ **Tip:** To provide a different file name to the exported `.json`, use the following command:

```
./<app-binary-name> --export -o <new-app-binary-name>.json app
```

Importing an App

Importing the `.json` file of an app, makes it easy to use flows and triggers from another Flogo app. You can import the `.json` file to a new app, which does not have flows. You can also import it to an existing app that contains flows.

Important Considerations

Consider the following points before you import an app:

- Flogo apps that are exported from Flogo Enterprise 2.5.0 and later cannot be imported into previous versions of Flogo Enterprise.
- A flow in the app can have the community-developed extensions. You can import such apps without the extension. You can import the extension later by clicking the missing extension.
- Some apps created in Project Flogo use the any data type. The any data type is not supported in Flogo Enterprise. Such apps get imported successfully, but the element of type any gets converted into an empty object. Explicitly use the mapper to populate the empty object with member elements.
- The passwords and secrets for any connections configured in the app do not get imported. Reconfigure any password or secret for the connection after the app has been imported.
- When you import an app that does not have a **Return** Activity in any flow (main or branched flow), the **Return** Activity is not added automatically by default. However, if an existing app already has **Return** activities in main or branched flows, the app is imported as expected.

- When importing an app, the long and double data types get converted to the number data type.
- When importing an app into an existing app, if the existing app has entities with the same name as the ones you are importing, a warning is displayed. You can opt not to import those flows, activities, or triggers. You can go back and rename them using the UI, export the app again, and reimport it.
- When importing apps that were exported from Project Flogo, be aware of the following:
 - If the apps being imported use an Activity that is not supported in Flogo Enterprise, a validation error is displayed.
 - You can only import apps that were created in Project Flogo version 0.5.2 or above.

Importing Your App to a New App


Procedure

1. On the **Apps** page, click **Create/Import**.
2. In the block that displays below, upload the JSON file of the app to be imported. You can browse and select the file or drag it to upload the file.
3. Click the **Import Flogo app**.

Result

The app is created. After the import is complete, the **App Details Page** is displayed for the new app.

If your app uses a connection and that connection name and type exist in the org, your Flogo app uses the same connection by default. Otherwise, a new connection is created based on the imported app.

 **Note:** If you are reusing the same connection name and type, make sure that the credentials are correct or match with the intended usage.

For example, if you import an app that uses the Salesforce connection SFTest and a

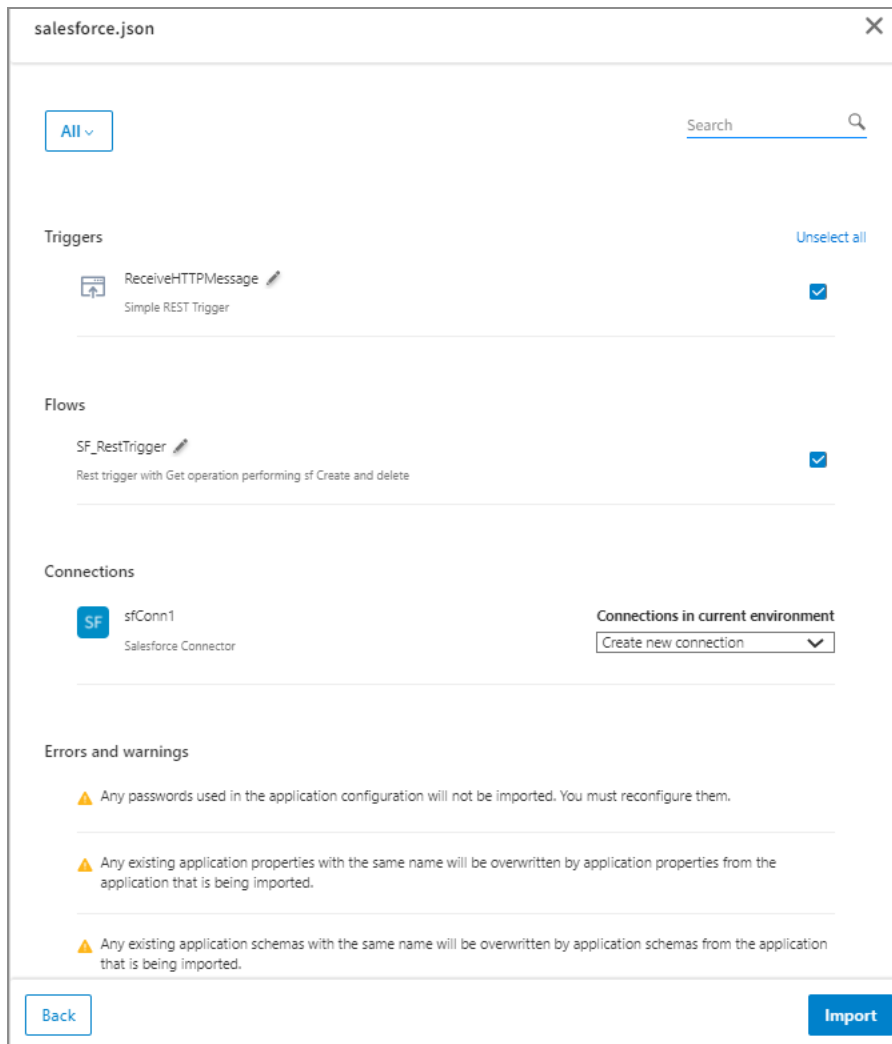
SalesForce connection with the name SFTest exists in your org, then the app being imported uses SFTest by default.

Importing Your App to an Existing App

Procedure

1. Log in to TIBCO Cloud™ Integration.
2. On the **Apps** page, open the existing app by clicking its name. The **Apps Details Page** is displayed.
3. Click the shortcut menu (⋮) and select **Import**.
4. In the **Import app** dialog, upload the JSON file of the app that you want to import. You can select the file or drag the file to upload. Click **Upload**.
5. In the dialog that opens, check the **Errors and Warnings** section for generic messages as well as any specific errors or warnings about the app you are importing. Flogo Enterprise validates whether all the activities and triggers used in the app are available on the **Extensions** tab.

i Note: The suffixes used in the mapper have undergone some changes. Due to this, you may receive a mapper-related warning in the dialog when importing an existing app. See [Changes in Suffixes Used in the Mapper](#).



6. Select the entities that you want to import from the source app:

- **Default:** All flows, triggers, and connections are selected for import.
- Use the dropdown list in the upper-left corner and the **Search** field to narrow down the information displayed.
- Use the checkboxes to clear selections of specific flows or triggers or click **Unselect all** to clear all the selections.
- If you select specific triggers or flows to import, the dialog lists only those connections that are used in the selected flows and triggers. If you want to use the existing connection, select the existing connection from the **Connections in**

current environment > Existing connections dropdown list. You can also choose to create a connection instead.

- If you select **All**, existing connections are automatically reused. A new connection is created by default. If you want to use an existing connection, select the existing connection from the **Connections in current environment > Existing connections** dropdown list.
 - If you select a trigger for import, all flows associated with that trigger are selected by default.
 - If you want to import a flow without importing the attached trigger, select the flow only. Do not select the attached trigger. The flow is imported as a blank flow without being attached to a trigger.
7. After ensuring that all the entities you want to import are selected, click **Import**.
 8. After importing an app, you must reconfigure all the newly created connections. For example, set the password of the new connection after the app is imported.

Changes in Suffixes Used in the Mapper

The suffixes used in the mapper have undergone some changes. Due to this, you may receive a mapper-related warning in the dialog when importing an existing app. Click **Continue** and the app imports successfully. After the import completes, be sure to remap the properties in the activities that show errors. This ensures that they switch to the new suffix format.

The following table lists the changes in the suffixes:

Original suffix appearing in imported apps	New suffix used by the Mapper (after you remap)	For example	Used when mapping
<i>Activity_id. Activity_ parameter</i>	<i>\$Activity[Activity_ id]. Activity_ parameter</i>	Old suffix: <i>\$InvokeRETSERVICE. responseBody.userId</i> New suffix after remapping property:	When mapping to a parameter in the Activity's output. Used to resolve Activity params.

Original suffix appearing in imported apps	New suffix used by the Mapper (after you remap)	For example	Used when mapping
		<pre>\$Activity [InvokeRESTService]. responseBody.userId</pre>	
\$TriggerData	\$trigger	<p>Old suffix:</p> <pre>\$TriggerData. queryParams.title</pre> <p>New suffix after remapping property:</p> <pre>\$trigger. queryParams. title</pre>	When mapping from the output of the trigger to the flow input
<p>N/A</p> <p>There was no equivalent for this in the old mapper</p>	<pre>\$flow.headers. parameter</pre> <pre>\$flow.body. parameter</pre>	<p>\$flow is a newly introduced suffix, which did not have an equivalent suffix in the old mapper.</p>	<p>When mapping to any parameter in the flow's header or input schema (schema entered on the Input tab of Flow Inputs & Outputs dialog) which is the same as the output of the trigger, since the output of the trigger is mapped to the input of the flow.</p> <p>Used to resolve parameters from within the current flow. If a flow has a single trigger and no input parameters defined, then the output of the trigger is made available via \$flow.</p>

Resolving Missing Activities and Triggers

When you import an app that contains one or more activities or triggers that are not installed in your environment, you see a warning in the **Import App** dialog.


i Note: When importing an app that has a connection configured in it, but the connector is not installed in your environment, after you install the connector, the connection configuration field values of type SECRETS are retained postinstallation as long as they were not configured using app properties. If you had configured your SECRETS as app properties, reconfigure them after installing the missing connector. This is because all configured app properties are wiped out when the app is imported.

To resolve missing activities or triggers for TIBCO provided connectors

When an Activity or trigger used in an app being imported is missing from your Flogo Enterprise environment, the flows in the app get imported, but you see a warning in the **Import App** dialog.

Errors and warnings

 The Connector **AWSKINESIS** is used in your application but not installed. Some activities and/or trigger from this connector used in flows. The application will be imported but will not work until you install the required Tibco extension. Check connector installation guide.

When you validate your app by clicking **Validate** in the app details dialog, you see an error marker () next to the flow name. This indicates that one or more activities or triggers are missing. The number next to it indicates how many activities or triggers that are missing appear in the flow. When you click the missing activities or triggers, you are prompted to refer to the connector installation guide.


i Note: Do not upload a TIBCO connector using **Upload Extension**. For more information on how to install a TIBCO connector, refer to the connector installation guide.

This is also true when you copy an app into the designated folder (the folder you specified when you started the UI) for your apps on your local machine.

To resolve missing custom activities or triggers

When one or more of your custom activities or triggers used in the app being imported are missing from your Flogo Enterprise installation, you see a warning in your **Import App** dialog similar to the following:



Once the app is imported, you see an error marker () next to the flow name. After you install the missing Activity or trigger, this marker goes away. The number next to the error indicates how many activities or triggers are missing in the flow.

To install the missing custom activities or triggers:

1. Click the flow name to open the flow details page. The **Upload an extension** dialog opens. You can upload the custom Activity or trigger from the Git repository, hence only the **From Git repository** option is enabled.
2. Click **From Git repository**. The **Git repository URL** text box is pre-populated.
3. Click **Import**. Flogo Enterprise downloads the Activity or trigger from the Git repository and uploads it on your **Extensions** tab. Refer to the section, [Uploading Extensions](#) for details on this option.

App File Persistence

Your Flogo app files get persisted to the directory that you specify on your local machine. You can use an external source control system such as Git or SVN to store your apps. You can then check in and check out your apps locally from the remote repository. This makes it possible for you to implement the continuous Integration/Continuous deployment (CI/CD) pipeline by leveraging any tool available in the market to integrate your app development with the app deployment.

When you start the UI, you are prompted to point to the directory where you have checked out your apps. If you do not provide any path, the apps are stored in the default directory, which is: `<FLOGO_HOME>/data/localstack/apps`.

If you restart the UI, at the time of restart, if you want to continue using the same directory that you had specified, click **Enter** on your keyboard when it prompts you to set the path. It stores the path preference that you set the last time.

After the UI starts, you should be able to see all your apps on the app list page in the UI. From this point on, when you create an app or modify an existing app, the changes are saved to the directory location that you provided when starting Flogo Enterprise.

Each app that you store on your local machine has its folder and the folder name must be identical to the app name. If another user makes changes to your app, you must sync your local repository with the remote repository (do a pull) to get the changes made by that user.



Warning:

- The app file name must be called `flogo.json`.
- The folder name containing the app must be identical to the app name appearing in the `flogo.json` file for the app.

Loading new apps from the disk - When a new app is added to the directory, refreshing the browser loads the app into the UI. You do not need to restart the UI.

Loading the updated app from the disk - In case the `flogo.json` on the disk is updated (due to minor changes or checkout a newer version from the source control system), click the **Reload from Disk** to load the updated app into the UI. Be aware that this action overrides existing changes in the app. **Reload from Disk** option is available under the shortcut menu that is next to the other buttons on the app page.



If another user adds an app to your remote repository, the app gets downloaded to your local repository when you do a pull from the remote repository. For the new app to display on the UI, you must refresh your browser. You do not need to restart either the browser or Flogo Enterprise.

You can import any exported app to Flogo Enterprise. To do so, create a folder with an identical name as the app name in your local repository, then copy the `flogo.json` file for the app to the folder. For apps that are created using the UI, Flogo Enterprise

automatically generates a unique ID for each app. But, if you load an existing `flogo.json` file, the app may or may not have an app ID defined in it. Flogo Enterprise checks to see if an ID exists in the `flogo.json` file for the app. If an ID does not exist for the app, Flogo Enterprise generates a unique ID and adds an ID attribute in the `flogo.json` file before loading the app.

Note the following:

- If you change the ID of the app in your `flogo.json` file, you see a duplicate app on the UI. Refresh your browser to fix this issue. If you continue to work on the app with the old app ID, your changes are lost when you restart the UI.
- All apps that exist in the path that you provided during Flogo Enterprise installation get loaded on the UI. You cannot selectively choose the apps to be loaded on the UI.
- Any **Launch Configurations** (containing your test data for the app) associated with the app are stored in the `<app_folder> > test` folder along with the `flogo.json` file for the app.
- File permissions - You must have "write" permission for the app directory on your local machine. Otherwise, the app is not loaded and displayed in the UI. An error is displayed in the log located in `<FLOGO_HOME>/<FLOGO_VERSION>/logs/studio.logs`.
- When importing an app, if any extensions are missing, a broken plug-in icon is displayed on the missing Activity.
- If the app has any missing extension or if a connector uses the associated connection, you see the connection post-installation of the missing extension or connector.
- If you add an app to your local app repository, if that app has any missing extension, after uploading the missing extension, the connection in the extension maintains the secrets and passwords that were already configured in the connection for the app. Refer to the [Resolving Missing Extensions](#) section for details on how to resolve missing extensions in an app.
- You may notice a change in secret encrypted values in `flogo.json` after opening the apps using the UI. This does not affect the run time.
- We recommend that you do not modify `flogo.json` manually to avoid any mishaps.
- When upgrading to Flogo Enterprise the current version from an older Flogo Enterprise version, the existing apps automatically get migrated to the directory that you have created on your local disk. You do not need to migrate them manually.
- If your app repository gets deleted while in use, you must restart the UI and set a

new app repository. Do not continue to work with the deleted repository. Also keep in mind that even if you recreate a directory with the same name, your changes do not take effect until you restart the UI.

Creating Flows and Triggers

An app can have one or more flows and a flow can be attached to one or more triggers. Similarly, a trigger can have multiple flows attached to it.

Flows

Each flow represents specific business logic in an app. A flow contains one or more activities. The flow execution is started by a trigger. A new flow can be created only from the app details page.

Triggers

You have the option to create a trigger without creating a flow. You can create a trigger from an existing specification that you have saved in either the TIBCO Cloud™ Integration - API Modeler or on your local machine. Optionally, you can create a trigger when creating a flow by selecting the **Start with a trigger** option during flow creation.

Activities, Triggers, Unit test, Properties and Schema Panel

You can add a trigger or an activity in a flow from the **Activities** or **Triggers** palettes available in this panel. You can also get in or out of the unit test mode using the **Unit Test** palette. **Properties** and **Schemas** palette can be used to manage the properties and schemas. By default the panel exists on the right side of the canvas, but by clicking the Move to Left « icon you can move the panel to the left of the canvas.


Flows

This section contains information about creating and managing flows in your app.

Creating a Flow

Every app has at least one flow. Each flow can be attached to one or more triggers. You have the option to first create a blank flow (a flow without a trigger) and then attach the flow to one or more triggers. On the **App Details** page, click **Create** to create the first flow in an app.

Before creating a flow that uses connectors, ensure that you create the required connections. For more information, see [created the necessary connections](#).

 **Warning:** In an app with multiple triggers, the port number must be unique for all the triggers that require a port number. For example, REST and/or GraphQL triggers. Two triggers in the same app cannot run on the same port.

For flows that are attached to multiple triggers, you cannot disable a trigger. Specify a particular trigger to run. Or, specify the order in which the triggers run. When a flow runs, all triggers get initialized in the order that they appear within the flow.

When using the Lambda, S3, or Gateway triggers, keep the following in mind:

- You can have only one Lambda trigger. An app that has a Lambda trigger cannot contain any other triggers including another Lambda trigger. Also, as the Lambda trigger supports only one handler per trigger, it can have only one flow attached to it. However, the apps that contain a Lambda trigger can contain blank flows that can serve as subflows for the flow attached to the Lambda trigger.
- You can have only one S3 trigger in an app. An app that has an S3 trigger cannot contain any other triggers including another S3 trigger. The S3 trigger supports multiple handlers (flows), so you can have multiple flows in the app that are attached to the same S3 trigger. You can also have blank flows in the app, which can serve as subflows for the flows that are attached to the S3 trigger.
- You can have only one Gateway trigger in an app. An app that has a Gateway trigger cannot contain any other triggers including another Gateway trigger. The Gateway trigger supports multiple handlers (flows), so you can have multiple flows in the app that are attached to the same Gateway trigger. You can also have blank flows in the app, which can serve as subflows for the flows that are attached to the Gateway trigger.

The output of a trigger provides the input to the flow. Hence, it must be mapped to the flow input. When creating a flow without a trigger, there must be a well-defined contract within the flow that specifies the input to the flow and the output expected after the flow

completes execution. You define this contract in the **Flow Inputs & Outputs** dialog. The **Flow Inputs & Outputs** contract works as a bridge between the flow and the trigger, hence every trigger has to be configured to map its output to the **Input** parameters defined in **Flow Inputs & Outputs**. You do this on the **Map to Flow Inputs** tab of the trigger.

Likewise, for triggers (such as the **ReceiveHTTPMessage** REST trigger) that send back a reply to the caller, the trigger reply must be mapped to the flow outputs (parameters configured on the **Output** tab of the **Flow Inputs & Outputs** accordion tab). You do this mapping on the **Map from Flow Outputs** tab of the trigger.

A **Return** Activity is not added by default. Depending on your requirements, you must add and configure the **Return** Activity manually. For example, if any trigger needs to send a response back to a server, its output must be mapped to the output of the **Return** Activity in the flow.

The **Map Outputs** tab of the **Return** Activity displays the flow output schema that you configured on the **Output** tab of the **Flow Inputs & Outputs** accordion tab. The **Map from Flow Output** tab of the trigger constitutes the trigger reply. This tab also displays the flow output schema that you configured on the **Output** tab of the **Flow Inputs & Outputs** accordion tab.

Perform the following steps when using a **ReceiveHTTPMessage** REST trigger:

- Add a **Return** Activity at the end of the flow.
- On the **Map Outputs** tab of the **Return** Activity, map the elements in the schema to the data coming from the upstream activities.
- On the **Map from Flow Output** tab of the trigger, map the trigger reply elements to the flow output elements.

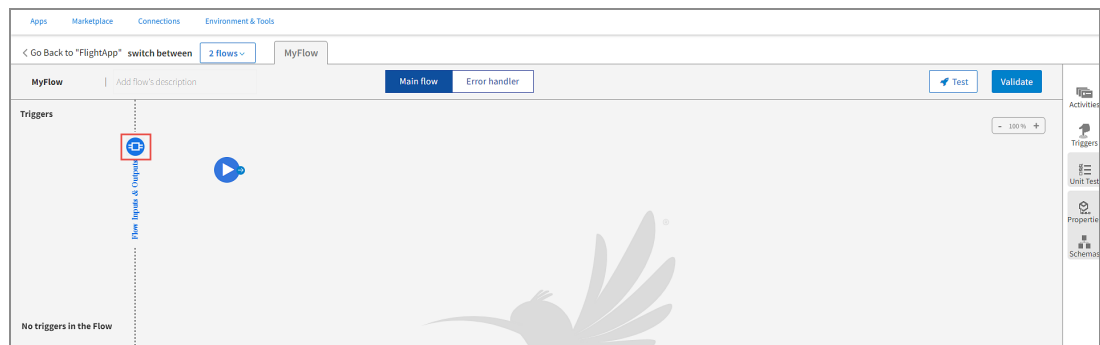
Follow these steps to create a flow:

Procedure

1. On the **Apps** page, Click an app name to open its page.
2. Under the **Flows** page, click **Create** . The **Add triggers and flows** dialog opens.
3. Enter a name for the flow in the **Name** text box. Flow names within an app must be unique. An app cannot contain two flows with the same name.
4. Optionally, enter a brief description of what the flow does in the **Description** text box. The **Flow** option is selected by default. To create a flow from a specification, select the specification under **Start with** and refer to the appropriate section under

Building APIs.

5. Click **Create**. The **Flow** gets created.
6. After a **Flow** is created, you can start with either of the following actions:
 - Start with a trigger - If you know the trigger with which you want to activate the flow, select this option. Select a trigger from the **Triggers** palette. For more details on the type of trigger that you want to create, see the relevant section in the [Starting with a Trigger](#) topic. If there are existing flows attached to triggers, you are prompted to either use an existing trigger or use a new trigger that has not been used in an existing flow within the app.
 - Configure flow inputs and outputs - Select this option if you know the algorithm for the flow, but do not yet know the circumstances that cause the flow to run. It creates a blank flow that is not attached to any trigger. Flow inputs and outputs create a contract between the trigger and the flow. When you create a trigger, you must map the output of the trigger to the input of the flow. This contract serves as a bridge between the trigger and the flow. You have the option to attach your flow to one or more triggers at any later time after the flow has been created.



If you selected **Start with a trigger**, the flow is attached to the trigger you selected. If you selected **Configure flow inputs and outputs**, a blank flow without a trigger gets created.

i Note: **StartActivity** is a special activity that is always added to the newly created flows.

Selecting a Trigger When Creating a New Flow

When creating a flow, you have the option to either select an existing trigger or select one from the triggers palette.

Trigger configuration fields are categorized into two groups as explained below. A single trigger can be associated with multiple handlers.

- **Trigger Settings** - These settings are common for the trigger across all flows that use that trigger. If and when a flow attached to the trigger changes any **Trigger Settings** field, the change gets propagated to all flows attached to the trigger.
- **Handler Settings** - These settings apply to a specific flow attached to the trigger. Hence, each flow can set its values for the **Handler Settings** fields in the trigger. To do so, open the flow and click the trigger to open its configuration dialog. Click the **Settings** tab and edit the fields in the **Handler Settings** section.

Creating a Trigger When Another Trigger of the Same Type Exists

There may be cases when a specific type of trigger exists. For example, there might be a REST trigger that exists. When creating a REST flow, you are prompted to select the existing REST trigger or create a trigger by selecting it from the triggers palette. If you want a REST trigger with a different trigger setting than the one that exists, such as a different port or a security options, Select the **Create new** option and then select the trigger from the ensuing trigger palette. This creates a REST trigger and attaches your new flow to it.

Creating a Flow Starting with a Trigger

When creating a flow, if you know the circumstances in which you want the flow to activate, select the **Start with a trigger** option and select an available trigger that activates the flow.



Warning: If an app has multiple triggers that require a port to be specified, make sure that the port number is unique for each trigger. For example, REST or GraphQL trigger. Two triggers in the same app cannot run on the same port.

If you are unsure of the circumstances under which the flow should be activated, or if you want the flow to be activated under more than one situation, use the **Configure flow inputs and outputs** option and attach the flow to one or more triggers later as needed. See [Creating a Flow without a Trigger](#) for more details on this.

Creating a Flow Attached to a REST Trigger

When creating a flow with a REST (Receive HTTP Message) trigger, you can enter the schema in the **Configure trigger** dialog during flow creation. Also, you can use a Swagger 2.0 or OpenAPI 3.0 specification file that you have saved either in TIBCO Cloud™ Integration - API Modeler or on your local machine.

For more details on using a specification file, see the [Using an OpenAPI Specification](#).

i Note: If you want to have two flows using the same operation on the same resource, while creating the flows on the **Settings** tab of the **ReceiveHTTPMessage** trigger, ensure that you configure different ports for each flow.

You can create a REST flow by entering a JSON schema or dragging an API specification JSON file. See the [Using an OpenAPI Specification](#) section about using a specification file.

⚠ Warning: If you modify the **Reply Settings** tab of a **ReceiveHTTPMessage** trigger, the corresponding **ConfigureHTTPResponse** activities within that flow do not change appropriately. This happens when you remove fields from the **Reply Settings** tab. Redo the mappings for the **ConfigureHTTPResponse** Activity.

To create a REST flow by entering the schema:

Procedure

1. Click an app name on the **Apps** page.
2. Click **Create**. The **Add triggers and flows** dialog opens. **Flow** under **Create New** is selected by default.
3. Enter a name for the flow in the **Name** text box. Flow names within an app must be unique.
4. Optionally, enter a brief description of the flow in the **Description** text box.
5. Click **Create**. A flow with a specified name is created.
6. Now, click **Triggers** palette. The triggers palette opens with all the available triggers listed.
7. Drag the **Receive HTTP Message** to the **Triggers** area on the left. The trigger

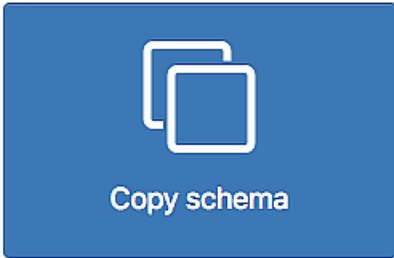
configuration dialog opens.

8. Select the REST operation under the **Method** that you want to implement by clicking it.


i **Note:** Two REST triggers cannot have an identical port, path, and method combination. Each REST trigger needs to differ from the other for the same flow with either a unique port, path, or operation.

9. Enter a resource path in the **Resource Path** text box.
10. Enter the JSON schema or JSON sample data for the operation in the **Response Schema** text box. This is the schema for both input and output.
11. Click **Continue**.
12. Select one of the following options:

Do you want to copy this trigger's **Output Schema** into the **Flow's Inputs**?



Copy schema



Just add the trigger

If you select **Copy Schema**, the schema that you entered in the step 10 above automatically gets copied or displayed in a tree format to the following locations when the trigger gets added:

- Trigger output, on the **Map to Flow Inputs** tab of the trigger
- Flow input, on the **Input Settings** tab of the **Flow Inputs & Outputs** accordion tab.
- Trigger reply (if the trigger has a reply), in the **Reply Settings** of the trigger.

For details on configuration parameters, see the [REST Trigger](#) section.

If you select **Just add the trigger**, a REST trigger is added to the flow without any configuration. You can configure this REST trigger later by clicking the trigger from the app details page. Any changes made to the trigger must be saved by clicking **Save**.

The flow page opens.

13. Map the trigger output to the flow input.
 - a. Open the trigger configuration dialog by clicking the trigger:
 - b. Open the **Map to Flow Inputs** tab.
 - c. Map the elements under **Flow inputs** to their corresponding elements under **Available data** one at a time.
14. Map the flow output to the trigger reply as follows:
 - a. In the trigger configuration dialog, click the **Map from Flow Outputs** tab.
 - b. Map the elements under **Trigger reply** to their corresponding elements under **Available data**.
 - c. Close the dialog.
15. Click **Save** to save your changes.

Creating a Flow attached to the GraphQL Trigger

You can create GraphQL flows by uploading a GraphQL schema file with an `.gql` or `.graphql` extension. Flogo then creates the appropriate flows based on your schema. When the flow gets created, a GraphQL trigger automatically gets generated and attached to each flow that gets created.

To create a flow using a GraphQL schema, see the [Using GraphQL Schema](#) topic. For details on the GraphQL trigger, see "GraphQL Trigger" section in *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide*.

Creating a Flow Attached to Other Triggers

This section applies to triggers that are not REST, or GraphQL triggers.

To create a flow with such a trigger:

Procedure

1. Click an app name on the **Apps** page to open the app details page.
2. click **Create**. The **Add triggers and flows** dialog opens.
3. Enter a name for the flow in the **Name** text box.
Flow names within an app must be unique.
4. Optionally, enter a brief description of what the flow does in the **Description** text box and click **Create**.
A flow gets created and the flow details page opens.
5. From the **Triggers** palette, select the desired trigger and drag it to the triggers area.
6. Click the trigger to display its properties.
7. Configure the properties for the trigger. See the respective trigger section in *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details.

Creating a Blank Flow (Flow without a Trigger)

You can create a flow in the Flogo app without attaching it to a trigger. This method of creating a blank flow is useful when the logic for the flow is available, but you do not know the condition under which the flow should activate. You can start by creating a flow with the logic and attach it to one or more triggers later.

Follow these steps to create a flow without a trigger:

Procedure

1. Click an app name on the **Apps** page to open the app details page.
2. click **Create**. The **Add triggers and flows** dialog opens. **Flow** is selected by default.
3. Enter a name for the flow in the **Name** text box.
Flow names within an app must be unique.
4. Optionally, enter a brief description of what the flow does in the **Description** text box.
5. Click **Create**. The flow details page opens.

6. Click **Flow Inputs & Outputs** to configure the inputs and/or outputs to the flow on the **Input** or **Output** tab respectively. See [Flow Inputs & Outputs Tab](#).

Mapping trigger outputs to flow inputs and flow outputs to trigger reply creates a contract between the trigger and the flow. Hence, when you attach the flow to a trigger later, you must map the output of the trigger to the flow input. You have the option to attach your flow to one or more triggers later after the flow has been created. See [Attaching a Flow to One or More Triggers](#) for details.

7. Enter a JSON schema containing the input fields to the flow on the **Input Settings** tab and click **Save**.
8. Enter the JSON schema containing the flow output fields on the **Output Settings** tab and click **Save**.
9. When you are ready to add a trigger, refer to [Adding Triggers to a Flow](#) to add one or more triggers to the flow. For triggers that need to send back a response to the server, you must map the flow output to the reply of the trigger.

Flow Input & Output Tab

Use these tabs to configure the input to the flow and the flow output. These tabs are particularly useful when you create blank flows that are not attached to any triggers.

i Note: The schemas for input and output to a flow can be entered or modified only on this **Flow Inputs & Outputs** tab. You cannot coerce the flow input or output from outside this accordion tab.

Both these tabs (the **Input** tab and the **Output** tab) have two views:

- **JSON schema view:**

You can enter either the JSON data or the JSON schema in this view. Click **Save** to save your changes or **Discard** to revert the changes. If you entered JSON data, the data is converted to a JSON schema automatically when you click **Save**.

- **List view:**

This view displays the data that you entered in the JSON schema view in a list format. In this view, you can:

- Enter your data directly by adding parameters one at a time
- Mark parameters as required by selecting its checkbox.

- When creating a parameter, if you select its data type like an array or an object, an ellipsis (...) appears to the right of the data type. Click the ellipsis to provide a schema for the object or array.
- Use an app-level schema by selecting **Use an app-level schema**. On the **Schemas** page that appears, click **Select** beside the schema that you want to use. The name of the schema is displayed beside **Use an app-level schema** and the schema is displayed in a read-only mode.

i **Note:** You cannot edit an app-level schema in the **List** view if **Use an app-level schema** is selected. To edit an app-level schema, follow the instructions in the section [Editing an App-level Schema](#). You can, however, switch to another app-level schema by clicking **Change** and selecting another app-level schema. You can also unbind the app-level schema (by deselecting **Use an app-level schema**) from a trigger, activity, or the input and output of a flow. After you unbind the app-level schema, you can make changes to it using the schema editor in the **List** view.

- Click **Save** to save the changes or **Discard** to discard your changes.

Attaching a Flow to One or More Triggers

If you had created a blank flow without attaching it to a trigger, you can attach it to an existing trigger that is being used by another flow in the same app.

A flow that was created without being attached to a trigger has its input and output parameters defined on the **Flow Inputs & Outputs** accordion tab. You can access it by clicking the blue bar with the same label. The output from the trigger is the input to the flow. So, you must map the input parameters defined on the **Input** tab of this dialog to the trigger output parameters. This mapping must be done in the trigger. The mapping creates a contract between the trigger and the flow and is mandatory for the flow and the trigger to interact with each other.

You can use one of these methods to attach a flow to a trigger:

1. From the app details page:
 - a. Open the app details page by clicking the app.
 - b. Hover over **No trigger**, then click **Add trigger**. The flow details page opens.

2. From the flow details page:
 - a. Open the flow details page by clicking the flow name on the app details page.
 - b. From the **Triggers** palette, drag a desired trigger to the triggers area.

For REST and GraphQL triggers, you are prompted to enter additional handler setting details. Refer to the "REST Trigger" and "GraphQL Trigger" section in *TIBCO Flogo® Enterprise Activities, Triggers, and Connections*.

Click the trigger icon to configure the trigger as needed. For REST and GraphQL triggers, be sure to map the trigger outputs to flow inputs and the flow outputs to the trigger reply.

Catching Errors

You can configure a flow to catch errors at two levels:

- At the flow level by configuring the **Error Handler** in the flow. Refer to the section, [Creating an Error Handler Flow](#) for more details on configuring the error handler in the flow.
- At the Activity level by creating an error branch from an Activity. Refer to the [Types of Branch Conditions](#) section for details on how to create an error branch from an Activity.

Creating an Error Handler Flow

Use the **Error Handler** to catch exceptions that occur while running a flow. The error handler is designed to catch exceptions in the activities within a flow. If there are multiple flows in an app, the error handler must be configured for each flow separately. Branching is supported for error handler flows similar to the other flows.

To configure the error handler:

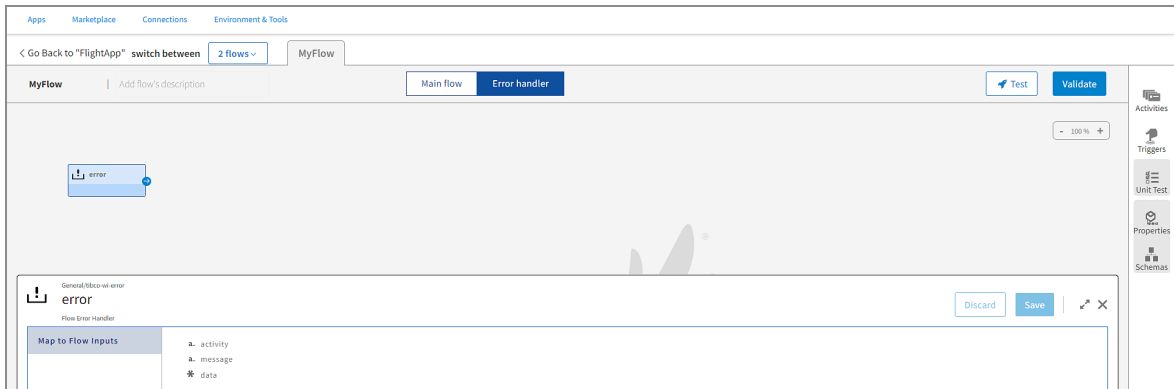
Procedure

1. Click an existing activity in a flow.
2. Click the **Error handler** tab.

The error handler opens with the **error** Activity displayed.

Clicking the **error** activity exposes the fields that you can configure for an error that

is generated by the activity.



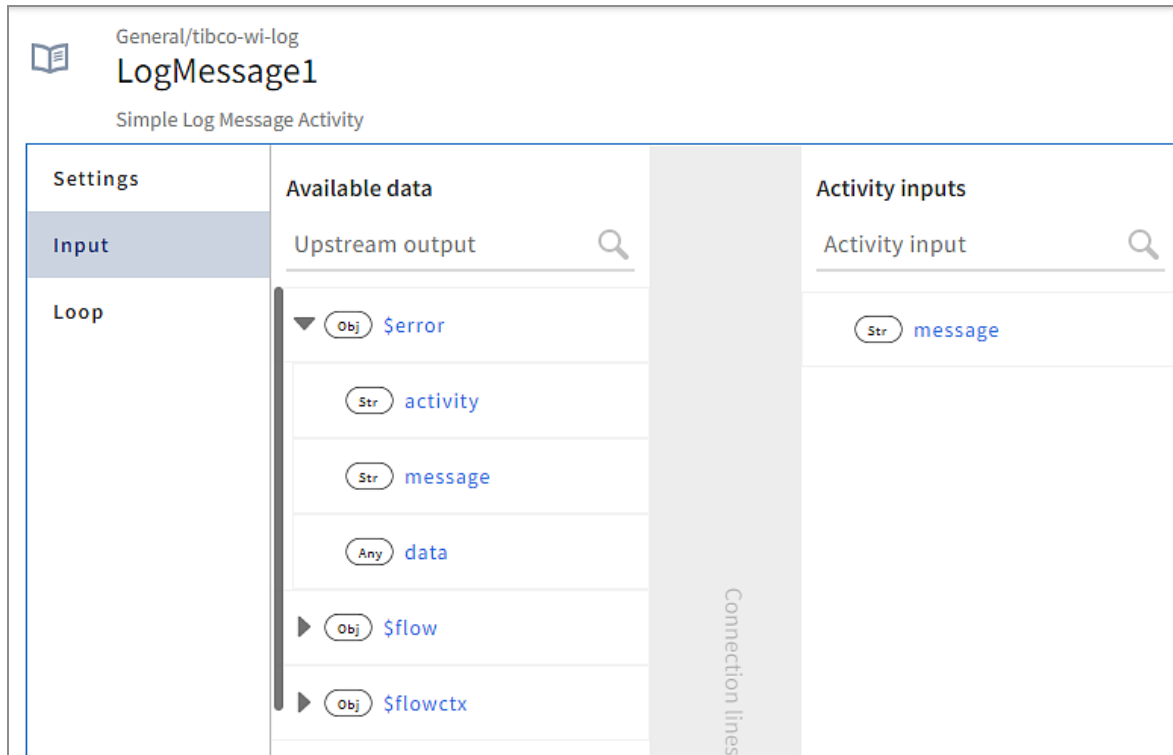
The **Map to Flow Inputs** tab of the **error** Activity has three elements, **Activity**, **message**, and **data**. The **activity** element is used to output the name of the activity that is generating the error, the **message** element is used to output the error message string, and the **data** element can be configured to output any data related to the error. The **message** element on the **Input** tab of any activity in the **Error Handler** flow can be configured to output one or all of these three elements.

3. From the **Activities** palette, add an activity for which you want to configure the error message. Add a branch to connect the **error** with the activity that you have added.

The **Input** tab of that Activity displays a **message** in its input schema. This is a required element that you must map.

Note: A **Return** Activity is not added by default. Depending on your requirements, you must add the **Return** Activity manually.

4. Click the **message** in the input schema to open its mapper.



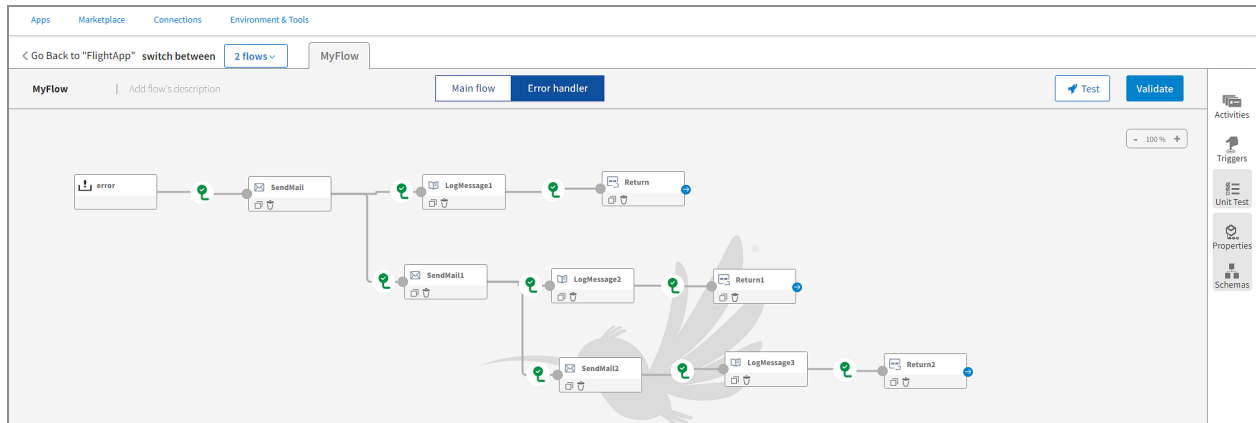
- Expand **\$error** to expose the **Activity**, **message**, and **data** elements that you can configure for the error message.

To map the **message** element under **Activity inputs**, you can either manually type in the error string enclosed in double-quotes or use the **concat** function under **string** in the mapper to output the Activity name along with a message. See [Using Functions](#) for more details.

- Continue configuring the error message for each activity in the flow.

If there is error for the activity in any flow of the app, it is output in the log for the app when the app is built.




Here is an example of how an error handler flow looks after it is configured:



Viewing Errors and Warnings

Flogo Enterprise uses distinct icons to display errors and warnings within an app.

The following icons are used:

-  - error icon. Resolve the errors before building the app. Errors should not be ignored.
-  - warning icon. Warnings are generated to alert you of something that might need to change in the entity where the warning icon is displayed. You have the option to ignore the warning and move on.
-  - missing extension icon. Check and correct missing extensions before building an app. Missing extensions must be fixed before proceeding.

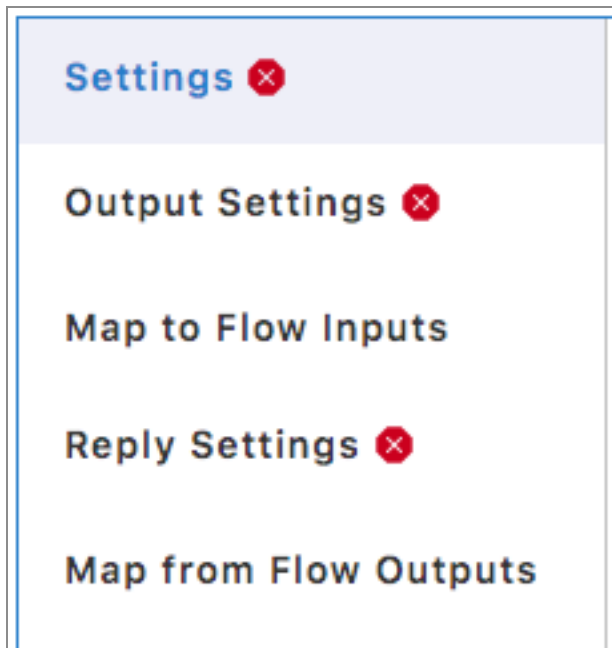
Here is the hierarchy of errors and warnings reported in Flogo Enterprise:

Flow level reporting - when you click an app name, the app details page opens displaying the list of flows in the app. If there are errors or warnings in a flow, appropriate icons are displayed next to the flow name along with a number, where the number indicates an aggregate number of errors or warnings in the flow. If there are no errors or warnings, these icons are not displayed.

Activity and Trigger level reporting - when you click a flow name, the flow details page opens displaying the implementation of the flow. This page displays errors if any at the activity level. For instance, a **LogMessage** activity may displays an error symbol within the activity configuration. Resolve the error before proceeding.

Activity and Trigger configuration tab level reporting - when you click an activity or a trigger in the flow, its configuration page opens, displaying the various tabs. Click a tab to see the errors or warnings in the configuration within that tab.

Activity and Trigger configuration tab level reporting - When you click on an activity or a trigger in the flow, its configuration page opens, displaying the various tabs. Click a tab to see the errors or warnings in the configuration within that tab.



Using Subflows

Flogo provides the ability to call any flow from another flow in the same app. The flow being called becomes the subflow of the caller flow. This helps in separating the common app logic by extracting the reusable components in the app and creating standalone flows for them within the app. Any flow in the app can become a subflow for another flow within the same app. Also, there are no restrictions on how many subflows a flow can have or how many times the same subflow can be called or iterated in another flow. Hence, subflows are useful when you want to iterate a piece of app logic more than once or have the same piece of logic repeat in multiple locations within the app.

Here are a few points to keep in mind when creating and using subflows:

- The subflow and its calling flow must both reside within the same app. You cannot call a flow from another app as a subflow in your app.

- Since you can call any flow from any other flow within the app, you must be careful not to create cyclical dependency where a flow calls a subflow and the subflow, in turn, calls its calling flow. This results in an infinite calling cycle and the "Cyclic dependency detected in the subflow" error is displayed.
- You can configure the iteration details on the **Loop** tab of the **Start a SubFlow** Activity. The **Start a SubFlow** Activity iterates multiple times, resulting in the subflow being called multiple times.

! **Important:** You can delete any flow in an app even though the flow might be in use as a subflow within another flow. You do not receive any error messages at the time of deletion, but when you run the app, its execution fails with an error.

Creating Subflows


You create a subflow exactly like you would create any other blank flow.

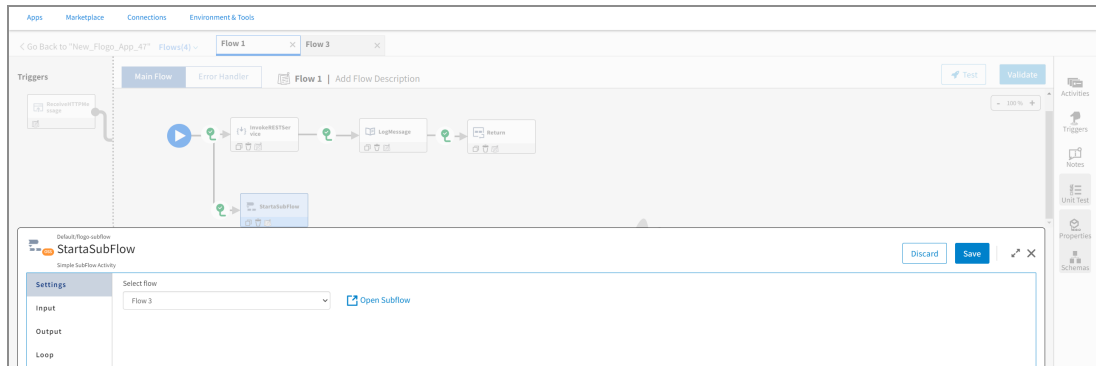
To create a subflow:

Procedure



1. Identify the piece of logic in your app that you want to reuse elsewhere in the app or iterate multiple times.
2. Create a flow without a trigger for that logic. For details on how to create such a flow, see the [Creating a flow without a trigger](#).
3. To use this flow as a subflow within another flow, you must add a **Start a SubFlow** Activity at the location in the calling flow from where you want to call the subflow. For example, if you want to call a subflow after the third Activity in your calling flow, insert a **Start a SubFlow** Activity as the fourth Activity in the calling flow. To do so:
 - a. Open the calling flow.
 - b. On the flow details page, click the **Activities** palette.
 - c. Under the **Default** category, select the **Start a SubFlow** activity and drag it to the activities area.
 - d. Add the branches to connect the **SubFlow** activity with the activity that you want to call a subflow from and to the activity where the subflow must end.

Also set the branch conditions for each connection line wherever required.

- e. Click the **StartaSubFlow** activity to open the configuration dialog. To call the required subflow, select the subflow from the **Select flow** dropdown in the **Settings** tab and save the changes. If you want to see the flow in detail use the **Open Subflow** option in the **Settings** tab or click the  icon on the **StartaSubflow** Activity. This appends the flow tab to the right of the previously appended flow tabs. In the below example, we call the **Flow 3** flow using the **StartaSubflow** Activity. After clicking the **Open Subflow** tab, we see the **Flow 3** flow tab next to the **Flow 1** flow tab.



Note:

- If the subflow is already selected in **StartaSubflow** Activity, then you can directly open that subflow in a different flow tab by clicking on the  icon on activity tile.
- If  icon is present on the **StartaSubflow** Activity, it means a subflow is selected in the **StartaSubflow** Activity.
- An opened subflow tab becomes "active" only after you select it from the **Flows** list dropdown or when you switch to that subflow.
- When you try to open an already opened subflow tab, it is highlighted.

The schemas that you had configured in the **Input Settings** and **Output Settings** of the **Flow Inputs&Outputs** tab in the selected subflow appear on the **Input** and **Output** tabs of the **StartaSubFlow** Activity.

You can now configure the input and output for the subflow in the **StartaSubFlow** Activity. If you add additional input and/or output parameters on the **Flow Inputs & Outputs** tab of your subflow, they become available to configure from the **Input** and/or **Output** tabs of the **StartaSubFlow** Activity. The output from the **StartaSubFlow** Activity is available for use as input in all activities that appear after it.

At app runtime, the **StartaSubFlow** Activity in the calling flow calls the selected subflow.

- f. If you want your subflow to iterate multiple times, configure the iteration details on the **Loop** tab of the **StartaSubFlow** activity. Refer to the [Using the Loop](#) section for details on how to configure the **Loop** tab.
- g. If you want to run certain events in the main flow without waiting for the subflow to complete its execution, you can do this using the **Detached Invocation** toggle on the **Settings** tab of the **StartaSubFlow** activity. When you set this **Detached Invocation** toggle to true, the **Output** option is not available in the **StartaSubFlow** activity window, and without waiting for the subflow output, the main flow is executed.

Creating a Flow Execution Branch

Activities in a flow can have one or more branches. If you specify a condition for a branch, the branch runs only when the condition is met. You also have the option to create an error branch from an activity. The purpose of the error branch is to catch any errors that might occur while running an activity. Branching is also supported for **Error Handler** flows, to catch all errors at the flow level.

i Note:

- You cannot create a branch from a trigger or a **Return** Activity.
- All activities that come after a branch are run irrespective of how the branch condition evaluates.

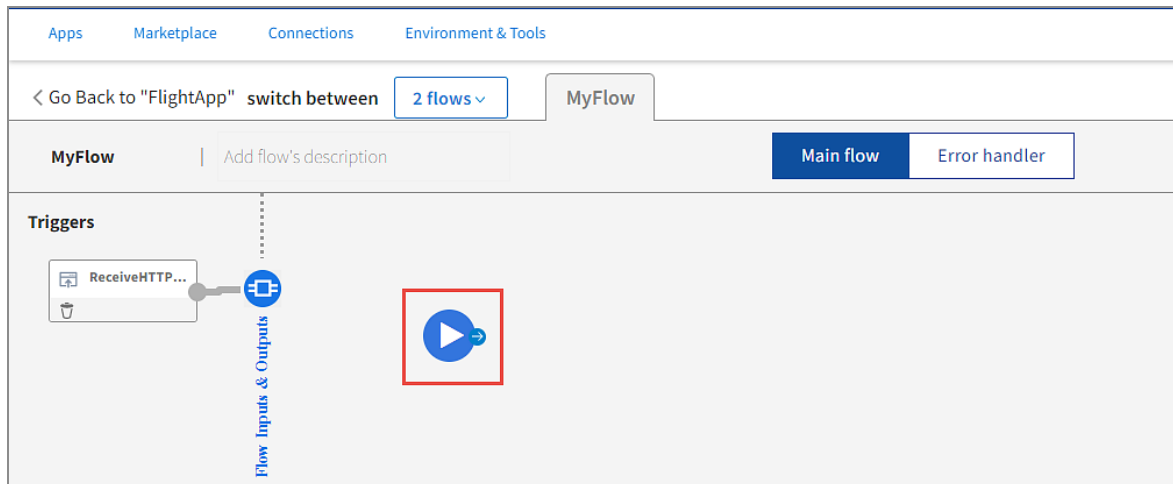
A **Return** activity ends the flow execution. Regardless of where the **Return** activity is placed in the flow, the flow execution exits the process as soon as it encounters a **Return** activity anywhere in the flow.

i Note: A **Return** Activity is not added by default. Depending on your requirements, you must add the **Return** Activity manually. For example, if any trigger needs to send a response back to a server, its output must be mapped to the output of the **Return** Activity in the flow.

To create a flow execution branch:

Procedure

1. From the **Apps** page, click the app name then click the flow name to open the flow details page.
2. For a start branch, drag a connection line from the a blue arrow on **StartActivity** icon to the desired activity that you want to start the execution with.



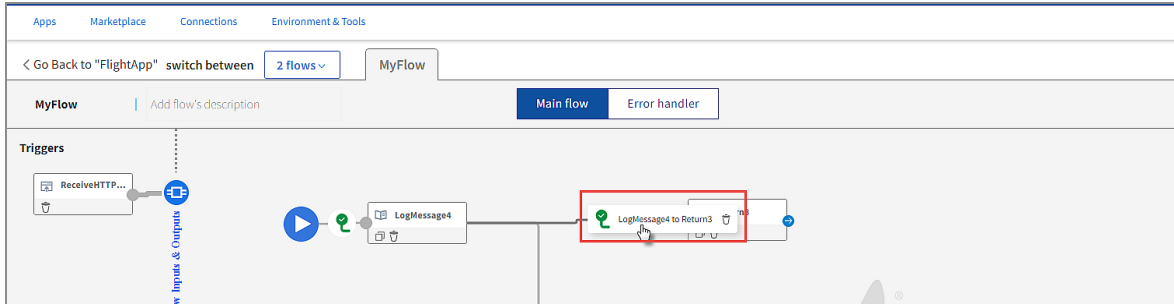
A branch gets created.

Each branch has a label associated with it. The label has the following format:

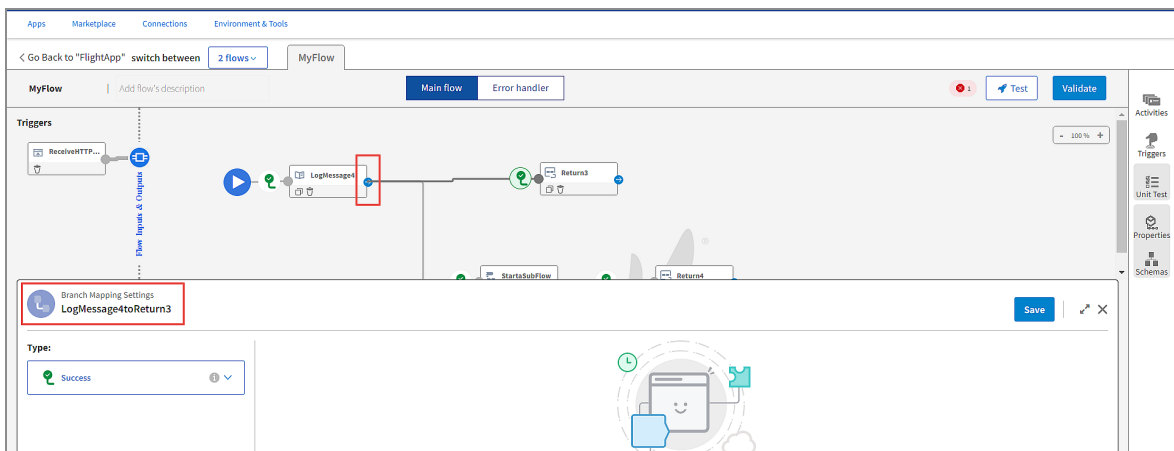
When branching to a specific activity:

<Name of activity in main flow>to<Name of activity in branch>

For example, LogMessageToInvokeRESTService.



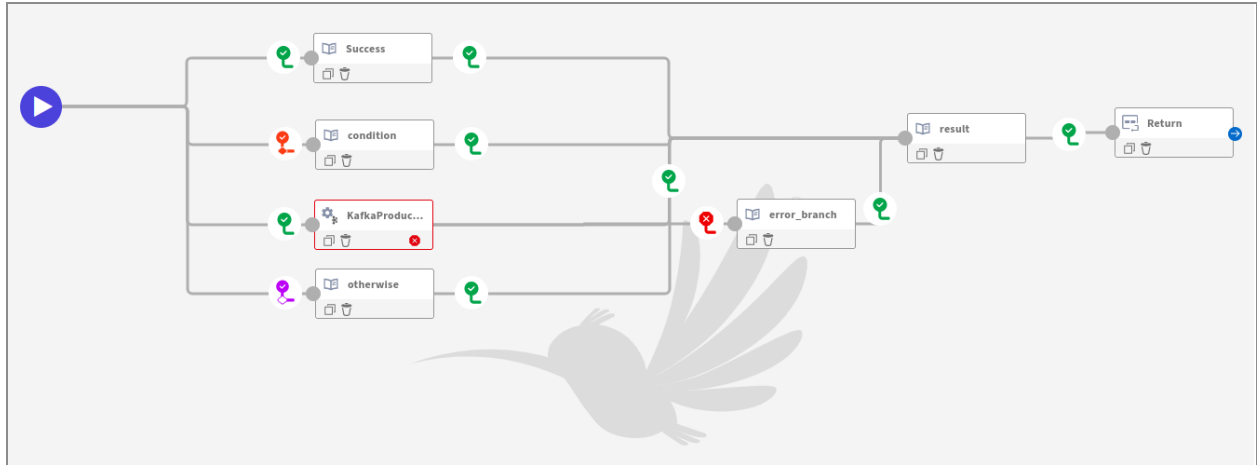
3. You can add a branch between the two activities. Hover over the activity that you want to start with and drag a connection line to the activity you want to connect to.
4. Clicking the branch opens the **Branch Mapping Settings** dialog.



5. Select either of the branch conditions: **Success**, **Success with condition**, **Success with no matching condition**, or **Error**. See [Types of Branch Conditions](#) for details on the conditions.
6. Click **Save**.
7. Add a condition to a branch as required. See [Setting Branch Conditions](#) for details.
8. If you want the flow execution to end after this branch is run successfully, add and configure the **Return** activity at the end of the branch. If you do not want the flow execution to end, do not add a **Return** activity at the end of the branch.

Joining or merging branches

You can now connect multiple activities to a single activity. In this case, an activity is executed only after all connected activities are either executed or skipped due to conditional branch.



Types of Branch Conditions

Flogo Enterprise supports multiple types of branch conditions.

Select one of the following conditions during branch creation:

- **Success**

A success branch is run whenever an activity is run successfully. If there is an error in the activity completion, this branch does not run. The branch has no conditions set in it.

- **Success with condition**

Select this condition if you want a branch to run only when a particular condition is met. If you select this condition and do not provide the condition, the branch never runs.

You can form an expression using anything available under upstream activity outputs and available functions, which should evaluate to a boolean result value.

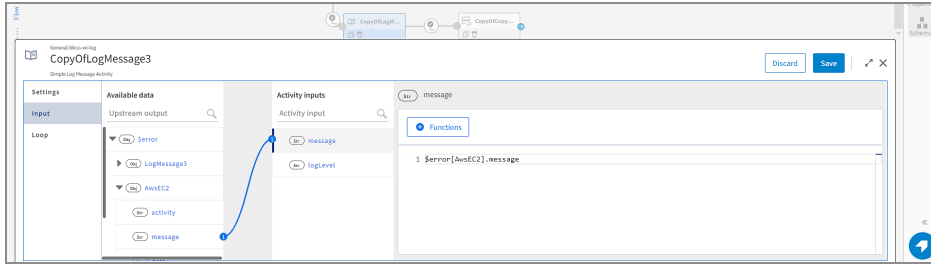
- **Success with no matching condition**

This branch condition is displayed only when you already have an existing **Success with condition** branch.

- **Error**

A branch with this condition runs if there are errors in completion of the activity. An activity can have only one **Error** branch.

Details of the error, such as the Activity and the type of the error message, are returned in `$error`. For example:



The **Error** branch flow differs from the error handler flow. In the error branch, the error branch is designed to catch exceptions at the activity level from which the error branch originates. Whereas the error handler flow is designed to catch exceptions that occur in any activity within the flow. So, if you handle the errors by creating an error branch at the activity level, the flow execution control never transfers to the error handler flow.

Order in which Branches are Run

When an Activity has multiple branches, regardless of the number of branches or the order in which the branches appear in the UI, the branch execution follows a pre-defined order.

i Note: The flow execution ends if it encounters a **Return** activity at any branch. In such situations, the activities that are placed after the return activity are not run.

The order in which the branches are run is as follows.

1. **Success with condition** branch

This branch runs only if its branch condition is met.

2. **Success with no matching condition** branch

This branch condition is displayed only when there is at least one existing **Success with condition** branch for the Activity. The **Success with no matching condition** branch is typically used when you want a specific outcome if none of the **Success with condition** branches meet their condition.

- This branch runs only if none of the **Success with condition** branches run. If the **Success with condition** branch runs and it does not have a **Return** Activity at the end of the branch, the flow execution control is passed to the success branch. If the **Success with condition** has a **Return** activity, the flow execution is ended after the **Success with condition** branch runs.

- If you delete all **Success with condition** branches without deleting the **Success with no matching condition** branch, you receive a warning informing you that the **Success with no matching condition** branch is orphaned.

3. **Success** branch

When an Activity has both **Success** and **Success with condition** branches, always the success with condition branch runs first. And if there are multiple success branches, the order of execution depends on the reverse order in which the each branch was created, that is, the success branch that was created at last is executed first.

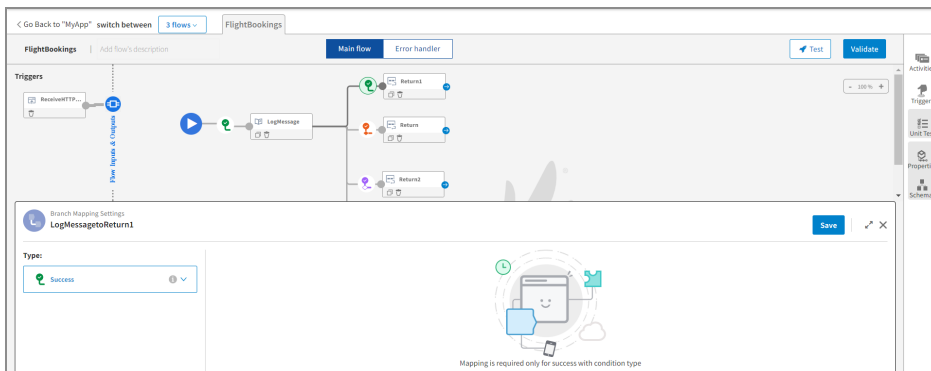
4. The **Error** branch is run as soon as the flow execution encounters an error.

Setting Branch Conditions

You can set conditions on a branch such that only if the condition is met the branch runs.

To set conditions on a branch:

1. Click the branch you want to set the conditions for. The **Branch Mapping Settings** dialog opens.



2. Select a branch condition: **Success**, **Success with condition**, or **Error**. If you already have a **Success with condition** branch present, you see **Success with no matching condition**.

See the section, [Types of Branch Conditions](#), for details on the three conditions.

3. Click **Save**.

4. If you selected **Success with condition**, the mapper opens for you to set the condition. Click the **condition**.

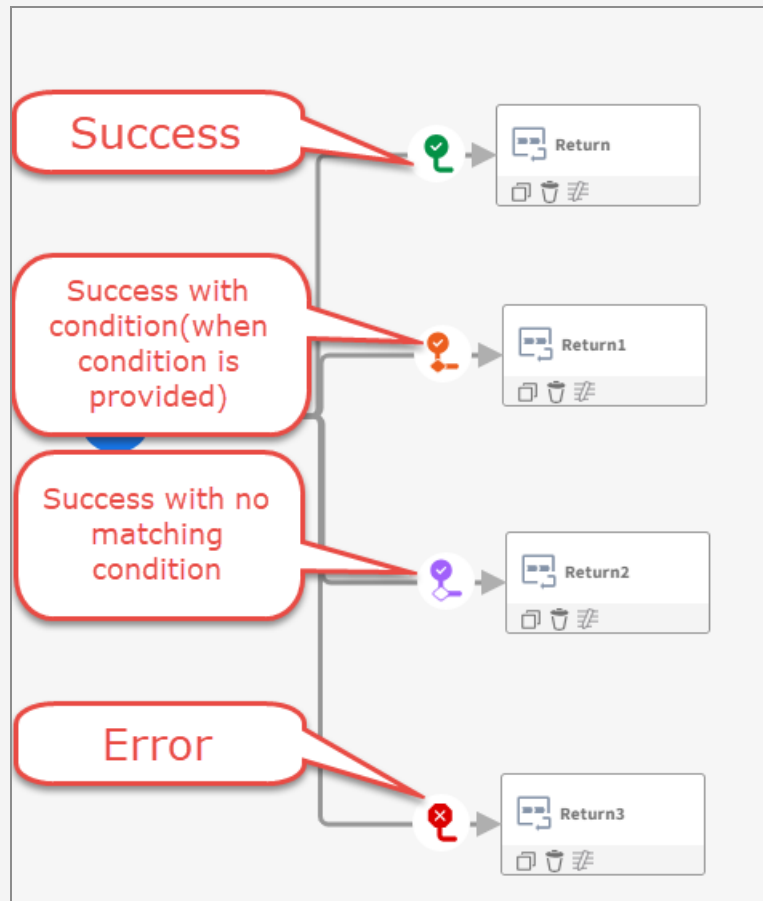
The mapper is exposed to the right of the dialog. The functions that you can use to

form the condition are shown under **Functions**.

5. Enter an expression with the condition or click a field from the output of a preceding Activity to use it. The output from preceding activities appears under the left **Upstream Output** in a tree format.

i Note:


- The condition must resolve to a boolean type. The following image shows how the branches appear based on the branch condition:

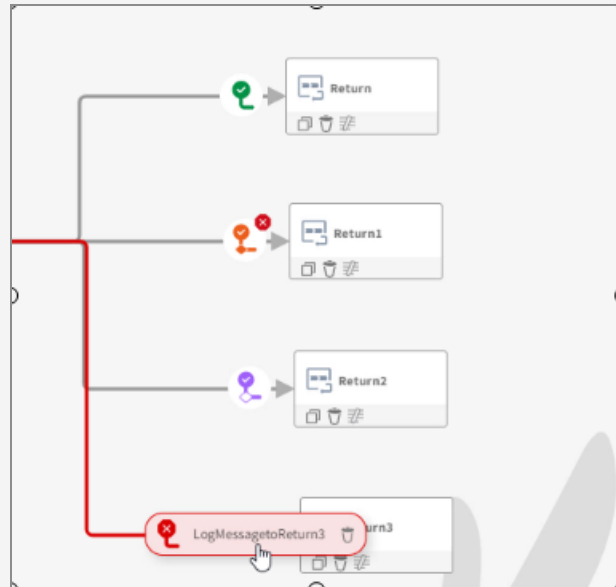


- When you hover over the branch lines or the branch labels, they appear in different colours according to the condition that is set.
 - Green - Success
 - Orange - Success with condition
 - Purple - Success with no matching condition
 - Red - Error.

These lines indicate the exact start and end points of the connection between any two activities. This is helpful in large and complex flows where the exact flow seems unclear and jumbled. The branch labels



indicate the names of the activities that are connected. You can rename the labels as per requirement. For the **success with condition** label, when it is empty or when there is a wrong condition, a  icon appears on it.




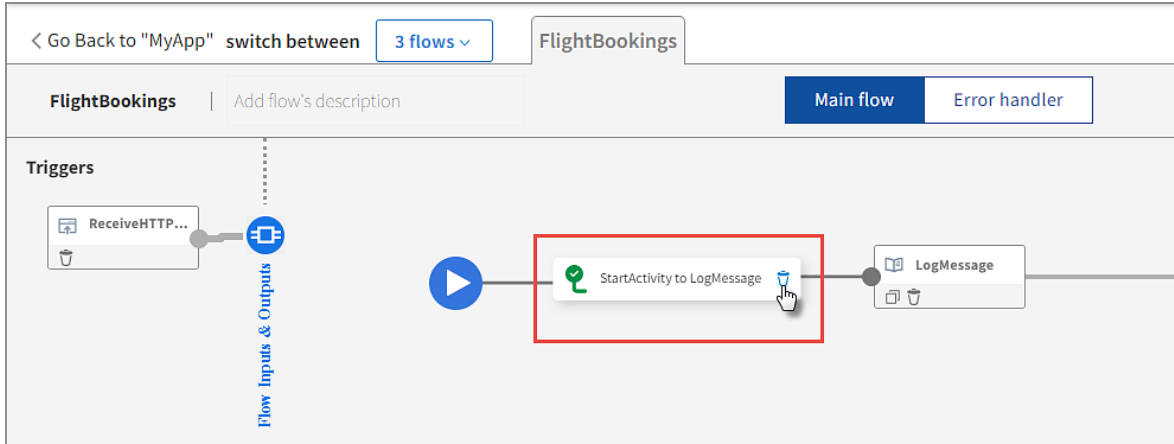
Deleting a Branch

You can delete a branch at any time after creating it.

To delete a branch:

Procedure

1. Hover over the branch that you want to delete. A branch label appears.
2. On the label, click  icon that appears.



3. On the confirmation dialog, click **Delete branch**. The selected branch is deleted.


Duplicating a Flow

You can duplicate an existing flow in an app. All activities in the flow along with their existing configurations get duplicated to a new flow in the app. The duplicate of the original flow gets created with a default name beginning with "Copy of" in the same app. You can rename the flow by clicking the flow name in the top-left corner of the flow details page. After you have duplicated the flow, you can add more activities, rearrange existing activities by dragging them to the desired location or delete activities from the flow duplicate.

Note: The triggers in the flow do not get duplicated. Also, if a flow has subflows, the subflows do not get duplicated.

To duplicate a flow:

Procedure

1. Open the **Apps** page and click the app to open the app details page.
2. Hover over to the extreme right of the flow that you want to duplicate until the **Duplicate flow** icon  displays.
3. Click the **Duplicate flow** icon. A duplicate of the flow gets created in the app.
4. Edit the duplicated flow as needed to add, rearrange, or delete activities in the flow and the app.

Editing a Flow

You can edit the flow name or its description after creating the flow. You can also add more activities. Rearrange existing activities by dragging them to the desired location or delete activities from the flow.

To edit a flow:

Procedure

1. On the **Apps** page, click the app name to open the app details page.
2. Click the flow name that opens the flow page. Rebuild the app after making the required changes.

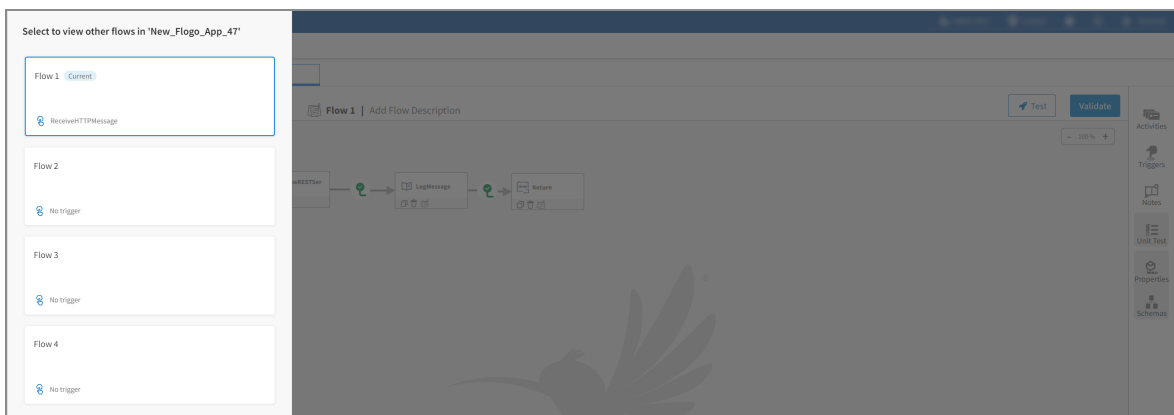
To edit the flow name, click anywhere in the flow name and edit the name. To add an activity between two existing activities, you can make a space by dragging the activities to anywhere you want in the activities area.

Switching Between Flows in an App

In an app that has multiple flows, you can switch between the flows within an app. There are two ways of doing it:

1. Using the **Flows** list dropdown:

Click the **Flows** list dropdown beside the flow name and select the flow you want to open.

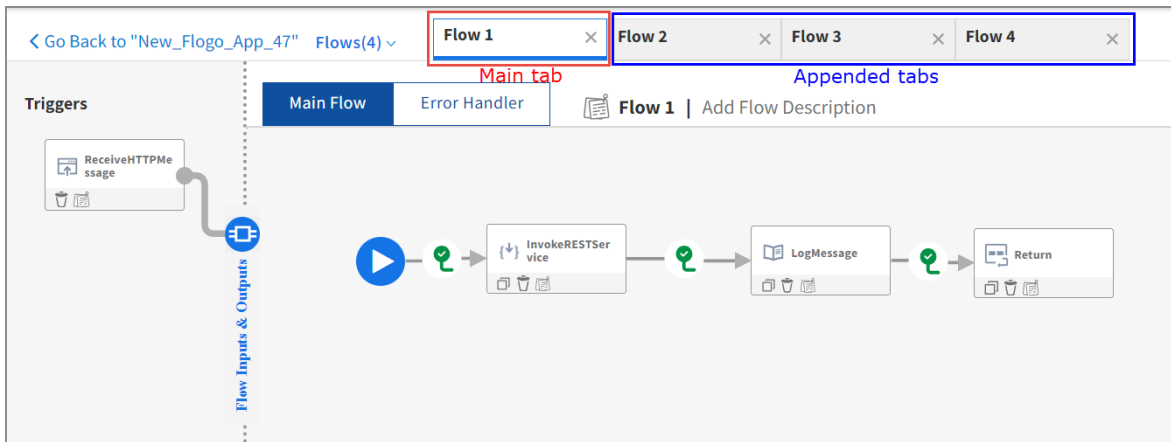


2. Using the flow tabs:

When you choose a flow from the **Flows** list dropdown, each flow appends to the right of the previously opened flow tab and this flow tab is set to active. The remaining tabs are inactive. You can simply click these flow tabs to switch between the flows.

You can also move a tab to the left or right of any existing tab.

Note: Only opened and appended tabs can be moved to the tabs section. The moving of tabs is applicable for the current instance only. For example, if you navigate back to the Flow List page and return to canvas, the state of the moved tabs is lost. Similarly, if you refresh the page, the state of the moved tabs is lost.



- Note:**
- If you try to open a flow tab that is already appended, then that flow tab is set to active wherever it is present.
 - Whenever you refresh the app, the order of the flow tabs remains same.

Caching of Flow tabs

When there are many flow tabs open, the first one is the main tab and the rest are appended tabs. In case you have to go back to the other flows and come back to the main flow tab again, the other appended flows remain there until closed. At a time, only one of the tabs remain active. For example, the **Flow 1** is the main flow tab and the rest are appended flow tabs.

i Note:


- A maximum of 10 flow tabs can be opened. You must close flows that are not required before opening a new flow.
- You cannot switch between flow tabs while configuring the App properties or Schema.
- During testing, the flow tabs are not accessible to user.

Deleting a Flow

You can delete a flow from the app details page.

To delete a flow:

Procedure

1. On the **Apps** page, click the app name to open its app details page.
2. Hover over to the extreme right of the flow name that you want to delete until the **Delete flow** icon  displays.
3. Click the **Delete flow** icon.
4. On the confirmation dialog, click **Delete**. The selected flow is deleted.

- i Note:** If multiple flows are attached to a trigger only the specific flow gets deleted. If there is only one flow attached to the trigger, the trigger also gets deleted.

Adding an Activity

After a flow is created, you must add activities to the flow.

Procedure

1. From the **Apps** page, click the app name then click the flow name to open the flow details page.

2. Click the **Activities** palette available on the right side. The categories of the activities are displayed.
3. Click the category from which you want to add an activity. For example, to add a general activity such as **Log Message**, click the **General** category.
4. Drag the required activity to the activities area.
5. To change the order in which the activities appear in the flow, you can drag the activity anywhere in the activities area.
6. Click the activity to open its configuration dialog and configure it.

✔ **Tip:** If you want to add an activity in between two activities, you can directly drop the activity on the branch label in between the two activities. You need not delete the incoming and outgoing connections and reconnect them. Adding an activity between two activities is only possible when an activity is dragged and dropped from the **Activities** panel, not for the activities already present on the flow canvas.

Searching for a Category or Activity

You can search an activity or category by entering the activity or category name in the **Search** box of the **Activity** palette.

You can enter either the full or partial name (a string of characters appearing in the name) of the activity or category in the **Search** box.

- All categories whose names either wholly match the search string or contain the partial search string in their name get displayed.
- Only those activities in the category whose names contain the search string are displayed in the search results. The activities in the category whose names do not match or contain the search string are not displayed in the search results.
- For any activity whose name wholly or partially matches the search string, the category that contains that activity is displayed. For example, if you enter "delete" in the search box, since there are activities whose name contains the string "delete" in Marketo, Salesforce, Zoho-CRM, all these categories are displayed, even though the category names themselves do not contain the string "delete".

Configuring an Activity

After adding an activity, you must configure it with the required input data. Also configure the output schema for activities that generate an output.

There are three ways to configure data for an activity:

- Configuring static data where you manually type the data in the mapper for the field. For example, type in a string that you want to output. Strings must be enclosed in double quotes. Numbers must be typed in without quotes.
- Mapping an Activity input to the output from one of the activities preceding it in the flow, provided that the previous activities have some output.
- Using functions. For example, the concat function to concatenate two strings.

To configure an activity:

Procedure

1. On the flow details page, click an activity.

The configuration box opens beneath the activity.


The screenshot shows the configuration interface for a LogMessage activity within a flow named "FlightBookings". At the top, there are navigation options: "< Go Back to 'MyApp'" and a dropdown menu showing "3 flows". The current flow is "FlightBookings", and there are tabs for "Main flow" and "Error handler". Below this, the "Triggers" section shows a "ReceiveHTTP..." activity connected to a "LogMessage" activity. A vertical dashed line separates the triggers from the "Flow Inputs & Outputs" section. The "LogMessage" activity is highlighted with a red box, and its configuration panel is open below it. The configuration panel has a "Settings" tab and shows the following options: "Log Level" set to "INFO", "Add Flow Details" set to "False" (with "True" also visible), and "Input" and "Loop" sections which are currently empty.

2. Click each tab in the configuration box under the activity name and either manually enter the required value, use a function, or on the **Input** tab, map the output from the trigger or a preceding activity using the mapper. Refer to the [Mapper](#) section for details on mapping.

If one or more activities are not configured properly in a flow, the error or warning icon is displayed in its upper-right corner. Click the activity whose tab contains the error or warning. For more details, see the [Errors and Warnings](#).


Duplicating an Activity

You can duplicate an activity within the same flow. The activity along with the existing configuration is duplicated into a new activity. The duplicate of the original activity is created with a default name beginning with `CopyOf`. You can rename the activity by clicking the activity name. Duplicating an activity saves you time and effort in situations when you want to create an activity with similar or the same configurations as an existing activity in the flow. After you duplicate the activity, you can change the configuration, move it around in the flow by dragging and dropping it to the required location or delete it from the flow.

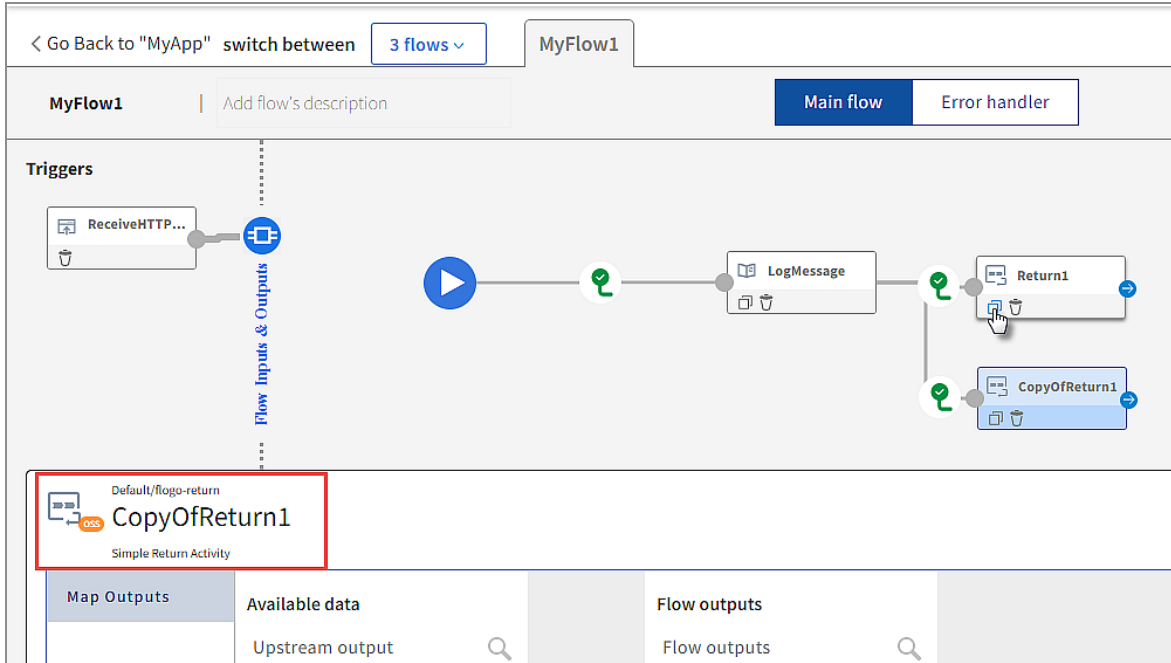
 **Note:** A trigger within a flow cannot be duplicated.

To duplicate an activity:

Procedure

1. From the **Apps** page, click the app name and then click the flow name to open the flow details page.
2. Hover over the activity that you want to copy and click .

For example, in the following screenshot, the **Return** activity is duplicated and added to the flow. The duplicate activity is called **CopyofReturn**:



3. Configure the duplicated activity as required.

Using the Loop Feature in an Activity

When creating a flow, you may want to iterate a certain piece of logic multiple times. For example, you want to send an email about an output of a certain activity *activity1* in your flow to multiple recipients. To do so, you can add a **SendMail** activity following *activity1* in your flow. Then configuring the **SendMail** activity to iterate multiple times when *activity1* outputs the desired result. Each iteration of the **SendMail** activity is used to send an email to one recipient.

Keep the following in mind when using the Loop feature:

- Iteration is supported for an activity only. You configure the iteration details on the **Loop** tab of the activity.
- The **Loop** tab is unavailable for certain activities that do not require iteration. For example, the **Return** activity. Its purpose is to exit the flow execution and return data to the trigger.
- You cannot iterate through a trigger.
- For apps that were created in Project Flogo and imported into Flogo Enterprise, the key type on the **Loop** tab is converted from the string to the relevant data type of

value in Flogo Enterprise.

To configure multiple iterations of an Activity:

Procedure


1. Click the Activity in the flow to expose its configuration tabs.
2. Click the **Loop** tab.
3. Select a type of iteration from the **Type** menu.

The default type is **None**, which means the Activity does not iterate.

Iterate

This type allows you to enter a number that represents the number of times you would like the Activity to iterate without considering any condition for iterating.

Click **iterator** to open the mapper to its right. You can either enter a number (integer) to specify the number of times the activity must iterate or you can set an expression for the loop by either entering the expression manually or mapping the output from the preceding activities or triggers. You can also use the available functions along with the output from previous activities and/or manually entered values to form the loop expression. The loop expression determines the number of times the activity iterates.

 **Warning:** The loop expression must either return a number or an array. The array can be of any data type. If your loop expression returns a number, for example 3, your activity iterates three times. If your loop expression returns an array, the activity iterates as many times as the length of the array. You can hover over the expression after entering the expression to make sure that the expression is valid. If the expression is not valid, a validation error is displayed.

If you select this type, the **Input** tab of the Activity displays the `$iteration` scope in the output area of the mapper. `$iteration` contains three properties, **key**, **index**, and **value**. **index** is used to hold the index of the current iteration. The **value** holds the value that exists at the index location of the current iteration if the loop expression evaluates to an array. If the loop expression evaluates an array of objects, **value** also displays the schema of the object. If the loop expression evaluates to a number, the **value** contains the same integer as the **index** for each iteration. To examine the result of each iteration of the Activity, you can map the **index** and **value** to the **message** input property in the **LogMessage** Activity and print them. The **key** is

used to hold the element name when configuring a condition if the value evaluates to an object. However, you can map only to the output of the last iteration if you did not set the **Accumulate Output** checkbox to **Yes**. See the [Accumulating the Activity Output for All Iterations](#) section for more details on this.

Repeat while true

Refer to [Flow Design Concepts sample](#) for an example of how to use this feature.

Select this type if you want to set up a condition for the iteration. This acts like the `do-while` loop where the first iteration is run without checking the condition and the subsequent iterations exit the loop or continue after checking the condition. You set the condition under which you want the activity to iterate by setting the **condition** element. The condition gets evaluated before the next iteration of the activity. The activity iterates only if the condition evaluates to true. It stops iterating once the condition evaluates to false. Click **condition**, and manually enter an expression for the condition. For example, `$iteration[index] < 5`.

Keep in mind that the index for the **Repeat while true** iteration begins at zero and iterates $n+1$ times. If you enter 4 as the iterator value, it runs as the following iterations: 0,1,2,3,4.

By default, the results of only the final iteration are saved and available. All previous iteration results are ignored. If you would like the results of all iterations to be stored and available, set **Accumulate** to **Yes**.

You have the option to set a time interval (in ms) between each iteration, which can help you manage the throughput of your machine. To spread the iterations out, set the **Delay** element. The default delay time is 0 ms, which results in no delay.

Result

After you enter the loop expression, the loop icon appears on the top-right corner of the activity.

Accumulating the Activity Output for All Iterations

When using the Loop tab to iterate over an Activity, you have the option to specify if you want the Loop to output the cumulative data from all iterations. You can do so by setting the **Accumulate** checkbox to **Yes**.

When the **Accumulate** checkbox is set to **Yes**, the activity accumulates the data from each iteration and outputs that collective data as an array of objects. Here, each object contains

the output from the corresponding iteration. The accumulated results are displayed as an array in the downstream activities in the mapper and be available for mapping.

When mapping to an element within an object in the output array of the activity, you must provide the index of the element to which you want to map. For instance, when you click a property within the object under **responseBody**, the expression displayed in the mapper is `$activity [<activity-name>] [<<index>>].responseBody.<property-name>`. Replace `<<index>>` with the actual index of the object to whose property that you want to map.

When the **Accumulate** checkbox is not selected, the output of the Loop displays an object that contains only the data from the last iteration. Data from all previous iterations is ignored. When mapping to an element in the output object of the activity, when you click a property within the object under **responseBody**, the expression displayed in the mapper is `$activity [<activity-name>].responseBody.<property-name>`.

The **Output** tab of the activity changes based on your selection of the **Accumulate** checkbox. The parent element (the name of the activity and the data type of the iteration output) is displayed regardless of your selection. If you set the **Accumulate** checkbox to **Yes**, the data type of the parent element is an array of objects. If you did not select the checkbox, the data type of the parent element is an object. The **Output** tab contents are also available in the mapper allowing for the downstream activities to map to them.

Accessing the Activity Outputs in Repeat While True Loop

This feature is useful when an activity needs to use the loop feature to do batch processing or fetch multiple records by running the activity multiple times. With each iteration of the activity, the output is available for mapping to the activity input.

This feature is available in all activities that generate an output (have an **Output** tab).

To use this feature:

Procedure

1. On the **Loop** tab, set the **Type** to **Repeat while true**.
2. Set the **Access output in input mappings** to **Yes**.

This makes the output of the activity iteration available in the **Upstream Output** for mapping. Now you can map your output as a next input parameter.

3. Enter a **condition** in its text box. The activity evaluates this condition before each run. If the condition evaluates to true the activity runs.

i Note: The output is only available in subsequent iterations after the first iteration. Since the activity output is not available for the first iteration, your condition must perform a check to see if it is the first iteration of the activity.

For example, use `$iteration[index] > 0 && isdefined($activity[SFQuery].output.locator)` to begin your condition. The `$iteration[index] > 0` checks to make sure that it is not the first run of the activity. The `isdefined($activity[SFQuery].output.locator)` function checks whether the output field exists.

Using the Retry On Error Feature in an Activity

Using the **Retry on Error** tab, you can set the number of times the flow tries to run the activity on encountering an error that can be fixed on retrial. The errors such as waiting for a server to start, intermittent connection failures, or connection timeout can be fixed on retrial.

You can set the count and the interval in one of the following ways:

- Manually type the value in the mapper
- Map the value from the previous Activity
- Select a function from the list of functions
- Map app property to override the values

Field	Description
Count	The number of times the flow should attempt to run the activity. This value must be an integer.
Interval (in millisecond)	The time to wait in between each attempt to run the activity. This value must be an integer.


i Note: The **Count** and **Interval** fields are mandatory. By default, the values are set to 0.

Deleting an Activity

You can delete an activity in a flow from the flow details page.

To delete an activity:

Procedure

1. On the **Apps** page, click the app name then click the flow name to open the flow details page.
2. Hover over the activity that you want to delete and click  icon.

Triggers

Triggers are used to activate flows. This section contains information on creating and managing triggers in your app.

Creating a Trigger without a Flow

You have the option to either create a trigger as a part of the process of creating a flow or you can create a trigger without creating a flow.

Refer to the section, [Creating a Flow](#), to create a trigger during the flow creation process.

To create a trigger without creating a flow, follow the steps below:

Procedure

1. On the app details page, click **Create**.
The **Add Triggers and Flows** dialog opens.
2. Under **Create new**, click **Trigger** to select it.
The triggers catalog opens to the right.
3. Select the trigger that you want to create in the triggers catalog.
The trigger gets created with a placeholder for a flow attached to it.

Deleting a Trigger

You can delete a trigger from the app details page by hovering over the trigger and clicking **Delete**.

Synchronizing a Schema Between Trigger and Flow

If you make any changes to the schema that you entered when creating the trigger, you must explicitly save any changes you make, then propagate the changes to the flow input and flow output. This is done by synchronizing the schemas.

To synchronize the schema between the trigger and the flow:

Procedure

1. Click the trigger to open its configuration details.
2. Make your changes and click **Save**. If you do not click **Save**, a warning message is displayed asking you to first save your changes before the schema can be synchronized.
3. Click **Sync** on the top-right corner.

The trigger output schema is copied to flow inputs and the trigger reply schema is copied to flow outputs.

Data Mappings

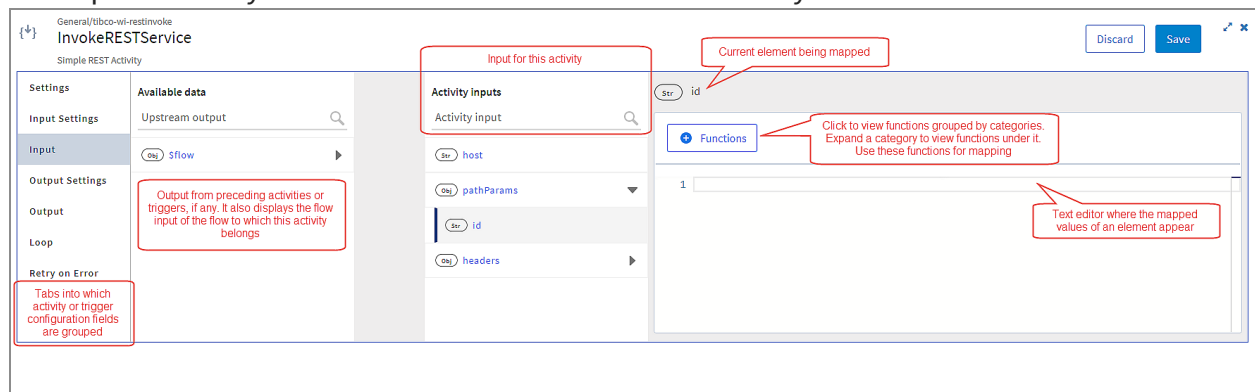
Flogo Enterprise provides a graphical data mapper to map data between the activities within a flow, and between the trigger and the flows attached to the trigger within an app. Use the mapper to enter the flow or Activity input values manually or map the input schema elements to output data of the same data type from preceding activities, triggers, or the flow itself.

Data Mappings Interface

An Activity has access to the output data from the trigger to which the flow is attached. It also has access to the output from any of the activities that precede it in the same flow

provided that the trigger or Activity has an output. This data is displayed in a tree structure under **Available data** in the mapper. The input schema for the Activity is displayed in the **Activity inputs** pane to the right of the **Available data** pane. You can map data coming from the upstream output to the input fields of the Activity. Also, each Activity has access to the input fields of the flow to which the Activity belongs. You can enter the flow input schema on the **Input Settings** tab of the **Flow Inputs and Outputs** tab.

When you click an activity or trigger on the flow details page, the configuration page for that activity or trigger opens. The following image is an example of the configuration page that opens when you click the **InvokeRESTService** activity.



The left-most pane displays the tabs for the configuration fields for that Activity or trigger. Each Activity or trigger has one or more of the following tabs:

- **Settings**

For triggers, this tab is displayed as **Trigger Settings**. This tab shows the Activity settings, trigger settings, or handler settings.

- Activity settings are specific to the Activity.
- Trigger settings are specific to the particular trigger.
- Handler settings apply to a specific flow attached to the trigger. Each flow attached to the trigger can have its own handler settings.

- **Input Settings**

On this tab, you can enter the schema for the flow or Activity input.

- **Input**

This tab is displayed for activities and shows the schema that you entered on the **Input Settings** tab in a tree format. You can manually enter values for any elements in the input schema or map any input element to the output from previous activities or triggers on this tab.

- **Output Settings**

On this tab, you can enter the schema for the flow or Activity output.

- **Output**

This tab displays the schema that you entered on the **Output Settings** tab in a tree format. The schema displayed on this tab is set to read-only as it is for informational purposes only.

- **Map to Flow Inputs**

The settings on this tab must be configured only if your trigger has an output, for example, in the REST or GraphQL triggers. You manually enter or map the elements from the trigger output (schema set on **Output Settings** tab) to the flow input elements (schema entered on the **Input Settings** tab of the **Flow Inputs & Outputs** tab). This allows the output from the trigger to become the input to the flow.

- **Reply Settings**

This tab is applicable only to triggers that send replies to the caller, such as the REST or GraphQL triggers. You enter the trigger reply schema on this tab.

- **Map from Flow Outputs**

This tab is specific to triggers that need to send a reply to the caller, such as the REST or GraphQL triggers. You manually enter or map the elements from the output of the flow (schema set on **Reply Settings** tab) to the flow output elements (schema entered on the **Output Settings** tab of the **Flow Inputs & Outputs**). This allows the output of the flow to become the reply that the trigger sends back to the request that it receives.

- **Loop**



On this tab, enter the iteration details for activities that you want to iterate.

When mapping, you can use data from the following sources:


- **Literal values** - literal values can be strings or numeric values. These values can be either manually typed in or mapped to a value from the output of the trigger or a preceding activity in the same flow. To specify a string, enclose the string in double quotes. To specify a number, type the number in the text box for the field. Constants and literal values can also be used as input to functions and expressions.
- An input element that is directly mapped to an element of the same type in the **Available data**.
- Mapping using functions - the mapper provides commonly used functions that you

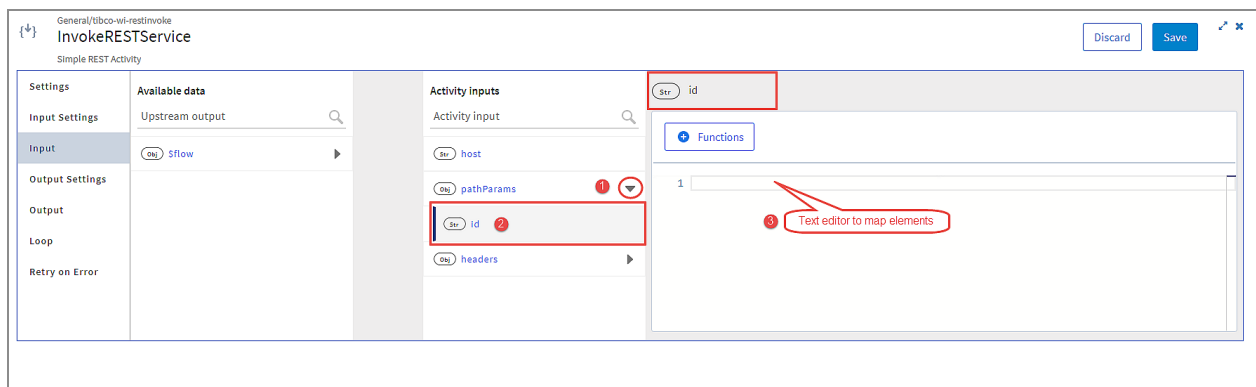
can use with the data to be mapped. The functions are categorized into groups. Click a function to use its output in your input data. When you use a function, placeholders are displayed for the function arguments. Click a placeholder argument within the function and drag an element from the **Available data** to replace the placeholder. Functions are grouped into logical categories. For more details, see [Using Functions](#).

- Expressions - you can enter an expression whose evaluated value is mapped to the input field. For more details, see [Using Expressions](#).


The error and warning icons are displayed on the **Activity inputs** pane, on the configuration fields in the left-most pane, and the activity tile. In case of errors in mapping (such as empty mandatory fields and incorrect mapping at activity or trigger level), an error icon  is displayed. A warning icon  is displayed if your changes are not saved or discarded, input and output are not mapped in triggers, or mappings are removed for mandatory fields.

Mapping Data from the Data Mappings Interface

In the following example, in the **Activity inputs** pane, clicking the arrow  expands the object **pathParams**. You can select the input (in this case, **id**) that you want to map. A section with a text editor opens on the right side in the mapper.



To map data coming from the upstream output to the input fields of the Activity:

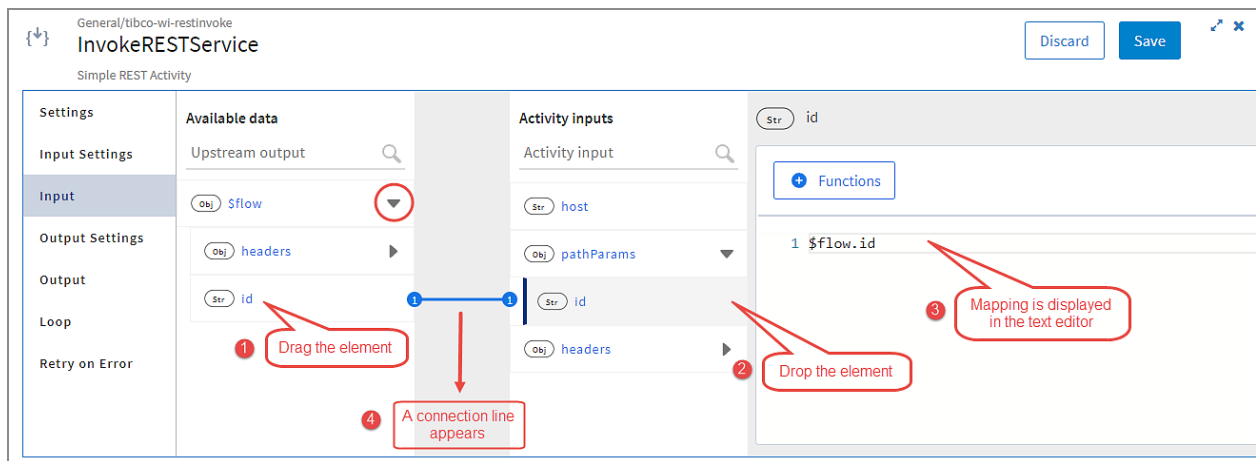
In the **Available data** pane, click the arrow  to view the fields. You can map an element from the **Activity inputs** pane to an element in the **Available data** pane using one of the following methods:

- Drag the element from **Available data** and drop it on the input in the **Activity inputs**

pane. The mapping is displayed in the text editor.

- Click the element from the **Activity inputs** pane. The text editor opens on the right side of the mapper. Drag the element from the **Available data** pane and drop it in the text editor.
- Click the element from the **Activity inputs** pane and double-click the element in the **Available data** pane to map it to the input.

A connection line appears to show the mapping between the **Available data** and the **Activity inputs**.



To add functions in the mapper, refer to the [Using Functions](#) section.

Connection Lines

Connection lines show the mapping between the data and the input. These lines appear when you map an element from the **Available data** with an element from the **Activity inputs**. The lines also appear for mapped arguments. When the mapped element is selected in the **Activity inputs** pane, the connection line is blue. Otherwise, it is gray. The numbers at the ends of a connection line indicate the total number of mapped elements for a particular element.

The following screenshot shows the connection lines and the total count of mappings for each element.

The screenshot shows the Tibco Flogo Mapper interface for a 'Mapper' activity. The 'Available data' pane on the left shows an array 'cakes' with properties 'id', 'type', 'name', and 'ppu'. The 'Activity inputs' pane on the right shows the same 'cakes' array being mapped to an 'id' property. A blue arrow indicates the mapping from the 'cakes' array to the 'id' property. The 'Functions' pane on the right contains the following code:

```
1 array.forEach($flow.body.cakes, "cakes", $loop.type=="donut")
```

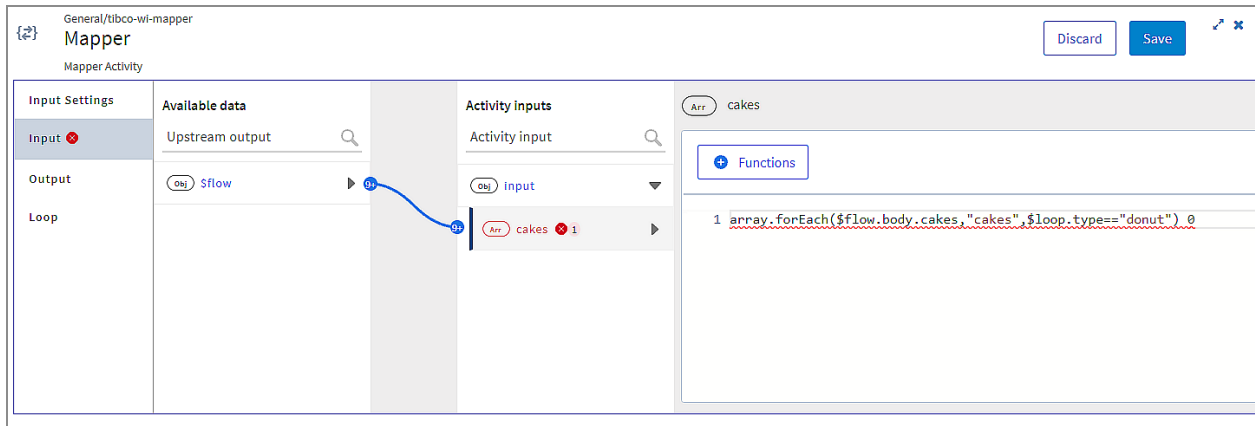
Errors in Mapping

In the mapper, you can see the total count of errors and warnings each in the mapping next to the parent object in the **Activity inputs** pane.

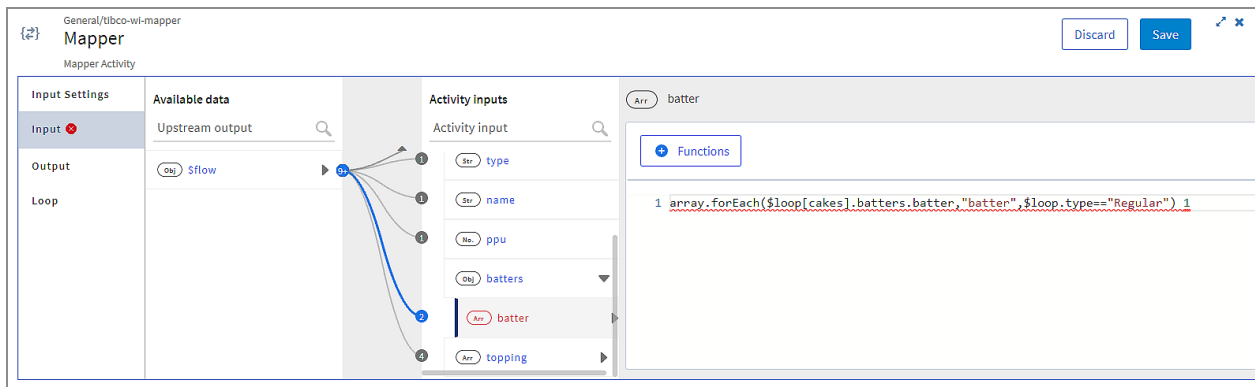
In the following example, the parent object **input** has a total of two errors in mapping.

The screenshot shows the Tibco Flogo Mapper interface for a 'Mapper' activity. The 'Available data' pane on the left shows an object 'Sflow'. The 'Activity inputs' pane on the right shows the 'Sflow' object being mapped to an 'input' property. The 'input' property is highlighted with a red box and has a red 'x' icon and the number '2' next to it, indicating two errors. The 'Input Settings' pane on the left shows the 'Input' property with a red 'x' icon.

Expanding the **input** object shows that the array **cakes** is mapped incorrectly. This also shows that **cakes** contains one element with incorrect mapping.



Expanding the array **cakes** shows that the array **batter** under the object **batters** has an error in mapping.



Note: The errors in mappings are also observed when the property in the app properties dialog is edited, moved from a group to another or from a group to top level as a standalone property. A warning message regarding the same pops up on the screen when you edit any properties.

Scopes in Data Mappings

The **Available data** pane in the mapper displays the output data from preceding activities, triggers, and flow inputs. This area groups the output elements based on a scope. A scope represents a boundary in the **Available data** within which an input element can be mapped. For example, when mapping an input element to an element from the output of a trigger, the scope of the input element is represented in **Available data** as **\$trigger**. The following scopes are currently supported by the mapper.

Scope Name	Used to...	Available in...
\$trigger	Map flow input to trigger output.	Trigger (Map to Flow Inputs tab) to map flow inputs to trigger outputs.
\$flow	Map flow output to trigger reply.	<ul style="list-style-type: none"> • Trigger (Map to Flow Outputs tab) to map flow output to trigger reply. • Activities (Input tab) to map Activity input to flow input. • Return Activity (Map Output tab) to map flow output to flow input.
\$Activity. [Activity-name]	Map input elements of the Activity to elements from the output of previous activities.	\$activity represents the scope of an activity. [activity-name] indicates the activity whose scope that you are defining. Each preceding activity has its own scope in the mapper.
\$iteration	Keeps record of the current iteration and is available only when the iterator is enabled for an activity on the Loop tab.	Input tab of an Activity that has Loop enabled. This tab is displayed only when the Loop for the Activity is enabled. The following elements are displayed under \$iteration : <ul style="list-style-type: none"> • key - This element represents the iteration index. Thus, it is always of type number. For example, if the Loop expression is set to an array, the key element represents the array index of the current iteration. • value - The value can be of any type depending on what is being iterated. For example, if you are iterating through an array of strings, the value is of type string.
\$property [property-name]	Map to app properties that are defined in the app.	For any app that has app properties defined, this scope is available for mapping from any activity that allows mapping. Even the app properties from the connection are available for mapping under this

Scope Name	Used to...	Available in...
		<p>scope.</p> <p>All the mapped configurations can be pre-checked using a flow tester or by creating a pre-check flow.</p>
\$loop	Map elements within an array.	\$loop is prefixed to the element name when mapping an element that is within an array. The scope of \$loop is the current array that you are iterating through.
\$flowctx	Map the flow context details to the current flow.	<p>Input tab of every activity. The scope provides flow context details that can be mapped to any activity that allows mapping. Using this scope, the unique parameter like FlowId, Flowname, ParentFlowId, ParentFlowName, SpanId, TraceId can be accessed in the flow and subflow.</p> <p>Here:</p> <ul style="list-style-type: none"> • The ParentFlowId and ParentFlowName is the ID and name of the flow that is invoking the current flow. • The TraceId is the unique ID of a single request, job, or an action initiated by the user. • The SpanId is the unique ID of the activity <p>Note: This scope is only available for the flow configuration and not for the trigger configuration.</p>

Data Types

Supported data types

The following data types are supported:

- BIT
- CHAR
- DECIMAL
- INTEGER
- TEXT
- NUMERIC
- REAL
- SMALLINT
- DATE
- TIMESTAMP
- MONEY
- ENUM
- JSON
- XML
- TINYINT
- VARCHAR
- SMALL MONEY

Unsupported data types

The following data types are not supported:

- BIGINT
- BINARY

Reserved Keywords to be Avoided in Schemas

Flogo uses some words as keywords or reserved names. Do not use such words in your schema. When you import an app, if the schema entered on the **Input** or **Output** tab of an

Activity or trigger contains reserved keywords, after the app is imported, such attributes are treated as special characters and might cause runtime errors.

Avoid using the keywords listed below in your schema:

- break
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else
- enum
- export
- extends
- false
- finally
- for
- function
- get
- if
- import
- in
- index
- instanceof

- new
- null
- return
- set
- super
- switch
- this
- Generate
- true
- try
- typeof
- var
- void
- while
- with

Mapping Different Types of Data

The mapper opens when you click any element in the input schema tree on an Activity configuration tab.

You can map the following elements:

- A single element from the input to another single element in the output.

i Note: If the single element comes from an array in the output, then you must manually add the array index to use. For example, `$flow.body.Account.Address[0] city`.

- A standalone object (an object that is not in an array).
- An array of primitive data type to another array of primitive data type.

- An array of non-primitive data types (object data type or a nested array) to another array of the same non-primitive data type.

Keep the following in mind when using the mapper:

- Make sure that you map all elements that are marked as required (have a red asterisk against them), whether they are standalone primitive types, within an object, or within an array. When mapping identical objects or arrays, such elements get automatically mapped, but if you are mapping non-identical objects or arrays, be sure to map the elements marked as required individually.
- The `in` and `new` attributes are treated as special characters if you use them in the schema that you enter in the REST Activity or trigger. For example, mappings such as `$flow.body ["in"]` and `$flow.body ["new"]` are not supported. If an imported app contains these attributes after the app is imported into Flogo. It results in runtime errors.
- Use of the anonymous array is not supported on the **Flow Input & Output** tab and the **Return** Activity configurations. To map to an anonymous array, you must create a top-level object or a root element and render that.
- You cannot use a scope (identified with a beginning `$` sign) in an expression, for example, `renderJSON($flow, true)`. You can use an object or element under it, for example, `renderJSON($flow.input, true)`.
- You can only map one element at a time.

- i Note:** If the output element names contain special characters other than an underscore (_), they appear in bracket notation in the mapping text box.

In the following example, **name** under **Available data** does not contain any special characters. Hence it is displayed in dot notation.

The screenshot shows the TIBCO Flogo configuration interface. On the left, under 'Available data', there is a list of data elements: 'Sflow' (Obj), 'name' (Str), 'strArray' (Str[]), 'numArray' (No.[]), 'boolArray' (Bool[]), and 'objArray' (Arr). A blue arrow points from the 'name' element to the 'name1' element in the 'Flow outputs' list. On the right, the 'name1' output is selected, and the mapping text box contains the expression `$flow.name`, which is highlighted with a red box.

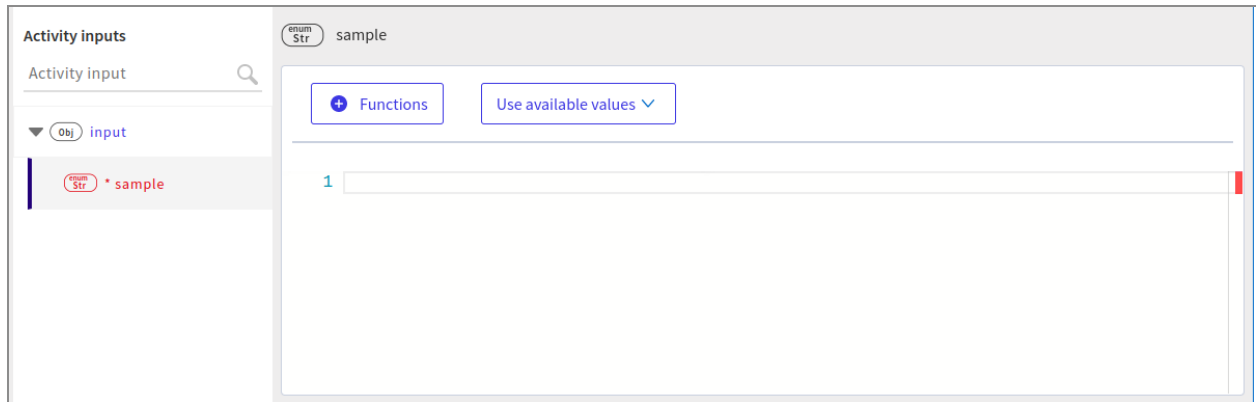
In the following example, **name 1** contains a space. Hence it appears in the bracket notation.

The screenshot shows the TIBCO Flogo configuration interface. On the left, under 'Available data', there is a list of data elements: 'Sflow' (Obj), 'name 1' (Str), 'strArray' (Str[]), 'numArray' (No.[]), 'boolArray' (Bool[]), and 'objArray' (Arr). A blue arrow points from the 'name 1' element to the 'name1' element in the 'Flow outputs' list. On the right, the 'name1' output is selected, and the mapping text box contains the expression `$flow["name 1"]`, which is highlighted with a red box.

Mapping an Enum value

You can map values of the Enum data type to the **Activity Inputs** element directly by selecting the values from the **Use available values** dropdown.

This feature is available for all Activities and Triggers that have a schema option.



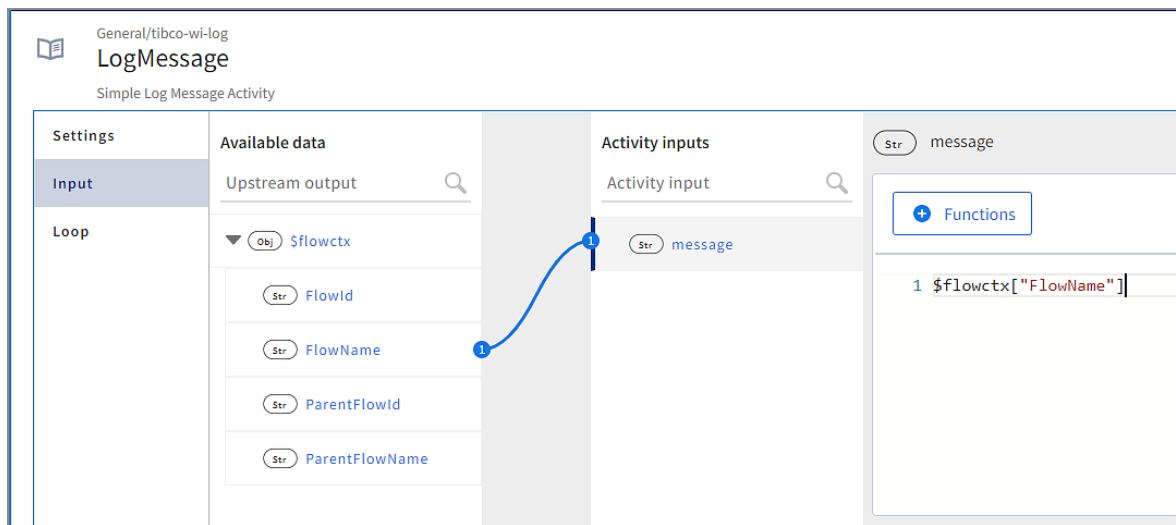
✔ **Tip:** Always use the **enum** keyword to identify the constant values.

Mapping a Single Element of Primitive Data Type

You can map a single element of a primitive data type to a single element of the same type in the output schema under **Available data**.

- Drag the element from the **Available data** and drop it on the destination element that you want to map in the **Activity inputs** pane.

In the following example, drag and drop **FlowName** (source) on **message** (destination) to map it. Alternatively, click **message**. Drag and drop **FlowName** in the text editor, or double-click **FlowName**.



Mapping an Object

Standalone objects (objects not within an array) whose property data types match can be mapped at the root level. If the destination object is identical to the source object under **Available data** (both, the names of the properties as well as their data types match exactly), you need not match the elements in the object individually. If the property names are not identical, then you must map each property individually within the object.

For example, in the image below the **Person** objects are identical. So, you can map **Person** to **Person**. You need not map name and age individually.

The screenshot shows the mapping interface with three panels:

- Available data:** Shows an upstream output of type `Obj` named `Sflow`. Underneath, there is an array `Person` containing two properties: `name` (type `Str`) and `age` (type `No.`).
- Flow outputs:** Shows a flow output of type `Arr` named `Person` containing two properties: `name` (type `Str`) and `age` (type `No.`).
- Mapping:** A blue line with a '1' at each end connects the `Person` array in the Available data panel to the `Person` array in the Flow outputs panel.
- Code Editor:** Shows the resulting mapping expression: `1 $flow.Person`.

In the following image, the data types match but the property names do not match. In such a case, you must map each property individually in addition to mapping the object root.

The screenshot shows the mapping interface with three panels:

- Available data:** Shows an upstream output of type `Obj` named `Sflow`. Underneath, there is an array `Person` containing three properties: `firstname` (type `Str`), `age` (type `No.`), and `name` (type `Str`).
- Flow outputs:** Shows a flow output of type `No.` named `age`. Underneath, there is an array `Person` containing two properties: `name` (type `Str`) and `age` (type `No.`).
- Mapping:** Four blue lines with '1' at each end connect the `Person` array in the Available data panel to the `Person` array in the Flow outputs panel. The lines connect `Person` to `Person`, `firstname` to `name`, and `age` to `age`.
- Code Editor:** Shows the resulting mapping expression: `1 $loop.age`.

Mapping Arrays

When mapping arrays, you must first map their array root before you can map their child elements.

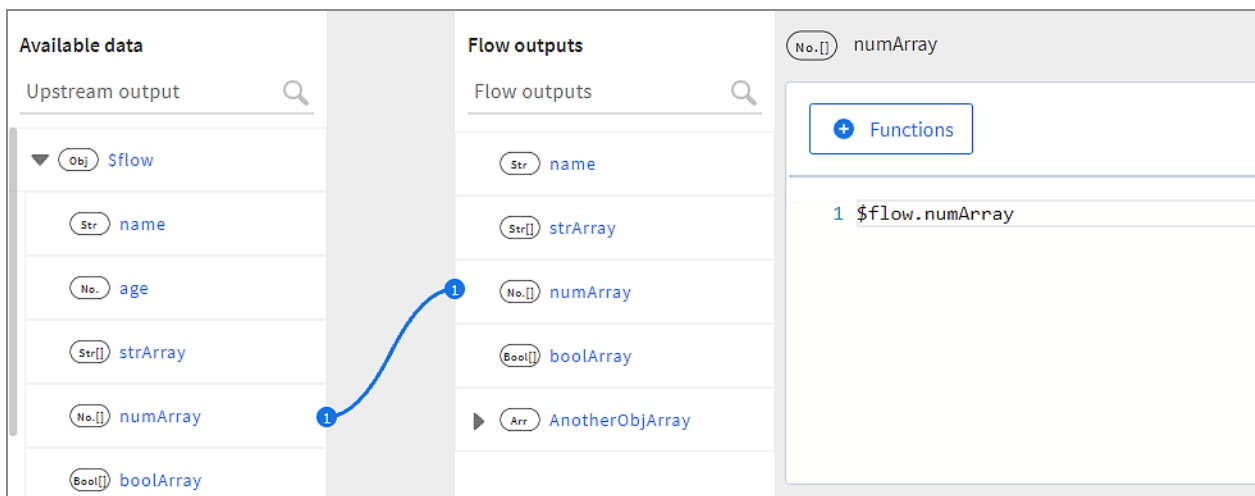
The following mappings are supported when mapping arrays.

- Mapping arrays of primitive data types
- Mapping an array of objects
- Mapping nested arrays

Mapping an Array of Primitive Data Types

To map arrays of the same primitive data type, you only need to map the array root. You do not need to map the array elements.

Here is an example of mapping arrays of primitive data types:



The array names need not match, but their data types must match. In **Available data**, `$flow` points to **numArray**, which is the scope for **numArray** in the input.

Creating Arrays When Data Types Do Not Match

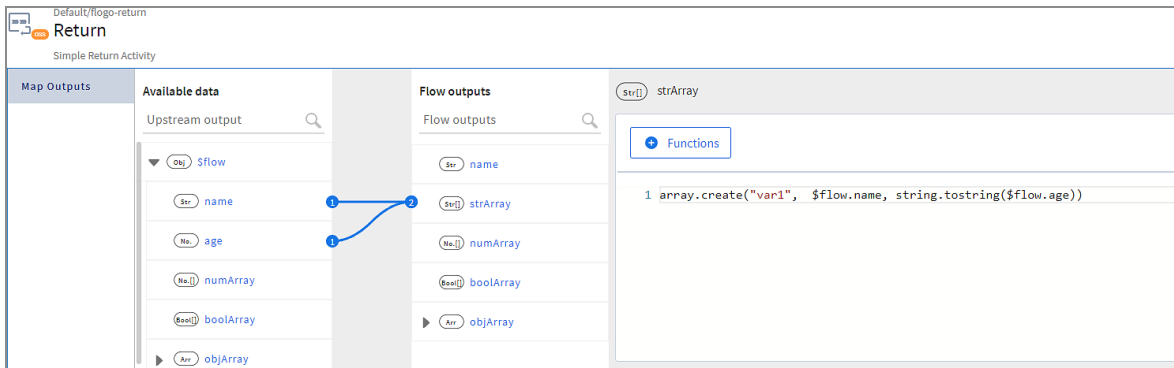
If you want to map an array of primitive data types, but you do not have an array of the same data type in your **Available data**, you can create an array using the `array.create(item)` function.

i Note: `array.create(item)` can only be used to create an array of primitive data types. You cannot use it to create an array of objects.

To do so:

1. Click the array for which you want to do the mapping in the input schema. The mapper opens to its right.
2. Click **Functions** and click **array** to expand it.
3. Click `create(item)`. It is displayed in the text editor.
4. Replace `item` with the output element to create the array.

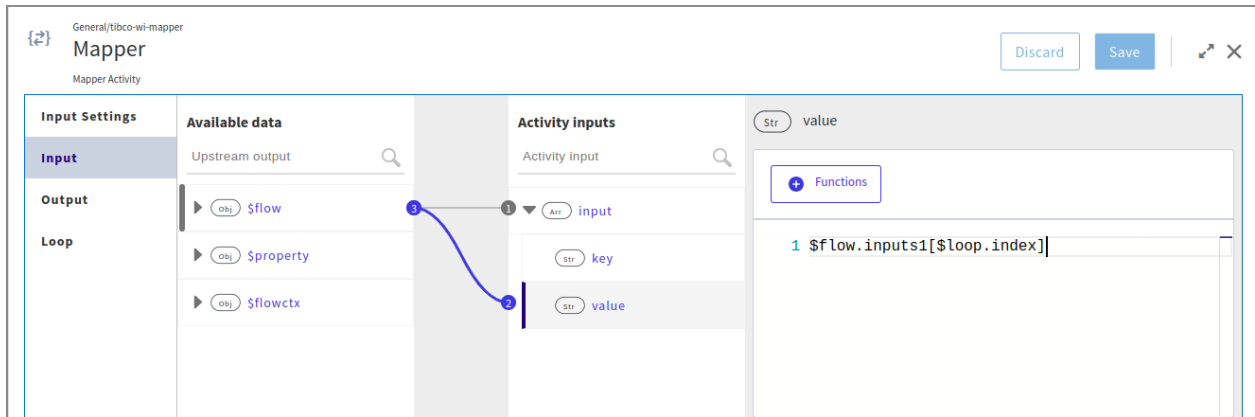
In the following image, to map **strArray**, create an array since there is no array of strings under **Available data**. The `array.create()` function accepts any of the following: a hardcoded string, an element from **Available data**, an expression, or a function as shown below as long as they all evaluate to the appropriate data type.



Iterating Over Arrays

To iterate over an array and map values in a loop, you can use `$loop.index`.

Here is an example of mapping values using array index:



Mapping Complex Arrays

Complex arrays are arrays of objects that can optionally contain nested arrays. You can map these arrays using the 3 available options - **Configure with Items**, **Configure with Source** and **Configure with JSON**.

For examples, refer to [arrayforEach sample](#).

When you use the **Configure with Items** option, you define an implicit scope consisting of everything available in the **Available data**. It is equivalent to creating an implicit array with a single object element consisting of everything in the **Available data**. Hence, the resulting length of the array is always one element.

To create a confined scope within the **Available data**, use the **Configure with Source** option. When using this option, you must map 3 fields - **Source**, **Loop name** and the **Filter by**. Here, the **Loop name** gets auto populated. When mapping identical arrays, the source name gets inserted in the **Select Source** field by default.

The **Source** defines the scope within the **Available data**. Simply put, the input object or array can only be mapped to elements in the **Available data** that fall within the boundary indicated by its scope.

The **Loop name** is a scoping variable given to the scope that you have defined in the first argument. By default, the scoping variable name is the same as the input element name for which you are defining the scope. By doing so, the mapper associates the input object to its scope by the scoping variable. Once there is a scoping variable for the scope, the mapper uses that scoping variable to refer to the scope in future mappings. You can edit the scoping variable to any string that might be more meaningful to you. The scoping variable is particularly useful when mapping the child elements in nested arrays.

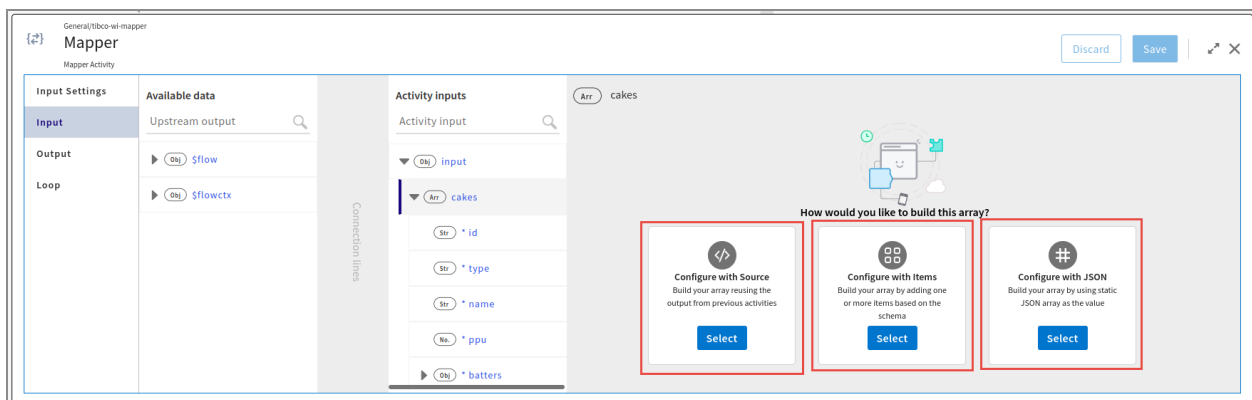
The **Filter by** field is optional. When iterating through an upstream output array, you can enter a filter to specify a particular condition for mapping as the **Filter by** field. When using this field, you must enter the scoping variable in the loop name field. Only array elements that match the filter get mapped. For instance, if you are iterating through an array, `array1`, in the upstream output with a filter `$loop.name=="Jane"` mapped in the **Filter by** field, if `array1` has 10 elements and only four out of them match the condition of the filter, only those four elements are mapped to the input array and the remaining six are skipped. This results in the size of the input array being only four elements, even though `array1` has 10 elements. See the section, [Filtering Array Elements to Map Based on a Condition](#) for more details.

i Note: If you have used the `array.forEach()` in a legacy app, to update your app with the current changes, delete the old mapping and remap the elements. A scoping variable is now included in the Loop name field. For example, if the old mapping is: `array.forEach($flow.body.Book)`, after remapping, `$flow.body.Book` is added to the Source field, where "Book" is added in the Loop name field, which is also the scoping variable.

i Note: If you use a function as a source array in the source field, the array element schema cannot be determined and a design-time validation error is returned. It is recommended that you use the mapper to define the function output schema and then use it in the source field.

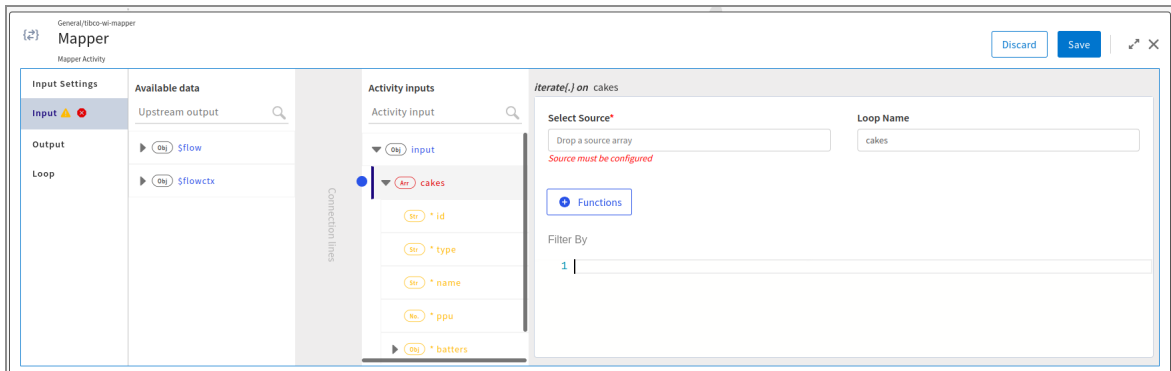
Mapping of unmapped arrays

With the support of first class `for.each()` in the Mapper activity, you can map elements to an unmapped array in 3 different ways.



- Using the **Configure with Source** option

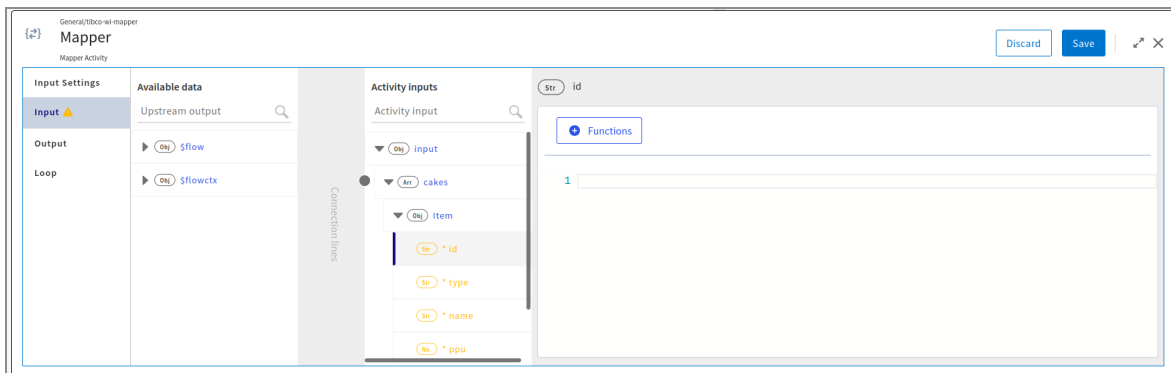
For mapping, double click the element from **Available data** array. You can also drag the element of the **Available data** array to the element of the **Activity inputs** array.



Note: To change the element that is already mapped, either drag another element or select the element from the source array.

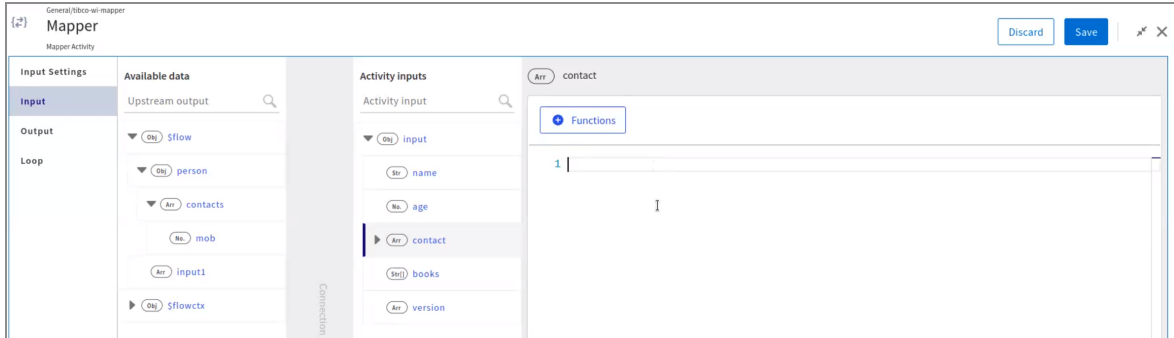
- Using the **Configure with Items** option

You can add the elements to your array manually.



- Using the **Configure with JSON** option

You can map the empty array by literal value mapping or type in the required expression.



Note:

- **Reset** option can be used to delete all the items from the array and set the array to default form.
- **Clear mappings** can be used to remove all the mappings on the item level.
- For an empty for.each() array, you can clear the mappings for child items only.

Add Items to Array

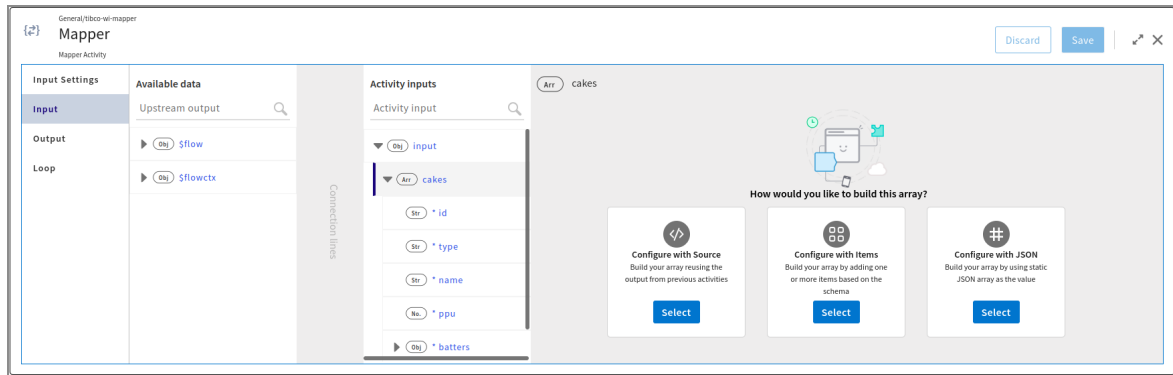
Now, when there is need to map more than one array object in same array, you can add items to the array. Each item can be mapped with different values.

For example, If one item is mapped with the flow input, the other can be mapped with literal values.

You can add an item to array:

1. For an unmapped array

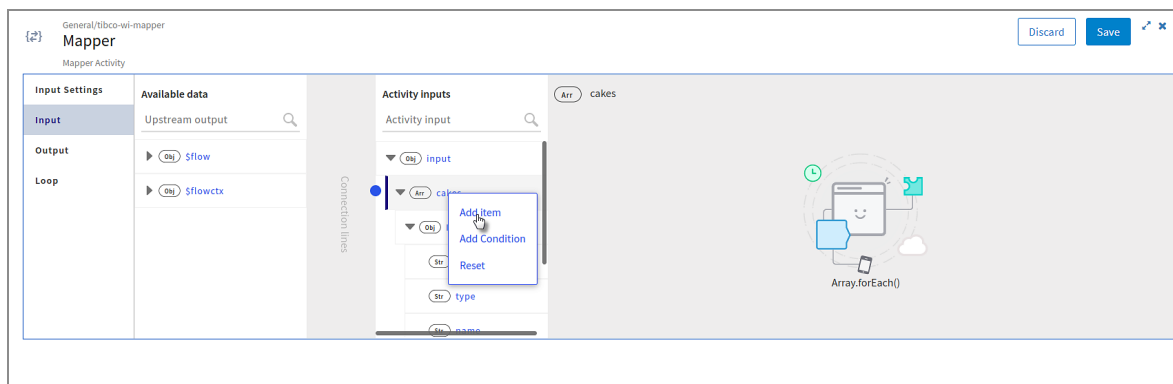
You can add an item in an unmapped array.



Note: Use the **Configure with Items** option for adding a single item.

2. For empty for.each() array

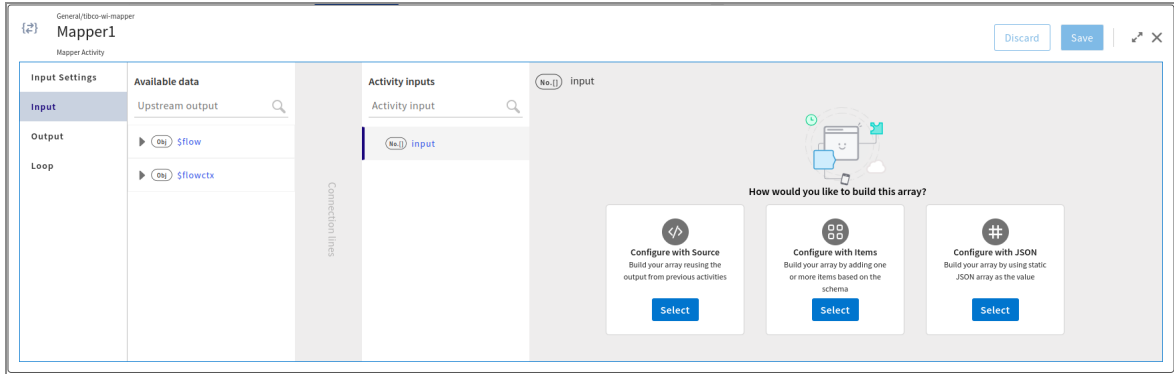
You can also add an item to an empty For.each() array.



- Note:**
- For all pre-existing array mappings with empty array.foreach() the properties are displayed as array item and the array level mapping is not editable.
 - On adding empty array.foreach(), input mapper at array level turns to non editable.
 - Elements under existing array mapping which has array.foreach() without source are wrapped in an item object.

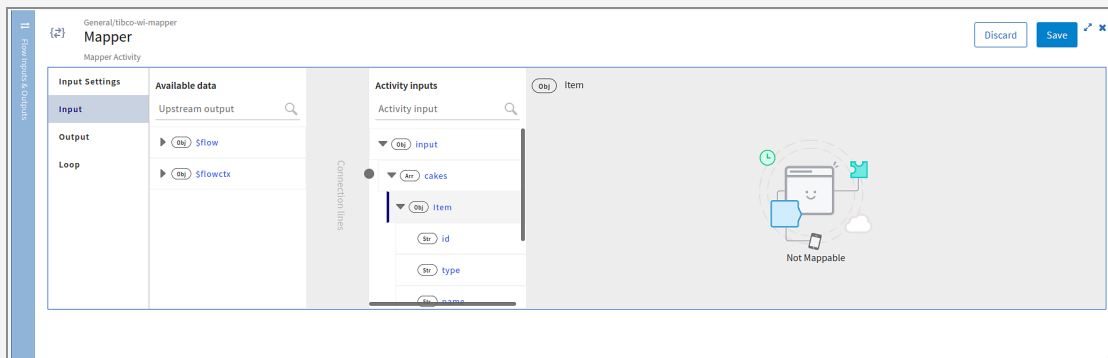
3. Primitive data type array

An item can even be added to an empty primitive data type array.



Note:

- **Add item** option is not available for an array of type 'any'.
- On importing an app with inline array mapping, array elements are wrapped in an item object.
- On adding items under an array, mapping cannot be done at the item-level



Mapping Identical Arrays of Objects

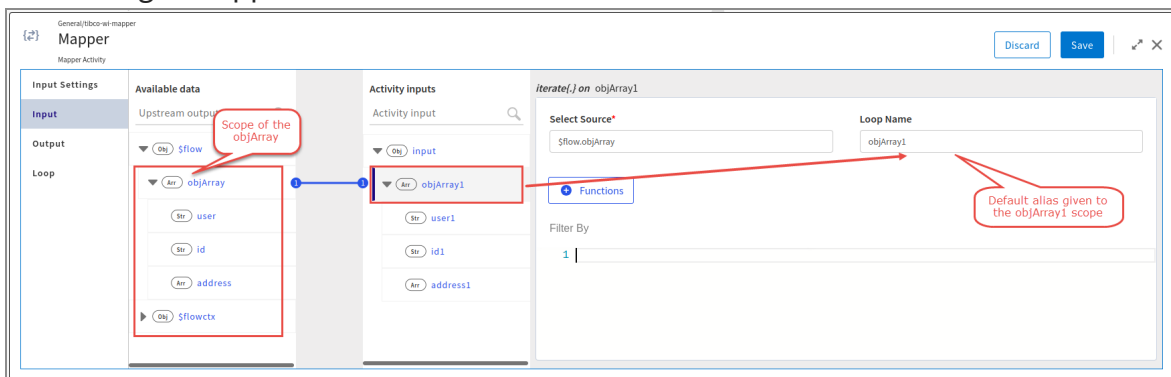
When mapping an array of objects in the input to an identical array of objects (matching property names and data types) in **Available data**, keep the following in mind:

- Map the array at the root level by either dragging or double clicking the **Available Source** array. The **Configure with Source** screen displays the array scope and the scoping variable. You need not map the array object properties individually if you want all properties to be mapped and if the object property names are identical. The properties are automatically mapped.

- If you do not want all the properties within the object to be mapped or if the names of object properties do not match, you must map the object properties individually too after mapping the root. If you do not do the child mapping individually, the mismatched properties in the objects remain unmapped if the properties are not marked as required (marked with a red asterisk). If such a property is marked as required, then you see a warning.
- The size of the input array is determined by the size of the array in **Available data** to which you are mapping.

To map identical arrays of objects:

- Drag the array you want to map from **Available data** (**objArray** in the image below) and drop it on the array in the **Flow outputs** pane (**objArray1** in the image below). The **Configure with Source** screen appears in the text box. If the names of all the child elements match, the child elements get mapped automatically. You need not match each child element individually. In this example, none of the child names match, so you would need to do the individual mapping otherwise none of the elements get mapped.



The "**objArray1**" in the **Loop name** is the scoping variable that constitutes the scope of the current input array. Basically, this means that you can map any element in **objArray1** with an element of the same data type in **flow.objArray** in the **Available data**. So, you are defining the scope of **objArray1** to be all the elements within **objArray**.

Mapping Array Child Elements to Non-Array Elements or to an Element in a Non-Matching Array

There may be situations when you want to map an element within an array of objects to an output element that is not in an array or belongs to a non-matching array in the

Available data pane. In such a situation, you must create an array with a single element. You do this by using the **Configure with JSON** option. When you use this option, it creates an array with an item having a single object element. The single object element treats everything in the **Available data** as the children of the newly created array object element. This allows you to map to any of the **Available data** elements as they are now treated as if they were within an array.

Important: When using the **Configure with Items** option be sure to map the child elements individually. Otherwise, no child elements get mapped. Only elements that you have specifically mapped acquire the mapped values.

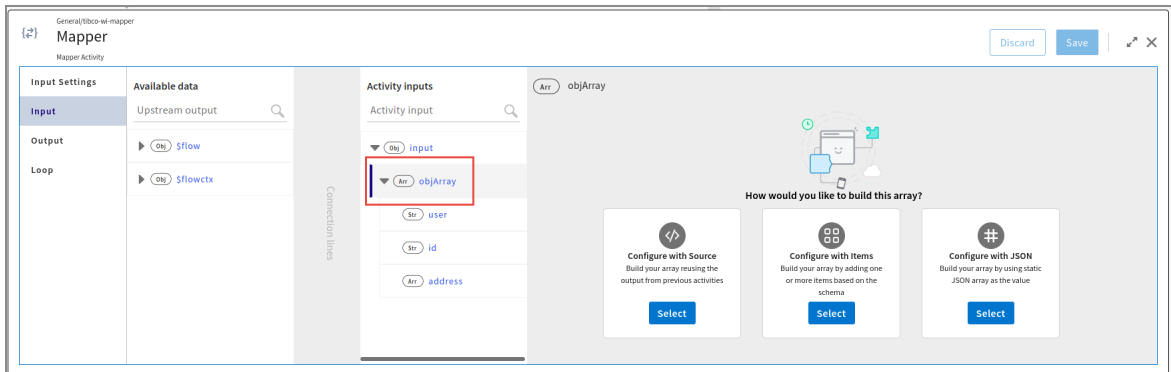
Note: Keep in mind that in this scenario, the resulting length of the array is always one element.

Mapping an array child element to a non-array element is a two-step process:

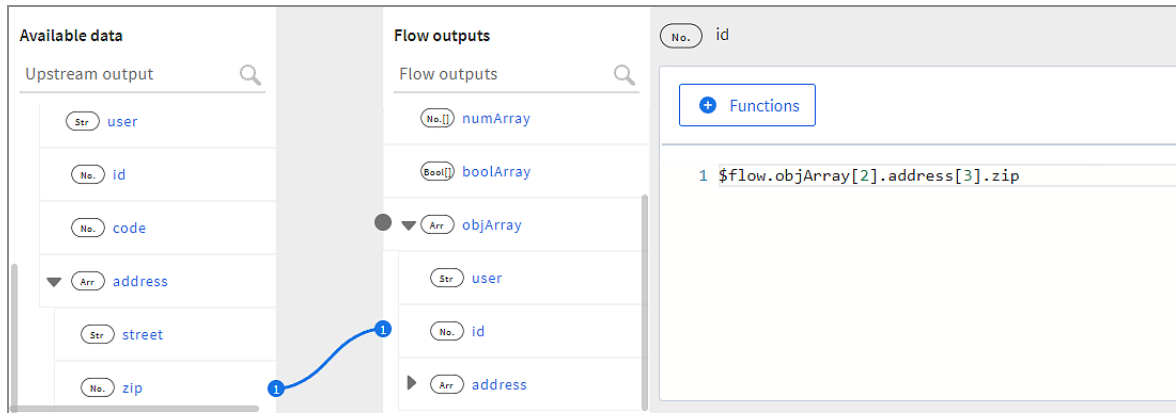
Procedure

1. Click the input array root (**objArray** in the example below) and select the **Configure with Items** option.

This creates an array of objects with a single element in it. The element contains everything under **Available data**, hence allowing you to map to any element in the **Available data** pane. The element you are mapping to can be a non-array element or reside within a nested array.



2. Map each element in the input array individually to any element of the same data type under **Available data**.

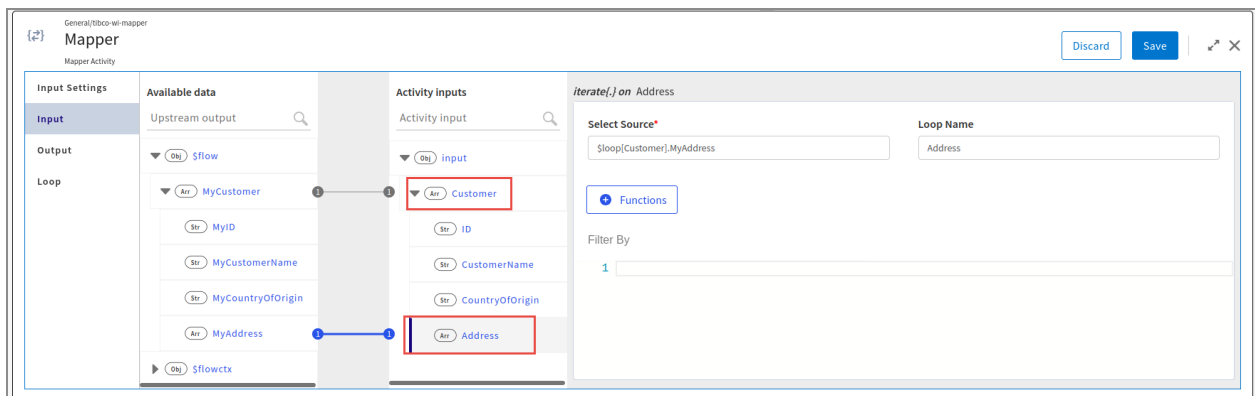


To map an element inside an array, provide the index of the array. To map an element in a nested array, provide the index for both the parent and the nested array as shown.

Mapping Nested Arrays

Before you map a nested array, you must map its parent root. The scoping variable is particularly useful when mapping the child elements in nested arrays.

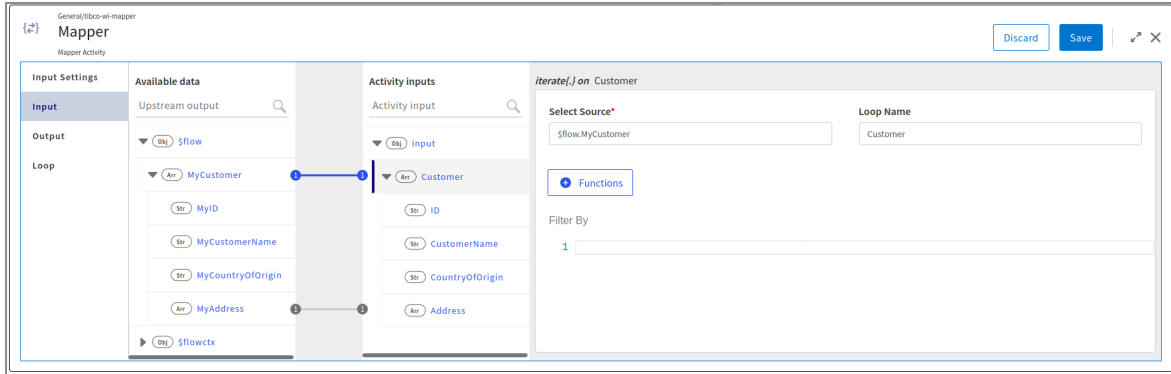
The example below is that of a nested array, where **Address** is a nested array whose parent is **Customer**:



To map **Address**:

Procedure

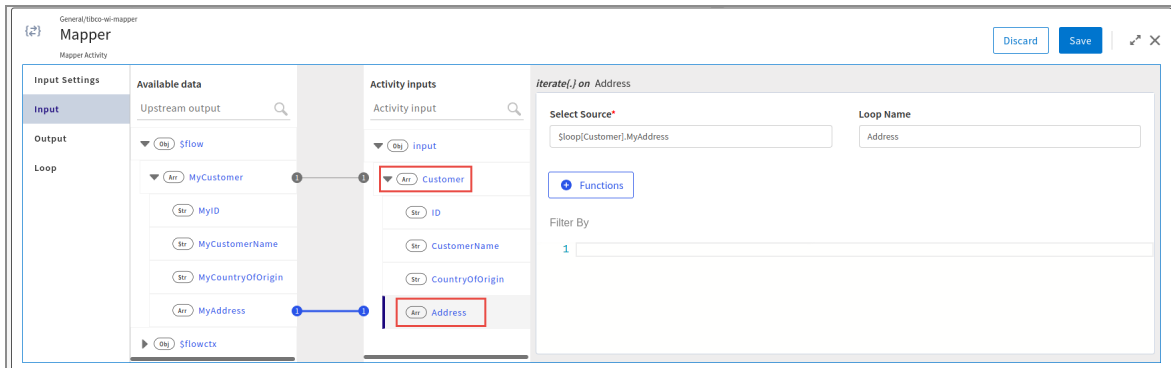
1. Map its parent, **Customer**. When you map **Customer**, you automatically set the scope of **Customer**.



In the image, **Customer** is mapped to **MyCustomer**. In the **Select Source** field, the `$flow.MyCustomer` is the source array (from which **Customer** gets the data) that you are mapping to. This defines the scope (boundary) in the **Available data** within which you can map **Customer**. So, this is the scope of **Customer**.

The **Loop name** field, "Customer", is the scoping variable given to this scope - the loop here refers to the iteration of **Customer**. By default, the scoping variable has the same name as the loop for which the scope is being defined (in this case **Customer**). You can edit the scoping variable to any string that might be more meaningful to you. This is equivalent to saying that mapping of a child element of **Customer** can happen only to children of **MyCustomer** in **Available data**.

2. Map **Address**. Now the scope of **Address** gets defined.



Notice the mapping for **Address**:

- contains the parent scope as well. The parent scope is referred to by its scoping variable, "Customer". Remember that the scope of **Customer** is already set when you mapping **Customer** to **MyCustomer** in the first step, so we can now simply refer to the parent scope by its scoping variable, "Customer".

- `$loop[Customer]` refers to the iteration of the **MyCustomer** array. `$loop` represents the memory address of the **MyCustomer** (the scope for **Customer**) in **Available data**.
- `$loop[Customer].MyAddress1` is the scope of **Address**. This scope is denoted by the scoping variable "Address", which is the second variable in this mapping. Since **Address** is a nested array of **Customer**, when you map to **Address** or its child elements, its mapping includes the scope of **Customer** as well.

Mapping Child Elements within a Nested Array Scope

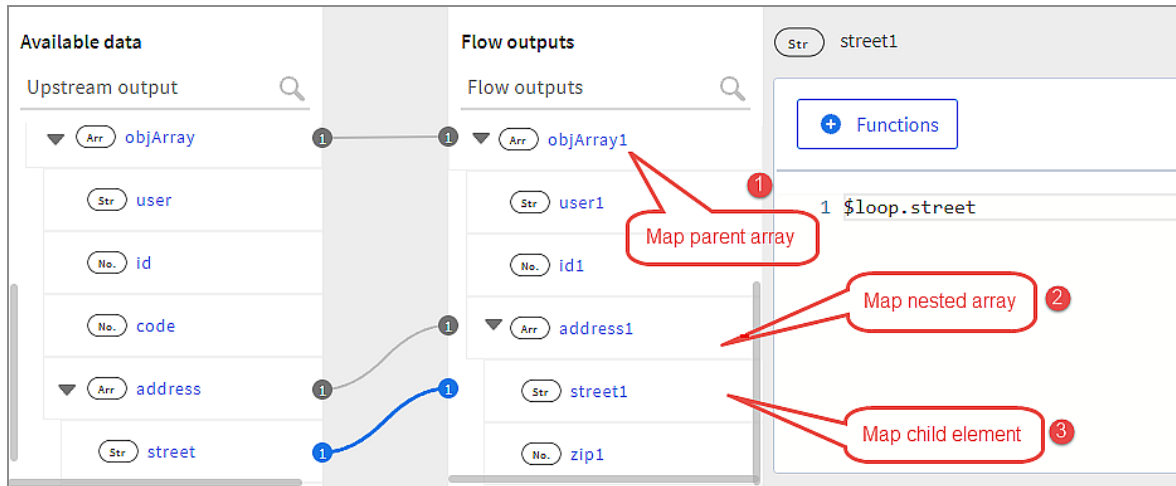
A child element in the input array can be directly mapped to a child element of the same data type within the array scope. As mapping is done within the nested array scope, you need not explicitly state the scoping variable for the nested array scope. The mapping appears as `$loop.<element>`.

To map a nested array child element:

Procedure

1. Map the parent of the nested array.
2. Map the nested array itself.
3. Map the nested array child elements if the names are not identical or if you do not want to map all elements in the nested array.

In the following example, since **street** is within the scope of **address1**, **street1** is directly mapped to **street**. `$loop` implicitly points to **address** which is the scope for **address1** in the input schema.

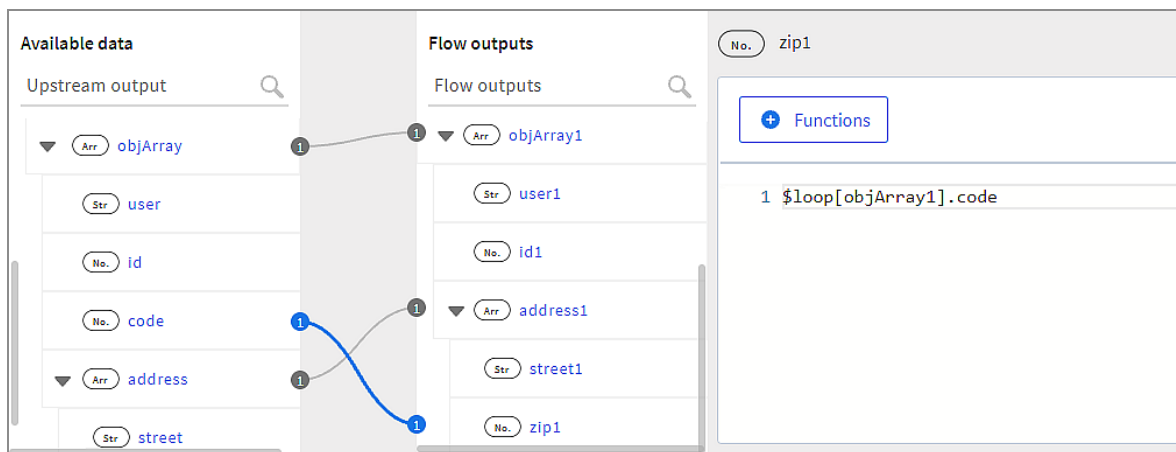


Mapping a Nested Array Child Element Outside the Nested Array Scope

To map a nested array child element outside the nested array scope but within its parent array, you must use the scoping variable of the parent array.

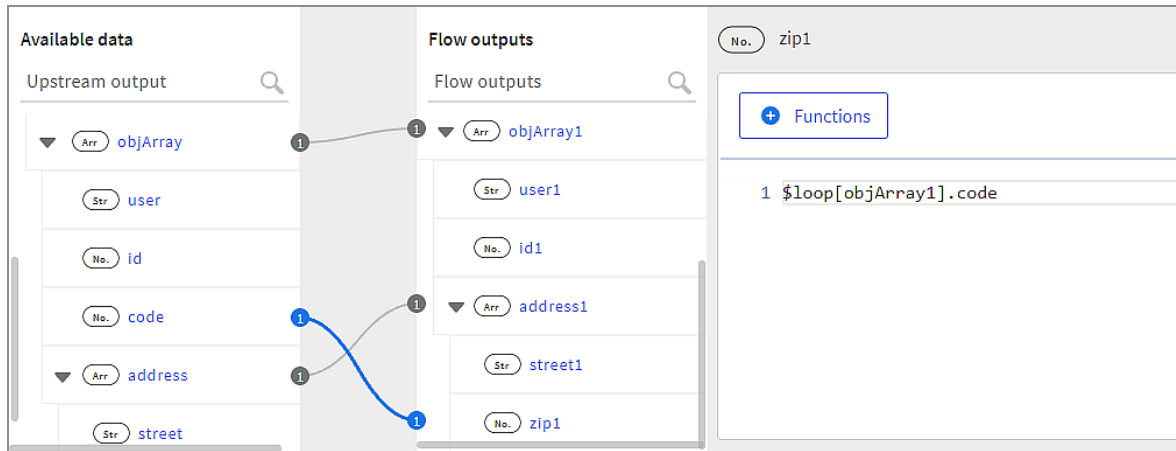
Procedure

1. Map the parent array root.
2. Map the nested array root.
3. Map the nested array child element.



In the image below, `$loop` implicitly points to **address**. In addition, the mapping also explicitly specifies the scope of the parent, "objArray1". This is because **zip1** is

mapped to **code** which is outside the scope of **address1**, but within the scope of its parent array, **objArray1**.

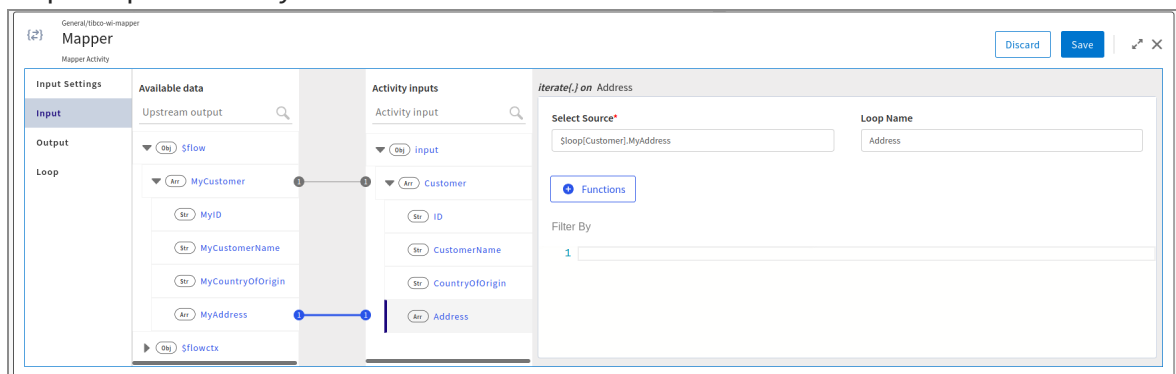


Mapping an Element from a Parent Array to a Child Element in a Nested Array within the Parent

When mapping a primitive data type child element of the parent array to a child element of its nested array, the scope in the mapping is implicitly set to the scope of the parent array. In addition, you must provide the index of the nested array element whose variable you want to map to.

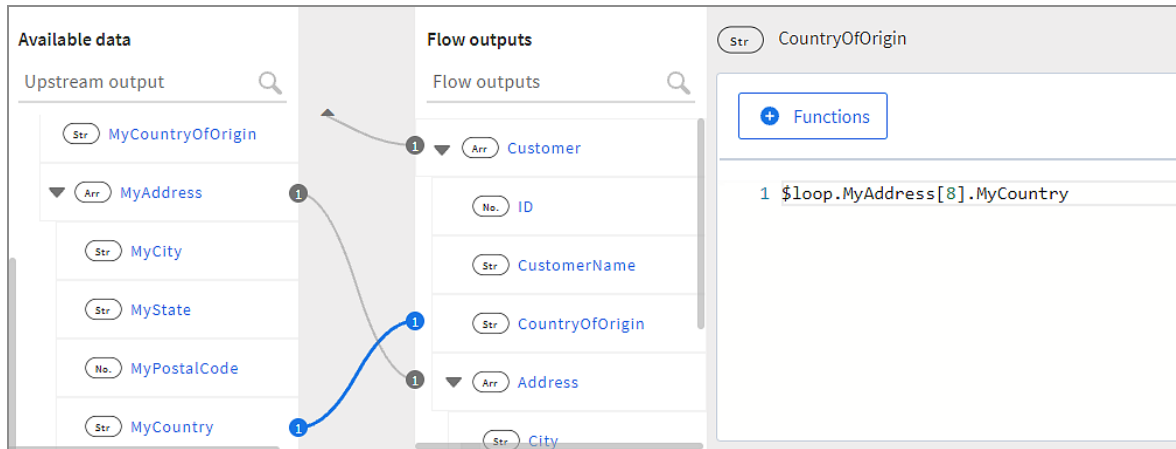
Procedure

1. Map the parent array root.
2. Map the nested array root.
3. Map the parent array element.



In this example, `$loop` is implicitly set to the scope of **Customer** which is

MyCustomer. Notice that you must provide the index of the object in the **MyAddress** array whose **MyCountry** element you want to map to.



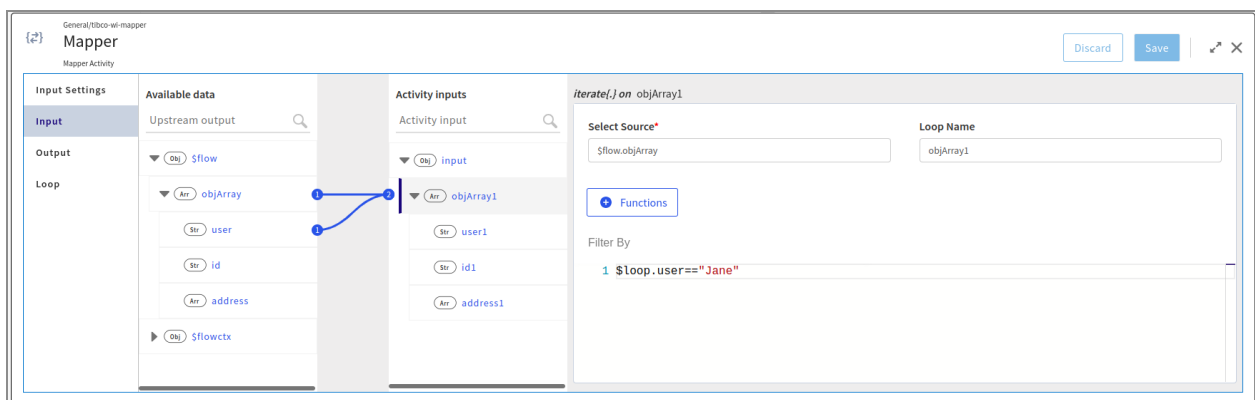
Filtering Array Elements to Map Based on a Condition

When mapping arrays of objects, you can filter the objects that are mapped by specifying a filter in the **Filter by** field when **Configure with Source** option is selected.

Specify the filter in the **Filter by** field. The **Select Source** value is the scope of the element that is mapped and the **Loop name** is the scoping variable.

To add the filter in the **Filter by** field, the **Source name** and the **Loop name** must be specified.

Here's an example that contains a filter in the **Filter by** field:



The above example indicates the following:

- **objArray1** is being mapped to **objArray** in **Available data**

- When iterating through **objArray** in the **Available data**, only the array elements in **objArray** whose child element, **user** is "Jane" get mapped. If **user** is not equal to "Jane" the iteration for that object is skipped and **objArray1** does not acquire that object.
- **\$loop** here specifies the scope of the current loop that is being iterated, in this case **objArray**, whose scope is **objArray1** in **Available data**.

Mapping JSON Data with json.path() Function

Use the `json.path()` function to query an element within JSON data. The JSON data being queried can come from the output of an Activity or trigger. In the mapper, you can use the `json.path()` function by itself when providing value to an input parameter or use it within expressions to refer to data within a JSON structure.

This function takes two arguments:

- the search path to the element within the JSON data
- the JSON object that contains the JSON data you are searching

You can specify a filter to be used by the `json.path()` function to narrow down the results returned by the `json.path()` function.

In order to reach the desired node or a specific field in the node in the JSON data, you must follow a specific notation defined in the JsonPath specification. Refer to [jsonpath](#) for details on the notation to be used and specific examples of using the notation.

Consider the example below which is available for you to experiment with at [jsonpath sample](#).

Examples

The following is an example of how to use the function:

```
json.path("$.store.book[?(@.price > 10)].title", $flow.body)
```

In this example, `$.store.book[?(@.price > 10)].title` is the query path. `[?(@.price > 10)]` is a filter used to narrow down the query results. `$flow.body` is the JSON object against which the query is run (in this case the JSON object comes from the flow input, hence `$flow`). So, this query searches the books array within the `$flow.body` JSON object and returns the title of the books whose price is more than \$10.

Consider the following sample JSON data:

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "store": {
    "book": [
      {
        "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "Availability": [
          {
            "Country": "India",
            "Quantity": 4000,
            "Address": [
              {
                "city": "houston"
              }
            ]
          }
        ]
      },
      {
        "category": "fiction2",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "Availability": [
          {
            "Country": "USA",
            "Quantity": 5000,
            "Address": [
              {
                "city": "sugarland"
              }
            ]
          }
        ]
      },
      {
        "category": "fiction3",
```

```

    "author": "Herman Melville",
    "title": "Moby Dick",
    "isbn": "0-553-21311-3",
    "Availability": [
      {
        "Country": "UK",
        "Quantity": 7000,
        "Address": [
          {
            "city": "stafford"
          }
        ]
      }
    ],
    "price": 8.99
  },
  {
    "category": "fiction4",
    "author": "J. R. R. Tolkien",
    "title": "The Lord of the Rings",
    "isbn": "0-395-19395-8",
    "Availability": [
      {
        "Country": "Australia",
        "Quantity": 2000,
        "Address": [
          {
            "city": "aaaaa"
          }
        ]
      }
    ],
    "price": 22.99
  }
],
"bicycle": {
  "color": "red",
  "price": 19.95
}
},
"expensive": 10
}

```

The following are examples of some JSON query paths that search the JSON data above and return the category of the book. In the examples below, the second input parameter for this function, data is the name of the file that contains the above JSON code.

- `json.path("$.store.book[?(@.Availability[?(@.Quantity >= 6000)])].category", $flow.data)`

In the example above, the query scope is the entire book array. The filter used to query this array is the condition, `[?(@.Availability[?(@.Quantity >= 6000)])]`. Only the `category` values for the book elements that have `Quantity >= 6000` is returned. So, this query returns `fiction3`.

- `json.path("$.store.book[?(@.author == 'Nigel Rees')].category", $flow.data)`

returns reference since it uses the filter `[?(@.author == 'Nigel Rees')]` and the only book authored by Nigel Rees in this array of books has its `category` as reference.

- `json.path("$.store.book[?(@.Availability[?(@.Address[?(@.city == 'sugarland')])])].category", $flow.data)`

This query is an example of a nested filter where `[?(@.Availability[?(@.Address[?(@.city == 'sugarland')])])]` is the outer filter and the nested filter within it is `[?(@.city == 'sugarland')]`. It returns reference.

- `json.path("$.store.book[0].category", $flow.data)`

This query does not use a filter. It returns reference, since your query scope is limited to the `book[0]` element only within the `store` object and your request is to return the value of `category`.

Constructing the any, param, or object Data Type in Mapper

When mapping values for data type `any` or `object`, you must manually enter the values in the mapper text box. Below are some examples of how to construct the data type `any`:

Assigning a literal value to data type `any`

To assign literal values to the `any` data type, you click on the element of type `any`, then simply enter the values you want to assign to it in the mapper text box. For example, to assign the string `Hello!` enter:

```
"Hello!"
```

Assigning an object value to an object or element of data type any

Here is an example of how to assign literal values to an object:

```
{
  "Author": "Martin Fowler",
  "ISBN": "0-321-12742-0",
  "Price": "$45"
}
```

Here, "Author", "ISBN", and "Price" are the object properties. You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

Assigning an array value to an object or data type any

Here is an example of how to assign an array value to an array of objects or to an element of data type any:

```
[
  {
    "Author": "Martin Fowler",
    "ISBN": "0-321-12742-0",
    "Price": "$45"
  }
]
```

You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

Assigning a value from the upstream output

When mapping to an element from the upstream output, the data type of the source element whose value you are assigning determines the data type of the destination element. For example, if you assign the value of an array, then the target element (the element of data type any) is treated as an array, likewise for a string, number, boolean, or object. For example, if you are mapping `$flow.Author` which is an array, then the Author object in the input (destination object) would also be an array. That is, there is a direct assignment from the source to the destination.

- **Single Element of Primitive Data Type:** To assign the value of a single element of a

primitive data type that belongs to the output of the trigger, a preceding Activity, or the flow input, you must enter the expression for it. For example to assign the value of `isbn` which comes from the flow input, enter the expression:

```
"=$flow.isbn"
```

Here, `$flow` is the scope within which `isbn` falls.

- **An object:** When assigning an object, you must create a mapping node within the object. The mapping node is used to define how the object should be constructed and the various fields within the object mapped. For example, to assign the `bookDetails` object, enter:

```
{
  "mapping": {
    "Author": "=$flow.author",
    "ISBN": "=$flow.name",
    "Price": 20,
    "BestSeller": true
  }
}
```

You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

- **An array of objects:** The following two examples show you how to assign values to arrays:
 - **Building a new array**

To provide values for an array that has a fixed size (where the number of elements is declared), you must provide the values for each array element. For example, if the array has two elements, you must provide the values for each property of the object for both objects. Here is an example of how to do that:

```
{
  "mapping": {
    "books": [
      {
        "author": "=$loop.author",
        "title": "=$loop.title",

```



```

        "price": "=$loop.price"
    },
    {
        "author": "Author2",
        "title": "BookTitle",
        "price": 19.8
    }
]
}
}

```

In the example above `books` is an array of two elements. The values for each property for both elements are provided.

You can use a function instead of a literal value when assigning values for each element. For details, see [Using Functions](#).

- **Building an Array from an upstream output array**

In the following example, `books` is an array of books coming from the upstream output. To iterate over the array, `$flow.store.books` in upstream output, and assign its values to the input array, you would enter the following in the mapper text box:

```

{
  "mapping": {
    "@foreach($flow.store.books)": {
      "author": "=$loop.author",
      "title": "=$loop.title",
      "price": "=$loop.price"
    }
  }
}

```

The `"@foreach($flow.store.books)"` indicates that you are iterating an array of objects where the `$flow.store.books` is the array. `$flow` is the scope within which `store.books` falls and `$loop` represents the scope for each property within the object. Refer to the [following section](#) for details on the `forEach()` function.

- **Using a function:** The following example leverages the output of a REST Invoke Activity to get a pet from the public petstore service. The mapper uses the `string.concat()` function and assigns the function return value to the description

field in the data structure:

```
{
  "mapping": {
    "data.description": "=string.concat(\"The pet category name is:
\",$Activity[rest_3].result.category.name)"
  }
}
```

Assigning Values to the param Data Type

When you import an app that was originally created in Project Flogo™, the app could contain elements that are of data type param. The param data type is similar to the object data type in that it consists of key-value pairs. The difference between an object and a param is that the object can contain values of any data type whereas the values for elements in the param data type *must* be of data type string only.

Here's an example of assigning values to a param data type element:

```
{
  "mapping": {
    "Author": "=$flow.author",
    "ISBN": "=$flow.name",
    "Price": "$20"
  }
}
```

Coercing of Activity Input, Output, and Trigger Reply Fields

In the OSS marked Activity input, output, or trigger reply configuration, if you have defined a parameter, but have not defined or cannot define a schema for the parameter, you can coerce the parameter to take the value from a schema that you dynamically define during design time. This feature is particularly useful for apps that were created in Project Flogo and imported to Flogo Enterprise. Such apps most likely have activities for which input parameters or output are not defined with a schema.

Currently, coercion of parameters is supported only for the following data types:

- array

- object
- param
- any

After you enter the schema, it is displayed in a tree format under **Activity inputs, Output** tab, or **Trigger reply** in the mapper. All subsequent activities also display the elements of the schema under the Activity in the Upstream Output. The schema elements are now available for you to map.

Important Considerations


- Coercion is supported only in the **Default** category activities which are the activities marked as OSS, except for the **Return** and **Start a SubFlow** activities. These two activities display flow-level data. The flow-level inputs and outputs can be entered or modified only on the **Flow Inputs & Outputs** accordion tab, hence they cannot be coerced from the **Input** tab of the Activity itself.
- Currently, coercion is supported only for top-level parameters. Nested coercion (for example, an object within an object) is not supported.
- Currently, coercing a schema for trigger input is not supported. The coercing option is not available on the **Map to Flow Inputs** tab in the trigger configuration. This is because the parameters you see on this tab are flow input parameters and are not related to the trigger. You have the option to coerce these parameters on the **Input** tab of the **Flow Inputs & Outputs** accordion tab.
- After you have mapped a child element within a parameter, if you change the name of the parent or the child, your mapping is lost. However, if you change the data type of the element, the mapping is preserved, but you see an error related to the mismatch in data type.
- The schema you enter is preserved when you export and import the app.
- If you edit the schema at a later time, as long as you click **Apply** after editing, your edits are displayed in the mapper. You must then click **Save** in the mapper to persist your schema changes.
- You cannot coerce a parameter or edit its schema in any activity displayed in a subflow. For example, if the **OracleDatabaseQuery** activity is displayed in both the main flow and the subflow, you cannot edit the schema of any of its parameters in the subflow. But you can edit the schema of the **OracleDatabaseQuery** activity in the

main flow. This is because the subflow activity input and output schemas are inherited from the main flow. There is a possibility that the same subflow could be used in multiple main flows, so if you edit an activity in the subflow it could break another main flow that uses the subflow.

To provide the schema for coercion:

Procedure

1. On the flow details page, click the activity or trigger to open its configuration.
2. Click any of the following tabs that you want to configure:
 - **Input:** To configure a parameter in the activity input
 - **Output:** To configure the schema for the activity output
 - **Map from Flow Outputs:** To configure the trigger reply
3. To configure a schema:
 - For a parameter in activity input, hover your mouse cursor over the parameter name for which you want to configure the schema under **Activity inputs**.
 - For the Activity output, hover your mouse cursor over the parameter name for which you want to configure the schema.
 - For a parameter in the trigger reply, hover your mouse cursor over the parameter name.

Click the ellipsis icon () that is displayed next to it. **Clear mappings** and **Coerce with schema** options are displayed.

4. Click the **Coerce with schema** option.

i **Note:** The **Coerce with schema** icon is displayed against the parameter name for only those parameters that do not have a schema defined on the **Input Settings** tab (or a schema cannot be defined because the Activity does not have an **Input Settings** tab, for example, the OSS-marked activities) *and* whose data type is one of the following: array, param, object, or any.

5. Enter the schema for the parameter or activity output and click **Apply**. The mapper validates that the data type of the schema you entered matches the data type of the parameter being coerced. If the data types do not match, **Apply** remains disabled

and you see an error. For activity input and trigger reply, the schema you enter displays in a tree format under the parameter name in the mapper.

- For the activity output, the schema is displayed in a tree format on the **Output** tab of the activity. **Available data** displays the output of the preceding activities.
6. Click **Save** to persist the schema into the database or **Discard** to discard the schema. Now you can map the child elements within the parameter. In the case of the activity **Output** tab, the output tree does not display in the current activity but is displayed in the mapper for subsequent activities only. Once persisted in the database, these schema trees get displayed in the **Available data** area of the mapper for subsequent activities. This allows you to map to them in subsequent activities.

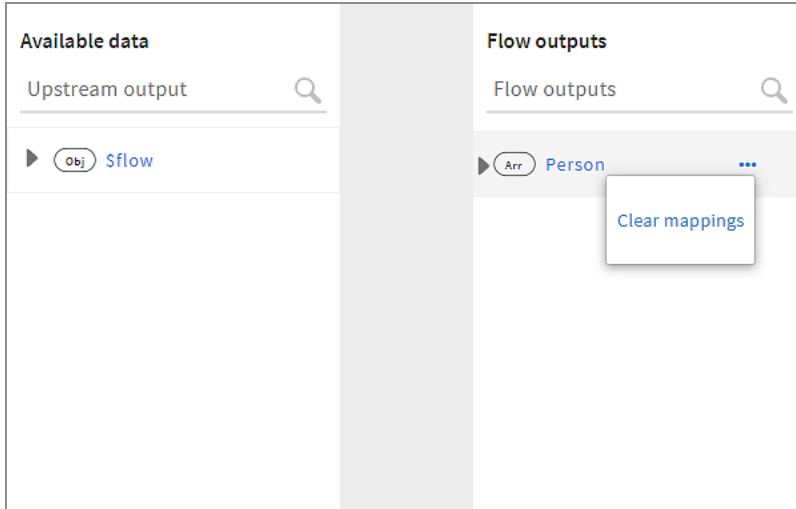
Clear Mapping of Child Elements in Objects and Arrays

After mapping an array or an object, you can clear the mapping of all the child elements within that array or object with one click. The mapping is cleared at the root level and mapping for everything under that root gets cleared, even the nested arrays and objects, should there be any. To clear mapping for individual elements in an array or object selectively, click on that element and delete the mapping for it.

To clear the mappings for all child elements of an array or object:

Procedure

1. In the mapper, hover your mouse cursor to the right end of the root name until the ellipsis icon (⋮) is displayed, then click it.
2. Click **Clear mappings**



Ignoring Missing Object Properties when Mapping Objects

There may be instances when you map objects where one or more object properties might be missing in the source or target object. The mapper can be set to ignore such cases.

If you want the mapper to ignore such cases, you must set the `FLOGO_MAPPING_SKIP_MISSING` engine variable to `true`. The mapper ignores the missing mapping as long as the element is optional (not marked as mandatory with a red asterisk against it). Elements marked as mandatory must be mapped. For more details, see the section on [Environment Variables](#).

Mapping Data by Using `if/else` Conditions

The `if/else` statements are used to execute blocks of code based on the specified conditions.

```
if (condition1)
{
  // execute this block of code
}
else if (condition2)
{
  // execute this block of code if the previous condition fails
}
```

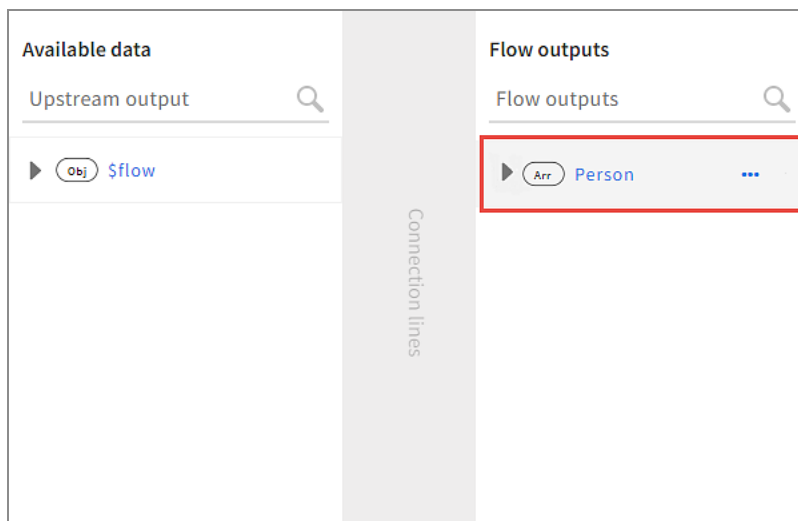
```
else
{
  // execute this block of code if all conditions fail
}
```

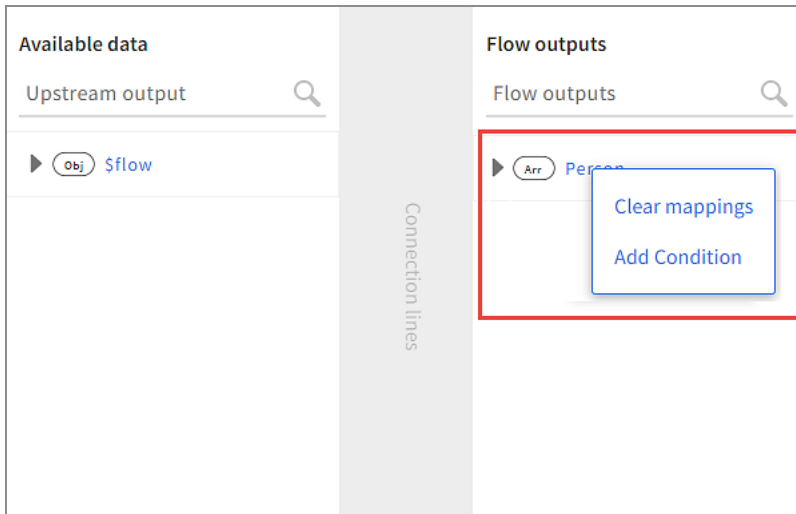
You can add conditions in your data mappings to get outputs based on those conditions. You can add conditions to primitive objects, nested arrays, nested objects, and any other type of input. `if/else` conditions are available in activities and triggers in the main flow and error handler.

To Map Data Using Conditions

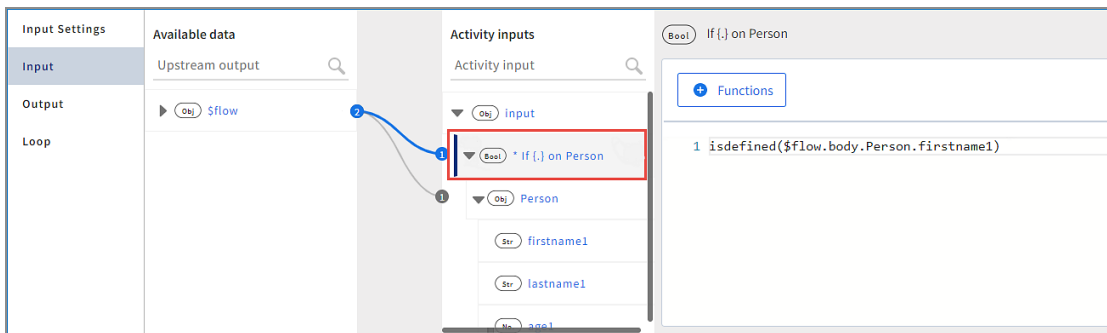
Procedure

1. Click the ellipsis icon **...** to open the menu of the element to which you want to add the conditions. Select **Add Condition**. An `If` condition is added to the element.

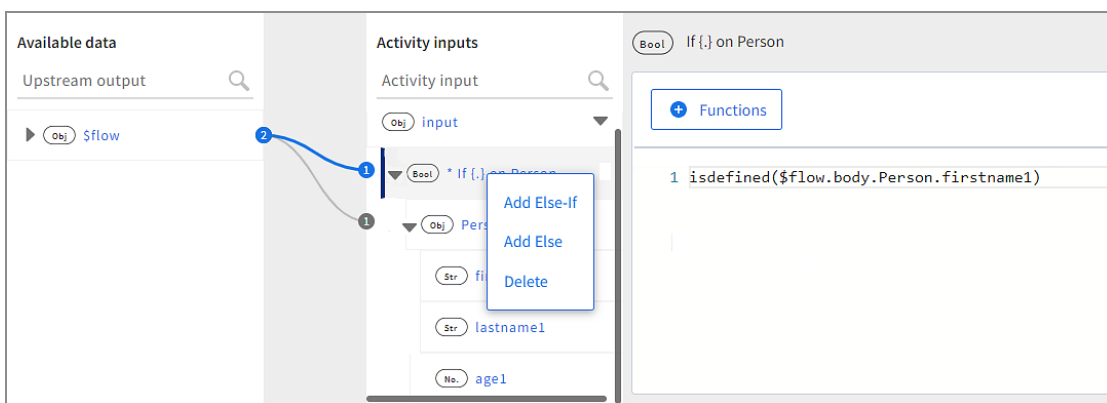




2. In the text editor of the If condition, enter an expression whose result evaluates to a Boolean value. You can enter the expression manually or map data from the **Available data** pane. If children elements exist, you can enter values for them.



3. To add an Else-if or an Else condition, click the ellipsis icon **⋮** to open the menu of the element with the If condition. Click **Add Else-If** or **Add Else**.



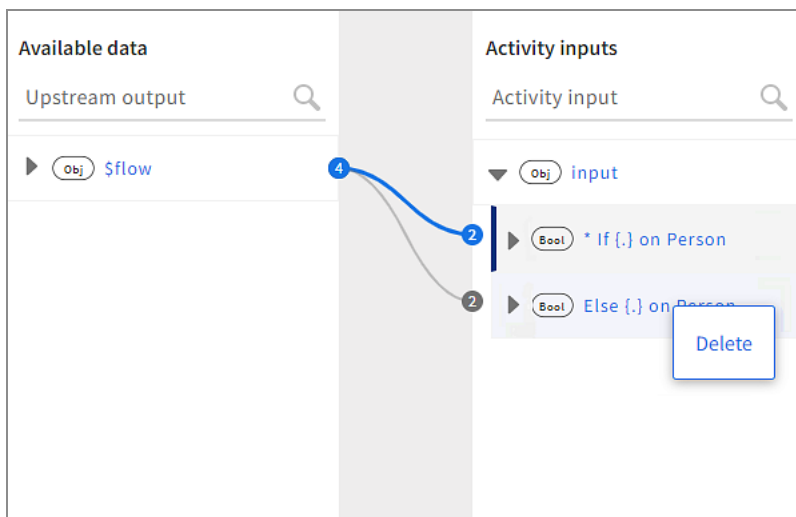
Considerations when using conditions:

- For one If condition, you can add multiple Else-if conditions and one Else condition.
- You can add an Else condition only from an element with an If condition.
- You can add an Else-if condition from an If condition and from an Else-if condition.

i Note: In case the option to add conditions is not visible for the last element in the **Activity inputs** pane, scroll further down to view the options.

Deleting a Condition

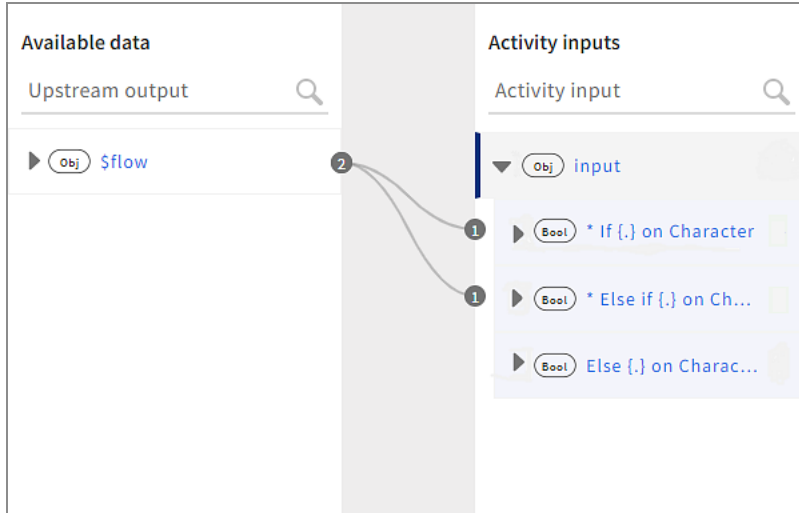
Click the ellipsis icon **⋮** to open the menu of the condition that you want to delete. Click **Delete**.



To delete an If condition that has Else-if and Else conditions:

You cannot directly delete an If condition that has Else-if and Else conditions. You must first delete the Else-if and Else conditions to delete the parent If condition.

In the following example, to delete the If condition on **Character**, you must delete the Else-if and Else conditions.



i Note: For OSS activities having the **Coerce with schema** option, you can maintain only one schema for the input that you coerce. If you add conditions to the coerced inputs, you cannot change the schema specific to a condition. When you update the schema, it is updated for all the blocks.

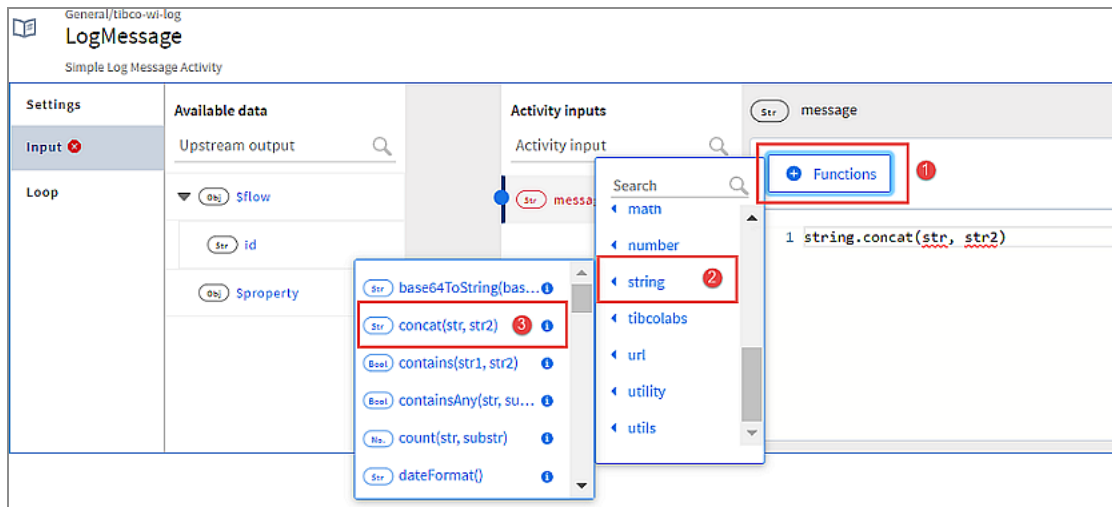
Using Functions

You can use a function from the list of functions available under **Functions** in the mapper. Input parameters to the function can either be mapped from an element under **Available data**, a literal value, or an expression that evaluates to the appropriate data type or any combination of them.

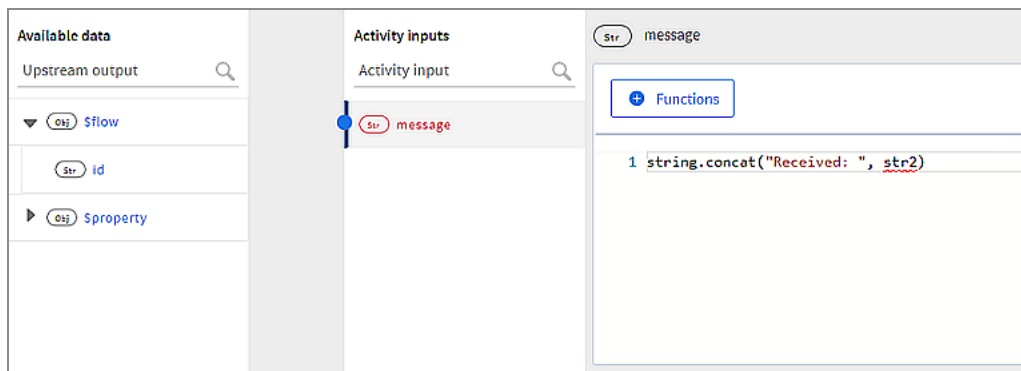
The following procedure illustrates an example that concatenates two strings and assigns the concatenated value to the **message**. We manually enter a value for the first string (**str1**) and map the second string to **id** under **\$flow**. The value for **id** comes from the flow input.

Procedure

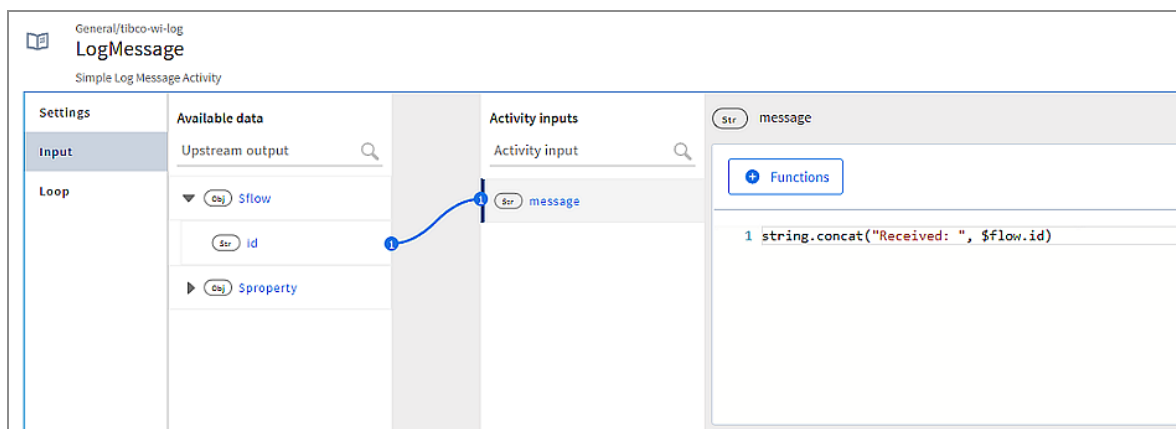
1. Click the **message** to open the text editor to the right.
2. Click **Functions**. Expand the **string** function group and click **concat(str1, str2)**.



3. Select **str1** in the function and type "Received: " (be sure to include the double quotes as shown) to replace **str1** with it.



4. Drag **id** from **\$flow** and drop it in place of **str2**.



At run time, the output from the concat function is mapped to the **message**.

Using Expressions

You can use two categories of data mapping expressions.

Basic Expression

Basic expressions can be written using any combination of the following by using operators:

- literal values
- functions
- previous Activity or trigger output

Refer to [Supported Operators](#) for details on the operators that can be used within a basic expression.

Here are some examples of basic expressions:

```
string.concat("Rest Invoke response status code:",$activity  
[InvokeRESTService].statusCode)
```

The above example combines the string and the statusCode from the InvokeRestService activity.

```
string.length($activity[InvokeRESTService].responseBody.data) >=7
```

The above example checks whether the length of data of the responseBody is greater than or equal to 7.

```
$activity[InvokeRESTService].statusCode == 200 && $activity  
[InvokeRESTService].responseBody.data == "Success"
```

The above example checks whether the statusCode is 200 and the data of responseBody has the value as "Success".

Ternary Expression

Ternary expressions are assembled as follows:

```
condition ? statement1 : statement2
```

The condition is to be evaluated first. If it evaluates to true, then `statement1` is executed. If the condition evaluates to false, then `statement2` is executed.

Here is an example of basic ternary expression:

```
$Activity[InvokeRESTService].statusCode == 200 ? "Response successfully":"Response failed, status code not 200"
```

In the above example `$Activity[InvokeRESTService].statusCode == 200` is the condition to be evaluated.

- If the condition evaluates to true (meaning `statusCode` equals 200), it returns `Response successfully`.
- If the condition evaluates to false (meaning `statusCode` does not equal 200), it returns `Response failed, status code not 200`.

Here is an example of a nested ternary expression:

```
$Activity[InvokeRESTService].statusCode == 200 ? $Activity [InvokeRESTService].responseBody.data == "Success" ? "Response with correct data" : "Status ok but data unexpected" : "Response failed, status code not 200"
```

The example above checks first to see if `statusCode` equals 200.

- If the `statusCode` does not equal 200, it returns `Response failed, status code not 200`.
- If the `statusCode` equals 200, only then it checks to see if the `responseBody.data` is equal to "Success".
 - If the `responseBody.data` is equal to "Success", it returns `Response with correct data`.
 - If the `responseBody.data` is not equal to "Success", it returns `Status ok but data unexpected`.

Supported Operators

Flogo supports the operators that are listed below.

- ==
- ||
- &&
- !=
- >
- <
- >=
- <=
- +
- -
- /
- %
- Ternary operators - nested ternary operators are supported.

For example, `$activity[InvokeRESTService].statusCode==200?($activity[InvokeRESTService].statusCode==200?true:false):false`

Combining Schemas Using Keywords

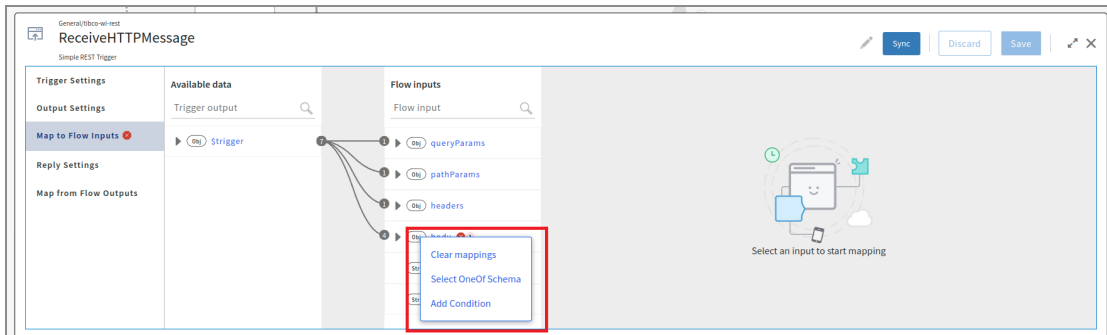
You can use the `oneOf`, `anyOf`, and `allOf` keywords to combine schemas.

- `oneOf` keyword: ensures that the given data is valid against exactly one of the selected subschemas.
- `anyOf` keyword: ensures that the given data is valid against one or more of the selected subschemas.
- `allOf` keyword: ensures that the given data is valid against all the subschemas.

Using the `oneOf` Keyword

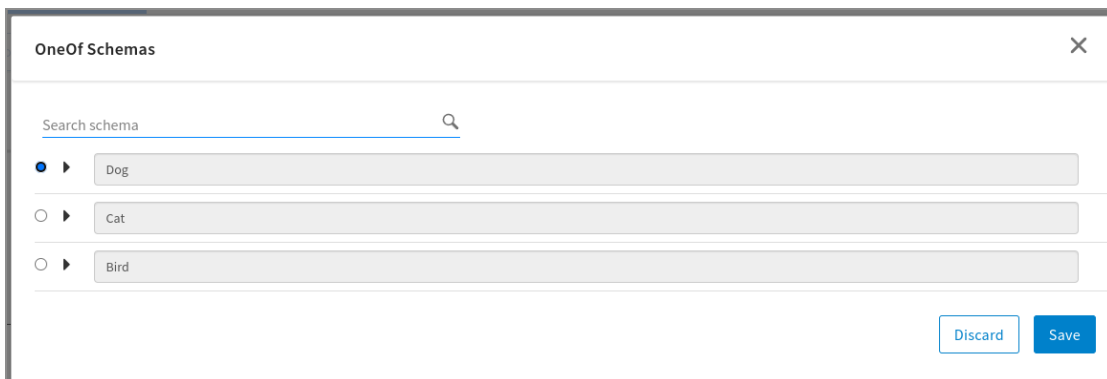
Procedure

1. On the schema object in the activity/trigger input, click the ellipsis icon **⋮**.
For an object with a **oneOf** keyword, the **Select OneOf Schema** option is displayed.



i Note: Select the **Select OneOf Schema** option before you add the if/else conditions using the **Add Condition** option. If you add the if/else conditions first, then the **Select OneOf Schema** option is not displayed in the menu.

2. Click **Select OneOf Schema**.
The schema selector dialog displays all available schemas with a **oneOf** array.

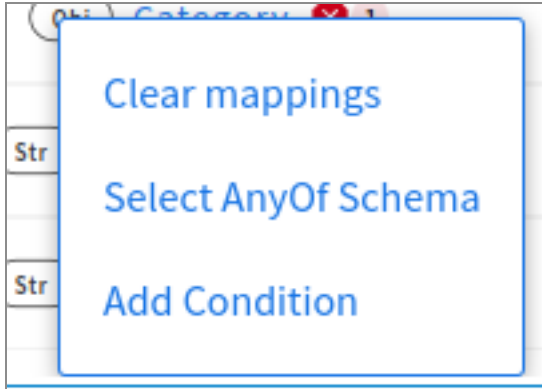


3. Select one schema from the schema selector dialog.

Using the anyOf Keyword

Procedure

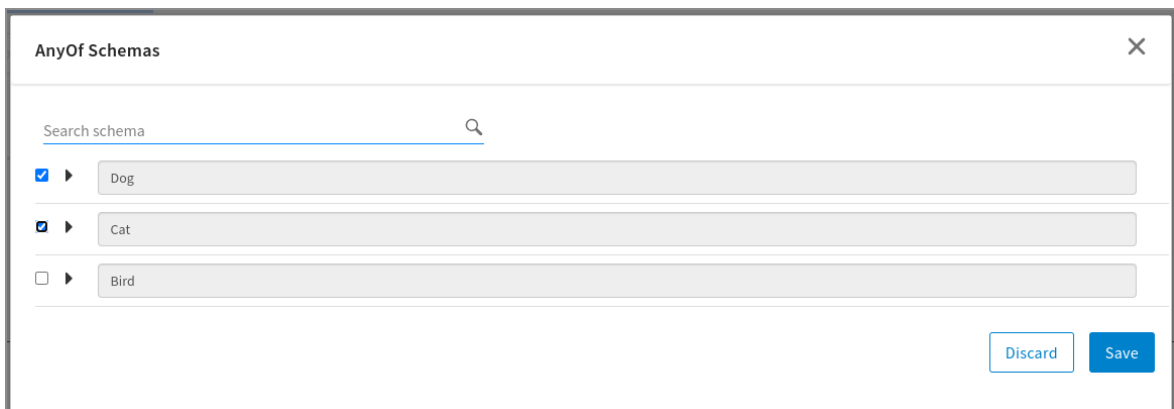
1. On the schema object in the activity/trigger input, click the ellipsis icon **⋮**.
For an object with the **anyOf** keyword, the **Select AnyOf Schema** option is displayed.



Note: Select the **Select AnyOf Schema** option before you add the if/else conditions using the **Add Condition** option. If you add the if/else conditions first, then the **Select AnyOf Schema** option is not displayed in the menu.

2. Click **Select AnyOf Schema**.

The schema selector dialog displays all available schemas with the anyOf array.



3. Select one or more schemas from the schema selector dialog.

Developing APIs

Flogo Enterprise lets you take an API-first development approach to implement APIs from a Swagger Specification 2.0, OpenAPI Specification 3.0, or GraphQL schema. After you upload an API specification file or a GraphQL schema, Flogo Enterprise validates the file and if the validation passes, it automatically creates the flows and triggers for you.

Using an OpenAPI Specification

You can create the Flogo app logic (flows) by importing an API specification file. You can simply drag a specification file to the UI or navigate to it. If you have an existing specification file stored in the TIBCO Cloud™ Integration - API Modeler or TIBCO Cloud™ API Modeler, select it when creating the flow. The flows for your app are automatically created based on the definitions in the specification file that you uploaded.



Tip: For more information on the TIBCO Cloud™ API Modeler, see [TIBCO Cloud™ API Modeler](#).

When you create an app from a specification, the **ConfigureHTTPResponse** and **Return** activities are automatically added to the flow. The mappings from trigger output to flow inputs are configured for you based on the definitions in the specification. The output of the **ConfigureHTTPResponse** Activity is automatically mapped to the **Return** Activity input. However, you must configure the input to the **ConfigureHTTPResponse** Activity manually. If you have multiple response codes configured in the REST trigger, the first response code is configured in the **ConfigureHTTPResponse** Activity by default. The only exception to this is if you have a response code of 200 configured. In that case, the 200 response code is configured in the **ConfigureHTTPResponse** Activity by default.

Before the Flogo app is created, a validation process ensures that the features defined in the specification are supported in Flogo Enterprise.

Considerations when using an API specification file to create a Flogo app:

- Flogo Enterprise supports Swagger Specification 2.0 and OpenAPI Specification 3.0.
- Currently, Flogo Enterprise supports only the JSON format.
- Cyclic dependency is not supported when creating flows from specifications. For example, if you have a type *Book* that contains an object element of the type, *Author*. The type *Author* in turn contains an element of the type *Book* that represents the books written by the author. To retrieve the *Author*, it creates a cyclic dependency where the *Author* object contains the *Book* object and the *Book* type, in turn, contains the *Author* object.
- String, integer, and boolean are the data types supported by Flogo Enterprise. A data type that appears in your specification but is not supported by Flogo Enterprise results in an error being displayed.
- Schema references within schemas are not supported.

- If the specification has a response code other than 200 (OK) or 500 (Error), the method that contains the unsupported response code is not created.
- You can enter a schema for the response code 200, but the 500 response code must be a string.
- The basepath element in the schema is not supported.

If you get a validation error, you can either cancel the process of generating the app or click **Continue**. If you opt to continue, the process of app creation continues and the parts of the specification that did not pass the validation are ignored.

i Note: The REST reply data type is by default set to any data type. To configure the reply to an explicit data type, see [Configuring the REST Reply](#) section.

To create an app using an API specification and upload the specification file:

Procedure

1. Log in to TIBCO Cloud™ Integration.
2. On the **Apps** page, select **Create/Import**.

The **What do you want to build?** dialog is displayed.


API Name	Version	Group	Last Modified
TIBCO Cloud Integration PetStore Sample	1.1	Default	09 March 2022 12:26 PM
Swagger Petstore	1.0.6	Default	16 March 2022 10:45 AM
Swagger Petstore - OpenAPI 3.0	1.0.5	Default	19 March 2022 3:46 AM
Swagger Petstore	1.0.0	Default	19 March 2022 3:46 AM
Dummy-apispec	1.0.0	Default	19 March 2022 3:51 AM

3. To create a Flogo app using an OpenAPI specification:

- Under **Quickstart > All app types > APIs**, click **Create an app from OpenAPI**.
 - On the left, select a category that identifies the type of integration you need. On the right, click **Create an app from OpenAPI**.
4. In the block that displays below your selection, select one of the following options:
 - Create a flow using an API specification that exists in the TIBCO Cloud™ Integration-API Modeler. To do this, on the **API Specs** tab, select the specification that you want to use.
 - Use an API specification saved locally on your computer by uploading it to Flogo Enterprise. To do this, click the **Upload file** tab. Browse to the saved API specification on your local machine or drag your saved API specification into the dialog.
 5. Click **Import OpenAPI spec**.
 6. In each flow:
 - a. Open the flow by clicking its name.
 - b. Click the trigger to open its configuration dialog.
 - c. Map the following:
 - On the **Map to Flow Inputs** tab, map the **Available data** to **Flow inputs**.
 - On the **Map from Flow Outputs** tab, map the **Available data** to **Trigger reply**.

To test the deployed app, follow the procedure in the [Testing the Deployed App](#) section.

You can also download the specification used to create the app by following the procedure in [Downloading the API Specification Used](#) section.

You also have the option to copy the endpoint URL from the **Endpoints** tab by clicking the **Copy spec URL**. Or you can click the () icon next to the endpoint URL itself.

The following is a list of Swagger 2.0 and OpenAPI Specification 3.0 features supported in Flogo:

- Path Templating
- Media Type

- Request Types: application/json, multipart/formdata, x-www-form-urlencoded
- Response Types: text/plain, application/json
- Multiple Status Codes
- Path Item Object
- Parameter Object
- Request Body Object
- Reference Object
- Header Object
- Security Scheme Object
- allOf, oneOf, and anyOf keywords

For more information, refer to [OpenAPI Specification](#).

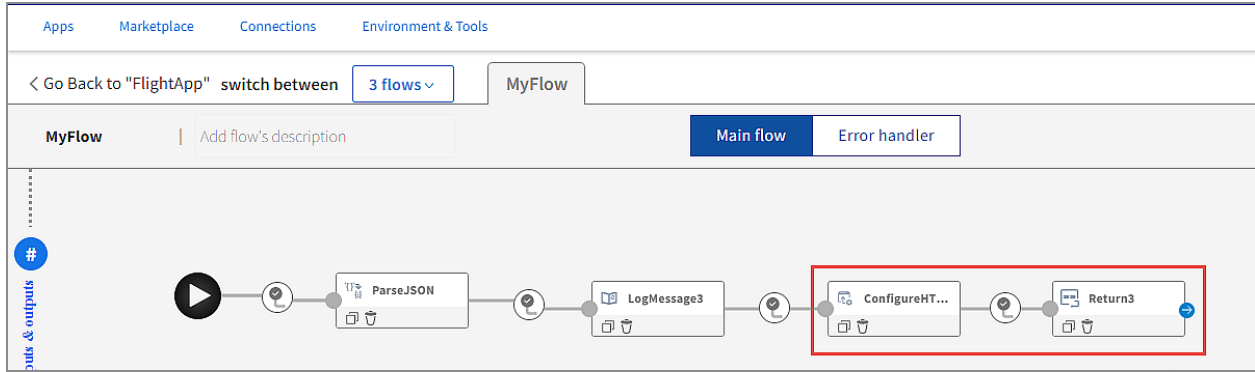
Result

The app is created and the **App Details Page** is displayed for the new app. Your app does not run and has zero instances. To start and scale your app, see [Starting, Stopping, and Scaling Apps](#).

Configuring the REST Reply

When creating a REST app from a Swagger 2.0 or OpenAPI 3.0 API specification, the **ReceiveHTTPMessage** reply data type is set to any by default. You must explicitly configure the reply type.

To explicitly configure the reply type, add a **ConfigureHTTPResponse** Activity in the flow. This Activity must immediately precede the **Return** Activity in the flow.

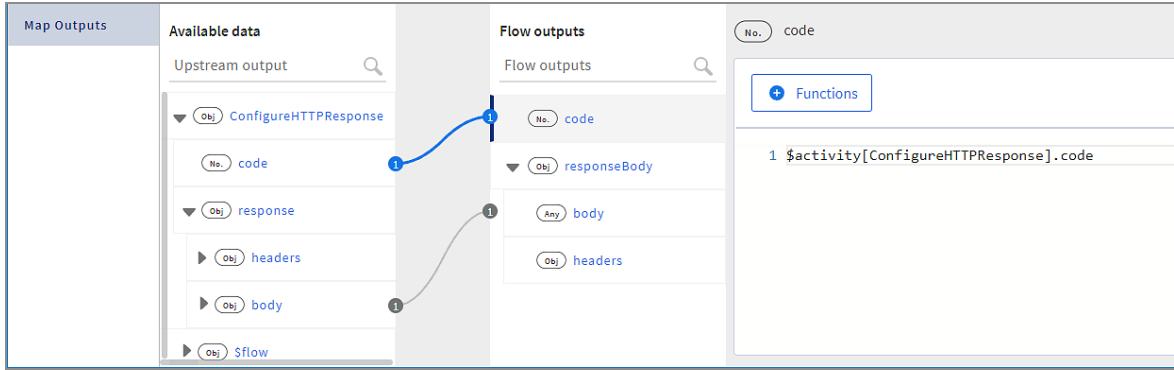


You can configure custom codes that you want to use in the HTTP reply on the **Reply Settings** tab of the **ReceiveHTTPMessage** trigger.

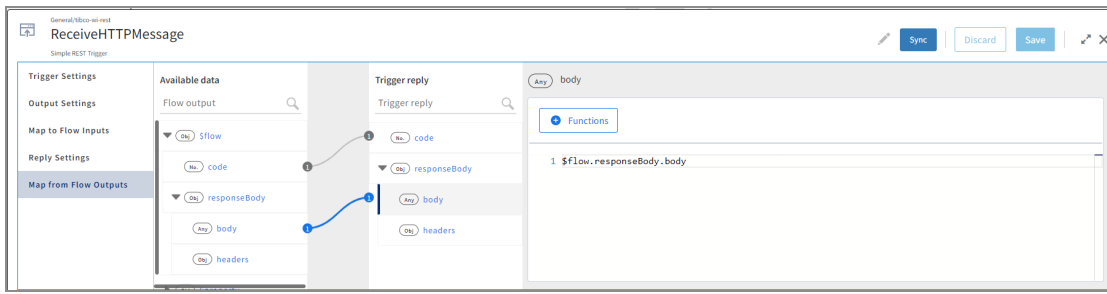
Follow these steps to configure your HTTP reply:

Procedure

1. Open the REST trigger configuration pane by clicking it.
2. On the **Reply Settings** tab of the **ReceiveHTTPMessage** REST trigger, configure the custom codes that you want to use. Refer to the section, "REST Trigger" in the *Activities, Triggers, and Connections Guide*.
3. Add a **ConfigureHTTPResponse** Activity immediately preceding the **Return** Activity in the flow.
4. Open the **ConfigureHTTPResponse** Activity by clicking it and configure it as follows:
 - a. On the **Settings** tab:
 - i. If your flow is attached to multiple REST triggers, select the trigger in which you have configured the code you want to use from the **Trigger Name** drop-down menu. The **Trigger Name** field does not display if your flow is attached to only one REST trigger.
 - ii. Select a response code from the **Code** field menu. Only the codes configured in the selected trigger are displayed in the menu.
 - b. The **Input** tab displays the schema for the response code. Map the elements or manually enter a value for the elements.
 - c. Click **Save**.
5. Configure the **Return** Activity by mapping the **code** and **body** (which is currently of data type any).



6. Click **Save**.
7. On the **Map from Flow Outputs** tab in the **ReceiveHTTPMessage** trigger, map the **code** and **body** to the corresponding elements from the flow output.



8. Click **Save**.

Testing the Deployed App

The deployed app can be tested using its API specification.

To test the deployed app:

Procedure

1. Open the app.
2. Click **Endpoints** to open the tab.




3. Click **Test**.

Downloading the API Specification Used

You can download the API specification used to create the app. To download the specification:

Procedure

1. Open the app.
2. Click **Endpoints** to open its tab.
3. Click the shortcut menu () to the extreme right of the Endpoint URL.
4. Click **Download spec**.

The downloaded specification may not be the same as the original specification that was used to create the app. This could happen because Flogo Enterprise follows its convention when generating a specification from its apps. Also, any changes that you might have made after creating the app are reflected in the downloaded specification but are not changed in the original specification from which you created the app. The original specification remains untouched. Use the downloaded specification only for testing the app.

Using GraphQL Schema

GraphQL provides a powerful query language for your APIs enabling clients to get the exact data that they need. It can get data from multiple resources in a single request by aggregating the requested data to form one result set. GraphQL provides a single endpoint for accessing data in terms of types and fields.

Flogo Enterprise provides an out-of-the-box GraphQL trigger that turns your Flogo app into a GraphQL server implementation. Each flow in the app acts like a GraphQL field resolver. So, the output of the flow must match the return type of the field in the schema.

Flogo Enterprise allows you to create GraphQL triggers by dragging and dropping your GraphQL schema file into the UI or by navigating to the file. A flow gets automatically created for every query and mutation type in your schema. You must then open the flow and define what kind of data you want the flow to return. This saves you the time and effort to programmatically define data structures on the server.

i Note: This section assumes that you are familiar with GraphQL. To learn about GraphQL, refer to the GraphQL documentation.

GraphQL server implementation in Flogo Enterprise

To obtain samples of GraphQL schemas and app JSON files, go to [GraphQL](#).

To use GraphQL in Flogo Enterprise, you must create a GraphQL trigger. Use one of the methods below to create a GraphQL trigger.

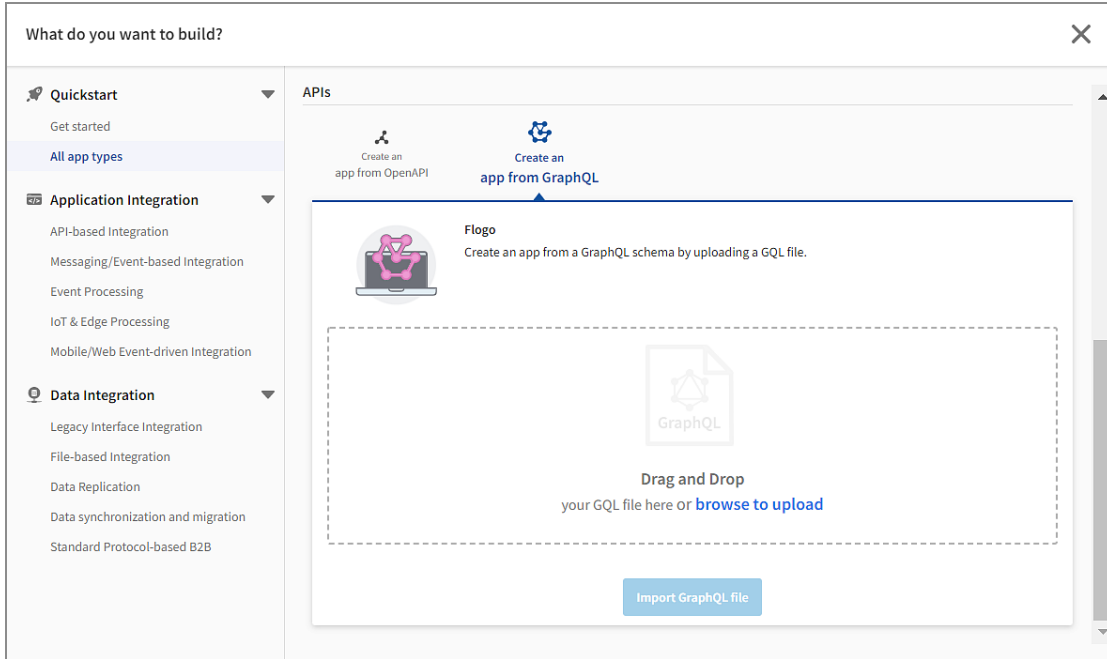
- You must use only one schema per app. If you attach your app to another GraphQL Trigger, you must use the same original schema.
- The implementation of the GraphQL server in Flogo Enterprise currently does not return the specified field ordering in a query when a request is received. It does not affect the correctness of the response returned, but affects the readability and is non-compliant with current specifications.
- The GraphQL schema must have either `.gql` or `.graphql` extension.

For details on the GraphQL trigger refer to the "GraphQL Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide*.

Creating a New Flogo App Using a GraphQL Schema

Procedure

1. Log in to TIBCO Cloud™ Integration.
2. On the **Apps** page, select **Create/Import**. The **What do you want to build?** dialog box is displayed.



3. Under **Quickstart** > **All app types** > **APIs**, click **Create an app from GraphQL**.
4. Click **browse to upload** and navigate to your locally stored GraphQL schema file to upload it.
5. Click **Import GraphQL file**. The new GraphQL trigger gets created with a flow attached to it.

Note: Once the trigger is created from the wizard, the trigger configuration is fixed and cannot be changed.

To implement a single method in your .gql file

To implement a single method:

Procedure

1. In Flogo Enterprise, open the app details page and click **Create**. The **Add triggers and flows** dialog box opens.
2. Under **Create new**, click **Flow**.
3. Enter a name for the flow in the **Name** text box. Optionally, enter a description for the flow in the **Description** text box.

4. Click **Create**.
5. Select **Start with a trigger**.
6. In the **Triggers catalog**, select the appropriate **GraphQL Trigger** card.
7. Follow the screen prompts to configure the trigger. A flow with the name you specified gets created and attached to the newly created GraphQL trigger. This flow implements the method that you selected.

✔ **Tip:** If needed, you can later make changes to the GraphQL schema file and upload it using the GraphQL trigger without creating a new flow.

To implement all methods defined in your .gql file

You can create flows to implement all methods defined in your .gql file. To do so:

Procedure


1. On the app details page, click **Create**. The **Add triggers and flows** dialog box opens.
2. Under **Start with**, click **GraphQL Schema**.
3. Upload your `<name>.gql` file by either dragging and dropping it to the **Add triggers and flows** dialog box or navigating to it using the **browse to upload** link. Flogo Enterprise validates the file extension. You see a green checkmark and the **Upload** appears.
4. Click **Upload**. Flogo Enterprise validates the contents of your schema and if it passes the validation, it creates the flows based on the methods defined in your schema file. One flow is created for each method in your schema. All the flows are attached to the same trigger.

Manually attaching a flow to an existing GraphQL trigger

If you have an existing flow in an app, you can manually attach it to a GraphQL trigger. To do so:

Procedure

1. Click the flow name to open the flow details page.

2. Click the  icon to the left of your flow. By default, the existing GraphQL triggers in the app are displayed.
3. Select one of the existing GraphQL triggers and follow the on-screen directions.

Limitations on constructs in a GraphQL schema

Flogo Enterprise currently does not support the following GraphQL constructs:

- Custom scalar types
- Custom directives
- Subscription type
- Cyclic dependency in the schema. For example, if you have a type `Book` that contains an object element of the type, `Author`. The type `Author` in turn contains an element of type `Book` which represents the books written by the author. To retrieve the `Author`, it creates a cyclic dependency where the `Author` object contains the `Book` object and the `Book` type, in turn, contains the `Author` object.

Using App Properties and Schemas

This section discusses how to create app properties, which you can use when populating field values. It also describes how to create a schema that can be reused in your app.

App Properties

App properties provide a way to override property values included in the app binary. You can configure some supported fields with app properties when configuring triggers and activities. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if need be during runtime. Configuration fields in your flow that require their values to be changed when the app goes from a testing stage to production are best configured using app properties instead of hard coding their values. App properties for triggers and activities reside within the app. App properties for connections are not modifiable from the **App Properties** dialog box in the app.


The URL field in an Activity is a good example of a field for which you would want different values – may be an internal URL when testing the app and an external URL when the app

goes into production. You may want the URL used in the Activity to change when the app goes from a test environment to production. In such a case, it is best to configure the URL field in the Activity with an app property instead of hard-coding the URL. This way, you can change the URL by changing the value of the app property used to configure the URL field.

Before building the app, you can change the default value of an app property from the [App Properties dialog box](#). Once you have built the app and have the app binary, use the CLI to change the value of an app property in the app.

An app property value can have one of the following data types:

- string
- boolean
- number
- password

Values for the password data type are encrypted and are not visible by default. But when configuring the password value, you can click on the **Show/Hide password property value** icon () to see the value temporarily to verify that it has been entered correctly.

App properties are saved within the app, so when you export or import an app, app properties configured in the app also get exported or imported with the app. Properties of data type password do not retain their values when an app is exported. So, you must reconfigure the password after importing the app.

If you import an app that was created in a prior version, even though this feature is available to the app since the activities were created in an older version of Flogo Enterprise you need to re-create them to be able to see the slider against their fields which allows you to configure an app property for that field.

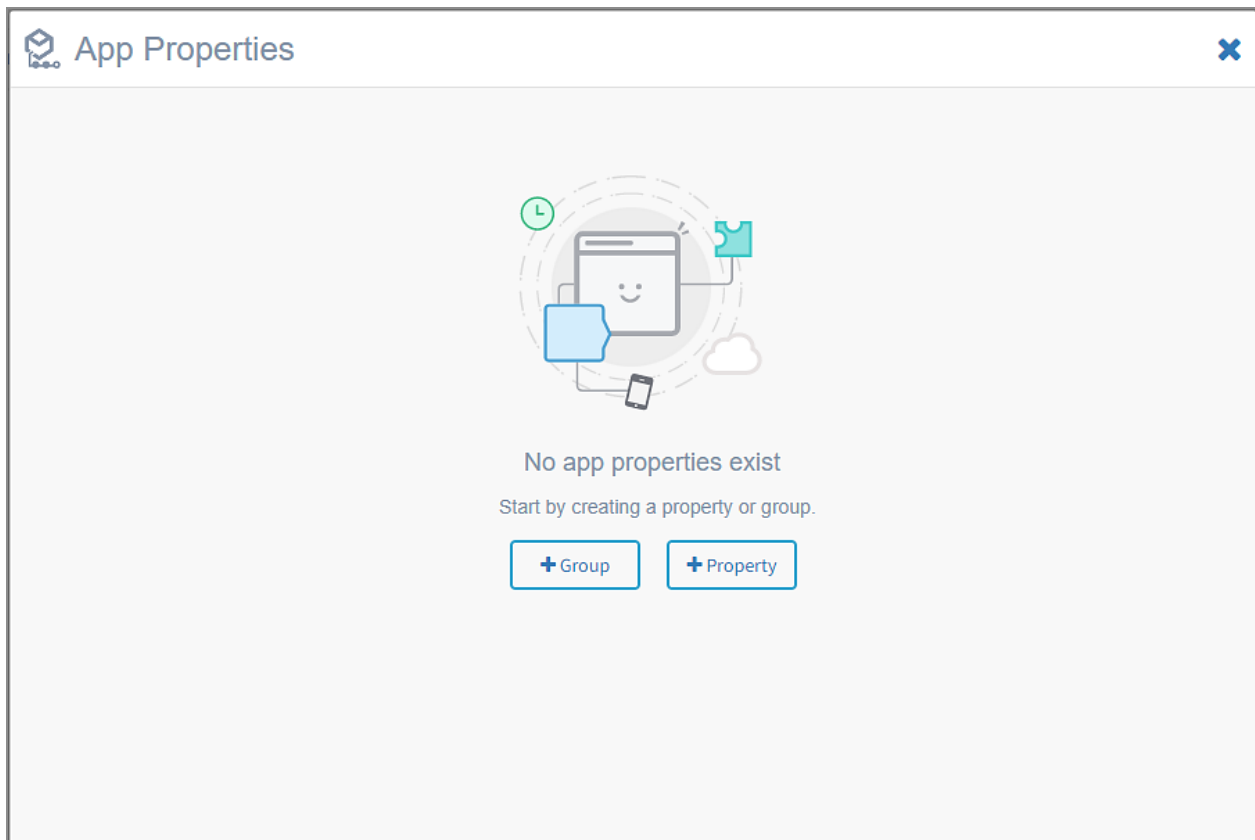
Creating App Properties

You can create an app property as a standalone property or as a part of a group. Use a group to organize app properties under a parent. A parent acts as an umbrella to hold related app properties and is labeled with a meaningful name. A parent does not have a data type associated with it. For instance, if you want to group all app properties associated with a particular Activity, you can create a group with a parent that has the Activity name and create all that Activity-related app properties under that parent.

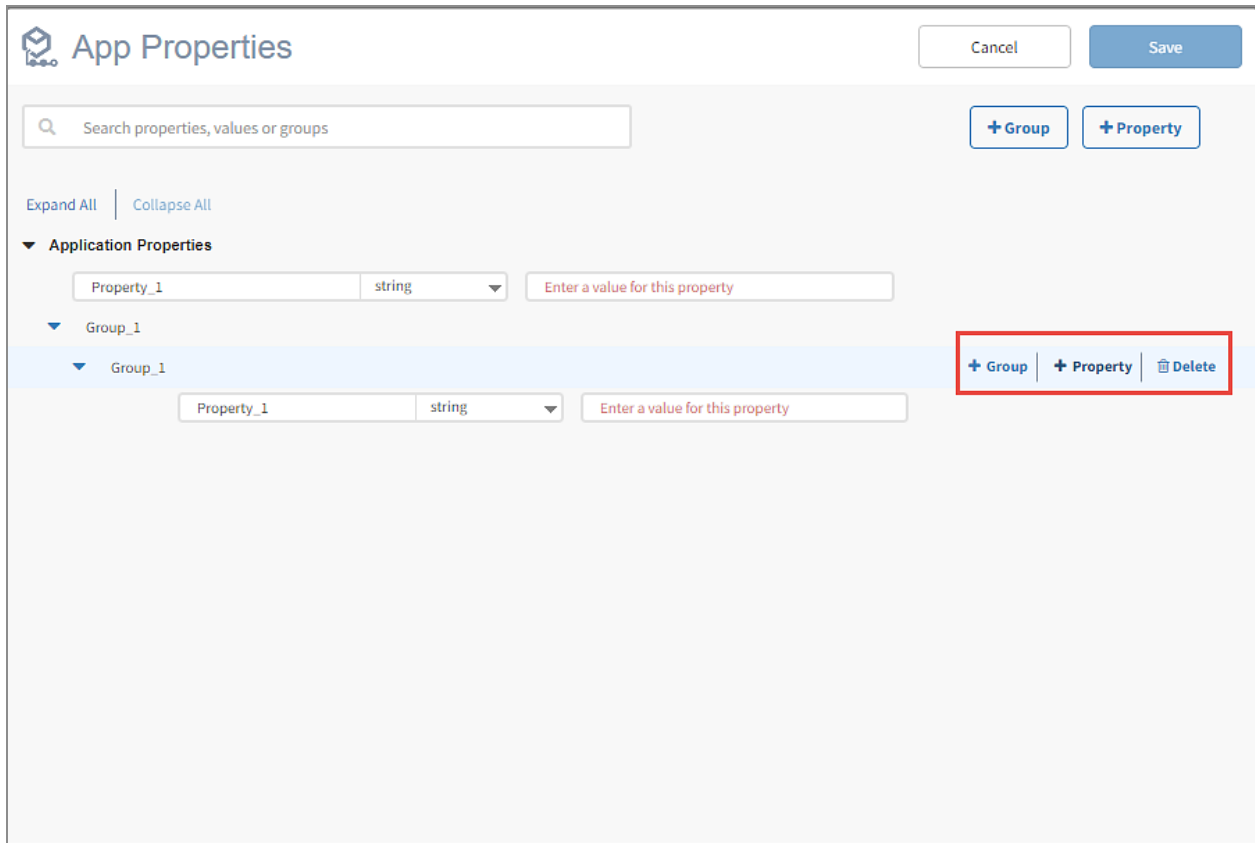
As an example, you can create LOG_LEVEL as a standalone app property without a parent. Or you can create it as a part of a hierarchy such as LOG.LOG_LEVEL with the parent of the hierarchy being LOG and LOG_LEVEL being the app property under LOG. Keep in mind that if you group properties, you must refer to them using the dot notation starting from the parent. For example, the LOG_LEVEL property must be referred to as LOG.LOG_LEVEL. You can nest a group within a group.

App Properties Dialog Box Views

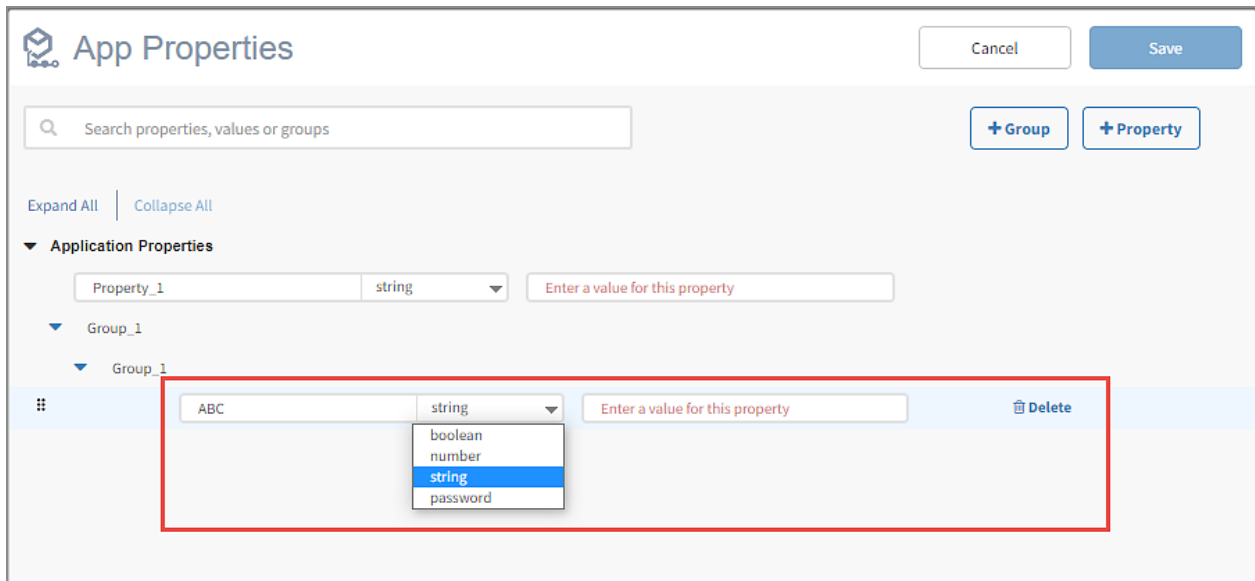
You can view existing app properties for an app in the **App Properties** dialog box. By clicking on the **+Group** or **+Property**, the app properties dialog box lets you add a new group or a property and rename it. An empty app properties' dialog box looks like this:



Nested groups and properties can also be created from the app properties dialog box by clicking on the **+group** or **+property** of each group.



The name of the property added can be changed from default to anything you want. Even the type of property value can be changed by selecting it from the drop down. You can drag a property with unique names from one group to another but not within the same group.



Creating a Standalone App Property

To create a standalone app property for your app, follow the steps below.

To create a group, see [Creating a Group](#).

Note: The standalone properties (properties that are not in a group) or the properties within the same group must have unique names.

Procedure

1. If your app does not exist, create a new app, and click **Properties** shown on the screen below.



If your app already exists, then open the app details page and click **Properties**. The **App Properties** dialog box opens.

If you already have existing properties, they are displayed. Click **+Property** to add another property.

- Click on the newly created property to make it editable and rename it. The property gets created.

Note: The property name must not contain any spaces or special characters other than a dash (-) or an underscore (_).

- Select the data type for the new property from its drop-down list.
- Enter a default value for the property in the text box next to the property.

Note: Only for certificates, the value must be of the format: <encoded_value>. To get the encoded value of the contents, you can use <https://www.base64encode.org/> or any other base64 encoding tool.

For example, for an SSL certificate, you can specify the app property as follows:

5. Click **Save**.

Note: Flogo Enterprise runs validation in the background as you create a property. The validation takes into consideration the property type and default value of the property that you entered. **Save** gets enabled only when the validation is successful. Make sure you do not skip this step of saving your newly created property or group.

Creating a Group

You can create one or more standalone app properties or group app properties such that they show up in a hierarchy. A group (or hierarchy) consists of a parent node, which is just a label and does not have a data type associated with it. You must create properties within the parent. You can do so in the **Application Properties** dialog box. When creating a group you must add the parent first and then create the app properties under the parent.

i Note:

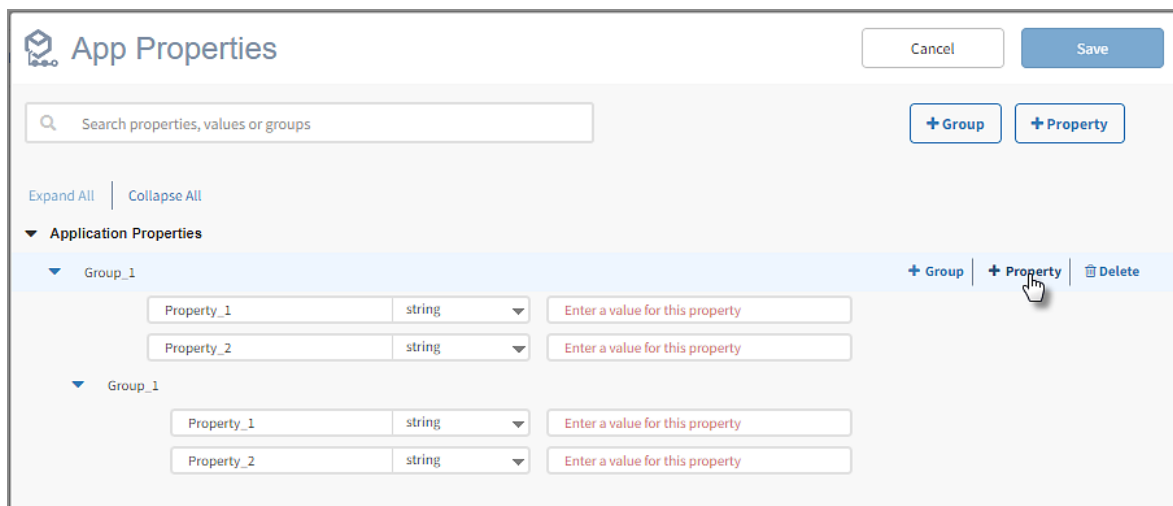
- With drag option, a standalone property can be rearranged to another location or a property under the group can be moved to another group.
- A group with its nested groups and properties can be dragged to move from one location to another. Also a nested group can be moved up in the hierarchy or to the root level. However, no two groups can have same name on same level.
- Group names within an app must be unique. Also, property names within a group must be unique.
- You cannot create a group and an app property with the same name in the same hierarchy.

Procedure

1. Open the app details page and click **App Properties**.
2. Click **+Group** on the upper-right corner to add the group.
3. Click on the newly created group name to make it editable and Enter a meaningful name for the group.

The group gets created. The group is simply a label and cannot be used by itself. So, you must add a group or a property within the group.

4. To add a property within the group, hover your mouse cursor to the extreme right of the group until **+Property** is displayed in the group row.



5. Click **+Property** to add the property and rename it.
6. Select a data type for the property and enter a value. Entering a value and selecting a data type is mandatory. **Save** remains disabled without it.
7. Click **Save**.

The property gets created under the parent.

i Note: You can even add a nested group under the parent group by clicking on **+Group** in the group row.

Deleting a Group or Property

An existing group or a property can be deleted in following ways.

To delete a property

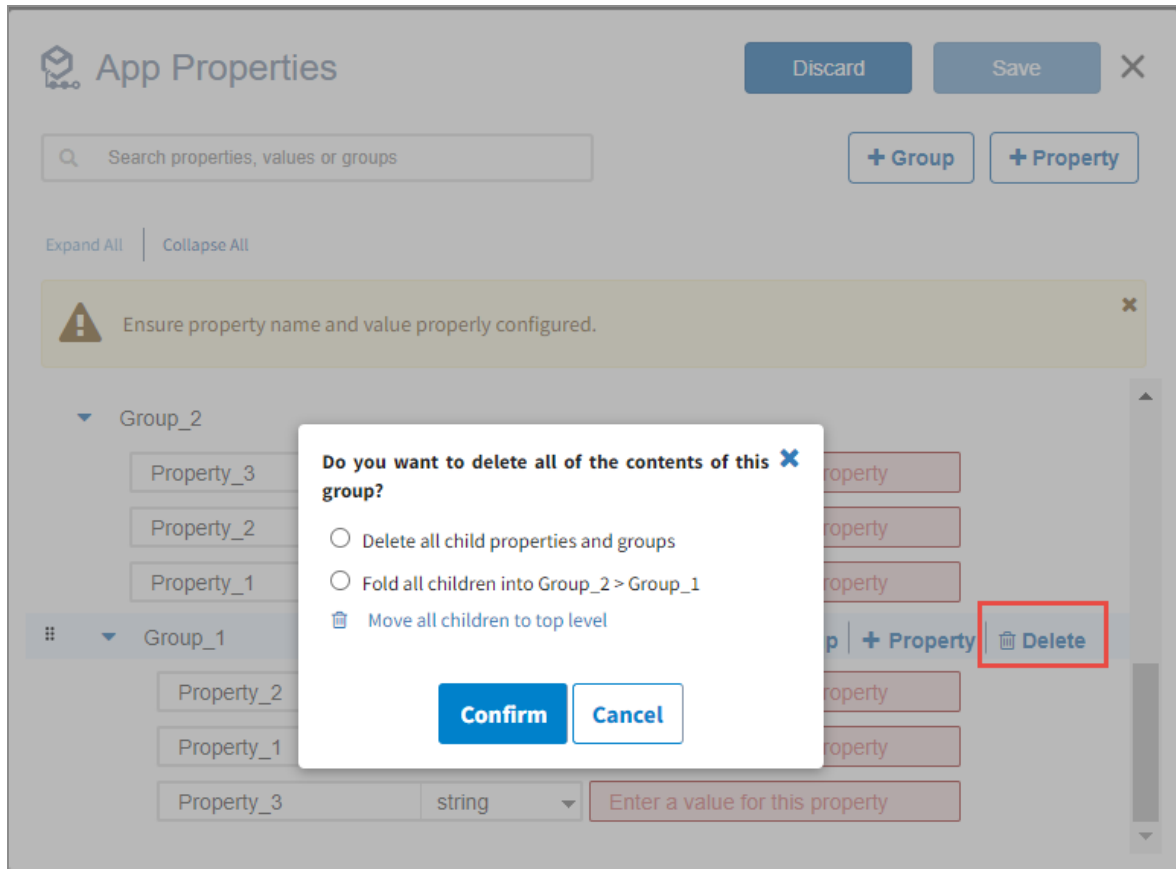
Procedure

1. Open the **App Properties** dialog box from the app details page.
2. Hover your mouse cursor to the extreme right end of the property and click **Delete**.
3. Click **Save**.

To delete a group or a nested group

Procedure

1. Open the **App Properties** dialog box from the app details page.
2. Hover your mouse cursor to the extreme right end of the group and click **Delete**. A confirmation window appears.



Here,

- **Delete all child properties and groups** deletes all the standalone properties and nested groups and properties under the group.
- **Fold all children into Group_2 > Group_1** deletes the nested group but the properties under the nested group are shifted into the parent group.
- **Move all children to top level** deletes the parent or a nested group and shifts all the properties to the top level as a standalone properties.

3. Select the desired delete option on the confirmation window and click **Confirm**.



Caution: The property path mappings may update on editing the property or on moving a property from a nested group to a parent group or if the property is shifted out of the group to top level as a standalone property.

Using App Properties in a Flow


Configuring a field with an app property is recommended for fields that require their values to be overridden when the app goes into production. Hence, the decision as to which fields in an Activity should support app properties (which fields can be configured using an app property) must be decided at the time when the extension for the category is being developed. The fields that can be configured using an app property display a slider against their names in the UI.

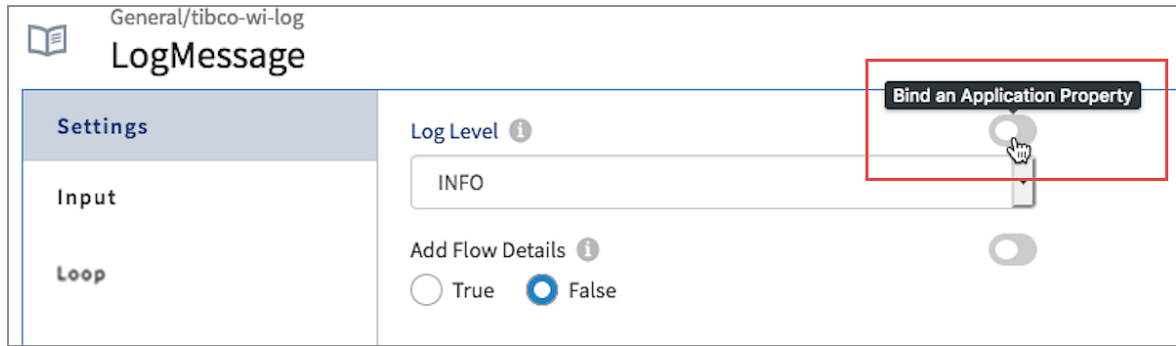
You can use environment variables to assign new values to your app properties at runtime. For more information, refer to [Overriding Security Certificate Values](#). You can also override the app property values at runtime using a JSON file. For more information, refer to [Using a JSON File to Override App Property Values](#).

Connection-specific app properties are visible in the **App Properties** dialog box after you select a connection when configuring the Activity or trigger, but they appear in read-only mode. This is because connections are reusable across apps and connection-related app properties are managed (refreshed) automatically. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if need be during runtime. For more details on how the connection properties get created and used, see [Using App Properties in Connections](#).

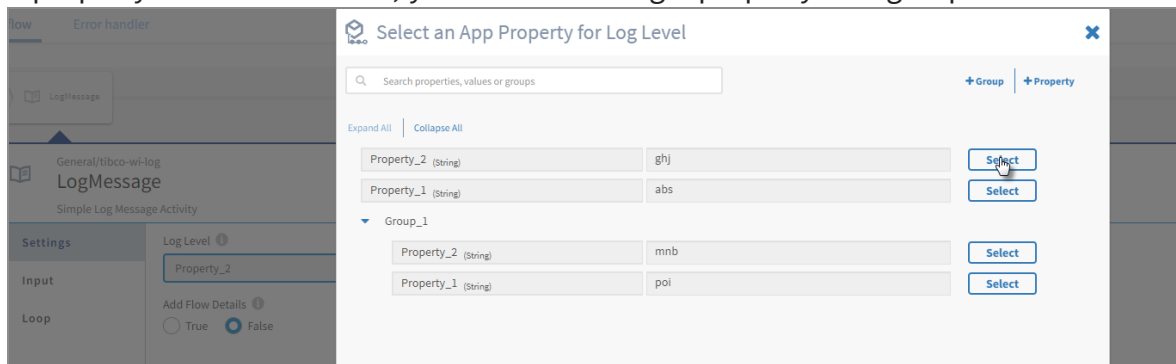
To configure a field with an app property:

Procedure

1. Open the flow details page.
2. Click the Activity whose field you want to configure with an app property.
This opens the configuration pane for the Activity.
3. Click the slider () against the name of the field you want to configure with an app property. If the field does not display a slider, the field cannot be configured with an app property.



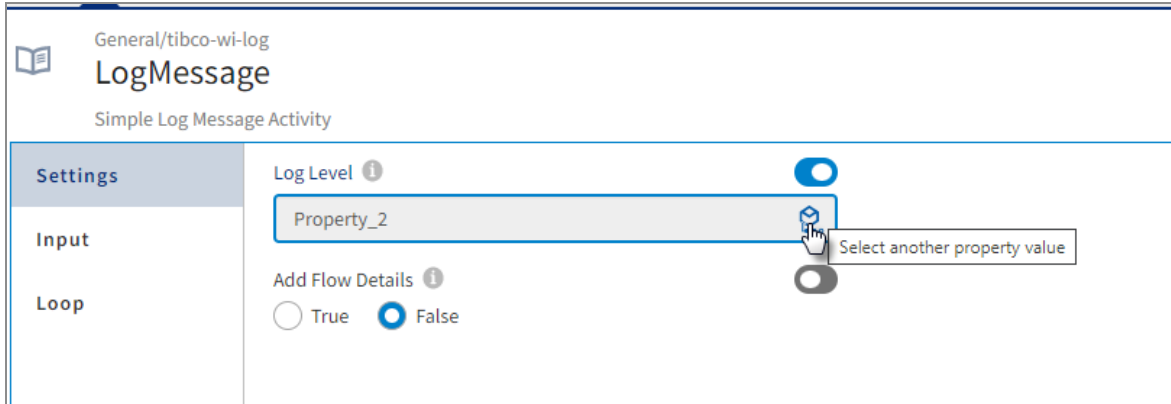
4. The **App Properties** dialog box opens. Only those app properties whose data type matches the data type of the field are displayed. You can also create a new group or a property in this view. Here, you can add a single property or a group at a time.



5. Select the property you want to configure for the field.

The property name appears in the text box for the field and the default value of the property gets implicitly assigned to the field.

After configuring the property, if you want to change a field to use a different property, hover your mouse cursor over the end of the text box for the field until the **Select another property value** icon appears. Click the **Select another property value** icon.



For a field that has been configured with an app property, you can unlink the property from the field. Refer to [Unlinking an App Property from a Field Value](#) for more details.

Using App Properties in the Mapper

You can use app properties when mapping an input field. The app properties available for mapping are grouped under the **\$property** domain-specific scope in the mapper. All mapper rules and conditions apply to the use of app properties as well. For example, the data type of the app property value must match with the input field data type when mapping. Connection-related app properties that are used by any connection field in an Activity do not appear under **\$property** since they cannot be accessed. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if need be during runtime. Hence, they cannot be used to map input fields.

Refer to the section on [Mapper](#) for details on how to use the mapper.

Unlinking an App Property from a Field Value

For a field that has been configured with an app property, if you decide at a later time not to use the app property, you can click and slide its slider ball () to the left. This removes the app property from the field (unlink it from the field) but leaves the field configured with the default value of the app property. The field retains the default value of the app property, but it gets disassociated from the app property and appears as a manually entered value. Hence, if you change the default value of the app property beyond this point, it does not affect the value of the field.



Using App Properties in Connections


Connection-related app properties can be used to modify or configure app properties anywhere within an app. If needed, the connection related app properties also allow you to change the connection values during runtime. Before you build, your app, their values can only be edited in the connection details dialog box, the dialog box where you provided the credentials for the connection. You can open this dialog box by editing the connection from the **Connections** page in the UI. Connection-related properties are useful when you want to change the value for one of the connection fields, for example, a URL, when an app goes from the testing stage to production.

You can use environment variables to assign new values to your app properties at runtime. For more information, refer to [Overriding Security Certificate Values](#). You can also override the app property values at runtime using a JSON file. For more information, refer to [Using a JSON File to Override App Property Values](#).

How the connection-related app properties get created

You cannot explicitly create connection-related properties. When you select a connection in the **Connection** field of an Activity, the supported properties associated with that connection automatically get created and populated in the **App Properties** dialog box.

While creating a connection, the fields in the connection details dialog box that support app properties are marked with  icon. One property gets created for each field that is marked with  in the connection details dialog box. The values you enter for such fields in the connection details dialog box become the default values for the connection properties. The properties take their name from the connection field they represent in the connection details dialog box.

You begin by creating a connection. In the example below, only the **Connection URL** and **Authentication Key** fields support app properties. These are the only two fields that display  against them.

The screenshot shows a dialog box titled "TIBCO Cloud Messaging Connector" with a close button (X) in the top right corner. It contains the following fields:

- Connection Name:** A text input field containing "MyTCMConnection".
- Description:** An empty text input field.
- Connection URL:** A text input field containing "http://google.com". A red box highlights a TIBCO logo icon to the right of this field.
- Authentication Key:** A text input field containing a series of dots. A red box highlights a TIBCO logo icon to the right of this field.

At the bottom right, there are two buttons: "Cancel" and "Save".

Once the connection is created, you can use it to configure the **Connection** field in an Activity. In the example below, the connection created above is being used to configure the **Connection** field of the **TCMMessagePublisher** Activity.

The screenshot shows the settings for the "TCMMessagePublisher" activity. The title bar reads "Messaging/tibco-messaging-tcm-pub" and "TCMMessagePublisher". On the left is a "Settings" sidebar with options: "Settings", "Input Settings", "Input", and "Loop". The main area is titled "Connection" with an information icon (i). A dropdown menu is open, showing "MyTCMConnection" selected. A red box highlights the dropdown menu.

After configuring the Connection field with the connection, if you open the **App Properties** dialog box, the connection properties for the field (enclosed in the red box in the image below) is displayed. Notice that only the supported properties (Connection URL and Authentication Key) are displayed in a read-only mode.

The screenshot shows the 'App Properties' dialog box with the following configuration:

Property Name	Data Type	Value
AppLogLevel	string	WARN
LogDefaultFlowInformation	boolean	true
FormatLogEntryAsJson	boolean	true
AppLogTarget	string	FlogoAppLog
LoggingAndErrorHandling		
RevisionInfo		
Version	string	1.0.5
BriefComments	string	YIPS-113: Corrected error event LEH service path in the app pro...
Messaging		
TCM_Ylopo		
Connection_URL	string	wss://01DJDV59ZH63JJ7D4CAX36D332-apps.messaging.cloud.tibco...
Authentication_Key	password	Enter a value for this property
AutoReconnectAttempts	number	20
Timeout	number	10

The properties that are displayed in the **App Properties** dialog box change dynamically based on your selection of the connection to use. You can only view the connection properties. You cannot edit or delete them from the **App Properties** dialog box. Deleting the Activity that uses the connection automatically removes the associated connection properties that the Activity used from the **App Properties** dialog box.

Using connection-related app properties

Connection-related app properties are available for use from the mapper. You can use these properties to change a connection value (for example, a URL or password) just before an app goes from a testing stage to production. All the mapped configurations can be pre-checked using a flow tester or by creating a pre-check flow. Their values cannot be changed from the **App Properties** dialog box, change their values in the connection details dialog box before building the app.

Editing an App Property

You can change the default value or data type of an app property at any time.

After the app has been built, you can override an app property from the CLI.

Changing the Default Value of a Property from the App Properties Dialog Box

You can change the default value of an existing app property at any time after creating the property. Before you build the app, you can change the default value in the **App Properties** dialog box.

To change the default value of an existing app property:

Procedure

1. Open the **App Properties** dialog box by clicking **Properties** on the app details page.
2. Click inside the text box for the property value you want to change.
3. Edit the value.
4. Click **Save**.

Changing the Name or Data Type of an App Property after Using It

If you change either the name of an app property or its data type after you have used the property to configure a field in an Activity or trigger, the field displays an error message. You must explicitly reconfigure the field to use the modified property by deleting the property from the text box for the field and adding the modified property.

When Importing an App

An app being imported could have its app properties. The app properties get imported along with the app. If an app property in the app being imported has a name that is identical to a property in the host app, a warning message is displayed with a choice to either overwrite the existing host property (by clicking **Continue**) with the property definition from the imported app or cancel the import process altogether.

App properties of type `password` do not retain their values when the app is exported, hence you must reconfigure the default values of all app properties of data type `password` after you import the app.

Overriding an App Property Value While Testing a Flow

Procedure

1. On the flow details page, click **Test**.
2. Start a new Launch Configuration by clicking **Create a Launch Configuration** or using an existing Launch Configuration that you had exported from another flow by clicking **Import a Launch Configuration**.

The Launch Configurations dialog box opens. For more information about Launch Configurations, see [Flow Tester](#).

App properties defined in the app and those defined in a connection are listed under **properties**.

3. Select the property whose value you want to override and specify the new app property value on the right side.
4. Click **Next**.
The input values you entered are displayed and validated. If no errors are found you get the message, **Input settings are alright**.
5. Click **Run** to execute the flow with the input data you provided in the step above.

6. Click **Stop Testing**.

App Schemas

You can define a JSON or Avro schema such that it is available for reuse across an app. Creating an app-level schema saves you time and effort of entering the same schema multiple times. An app-level schema can be used in any flow, Activity, or trigger configuration where a schema editor is provided. You can simply pick an existing schema from a list. For example, app-level schemas are available from the following locations:

- Inputs or Outputs tab of a flow (including Error Handler flows and subflows)
- Input or Output Settings tab of an Activity
- Output or Reply Settings tab of a trigger

App-level schemas are filtered based on the type of Activity or trigger. For example, only JSON schemas are displayed in a REST trigger or Activity configuration.

Currently, Flogo Enterprise only supports the JSON and Avro types of schemas.

Defining an App-Level Schema

Procedure

1. On the App Details page, click **Schemas**.
The Schemas page opens.
2. Click **+Schema**.
3. In the **Schema Name** field, enter a schema name.
4. Select the type of schema. You can select either JSON or Avro schema. The default is JSON schema.
5. Enter the schema in the schema editor.



Note: If you enter JSON data in the editor, it is automatically converted to JSON schema.

6. Click **Save**.

Result

After the schema is defined, it can be used in any Activity or trigger configuration by using **Use an app-level schema** in the schema editor of the Activity or trigger.

Editing an App-Level Schema

When you make changes to an app-level schema, the changes are automatically reflected everywhere the schema is used.

To edit an app-level schema:

Procedure

1. On the App Details page, click **Schemas**.
The **Schemas** page opens.
2. Expand the schema to be edited.
3. Edit the schema name or the schema in the editor, as required.
4. Click **Save**.

If the app-level schema is used in any flow, Activity, or trigger, a warning is displayed.

Deleting an App-Level Schema



Warning: Deleting a schema removes its reference from all the places where it is used, but it retains a copy of the schema in the fields that use the schema.

Procedure

1. On the App Details page, click **Schemas**.
The **Schemas** page opens.
2. Click the **Delete** icon beside the schema to be deleted.

Result

After confirmation, the selected schema is deleted.

Using an App-Level Schema

You can use an app-level schema from a flow, trigger, or Activity from the following tabs:

- Inputs or Outputs tab of a flow
- Input or Output Settings tab of an Activity
- Output or Reply Settings tab of a trigger

Flow Input & Output Tab

Use these tabs to configure the input to the flow and the flow output. These tabs are particularly useful when you create blank flows that are not attached to any triggers.

i **Note:** The schemas for input and output to a flow can be entered or modified only on this **Flow Inputs & Outputs** tab. You cannot coerce the flow input or output from outside this accordion tab.

Both these tabs (the **Input** tab and the **Output** tab) have two views:

- **JSON schema view:**

You can enter either the JSON data or the JSON schema in this view. Click **Save** to save your changes or **Discard** to revert the changes. If you entered JSON data, the data is converted to a JSON schema automatically when you click **Save**.

- **List view:**

This view displays the data that you entered in the JSON schema view in a list format. In this view, you can:

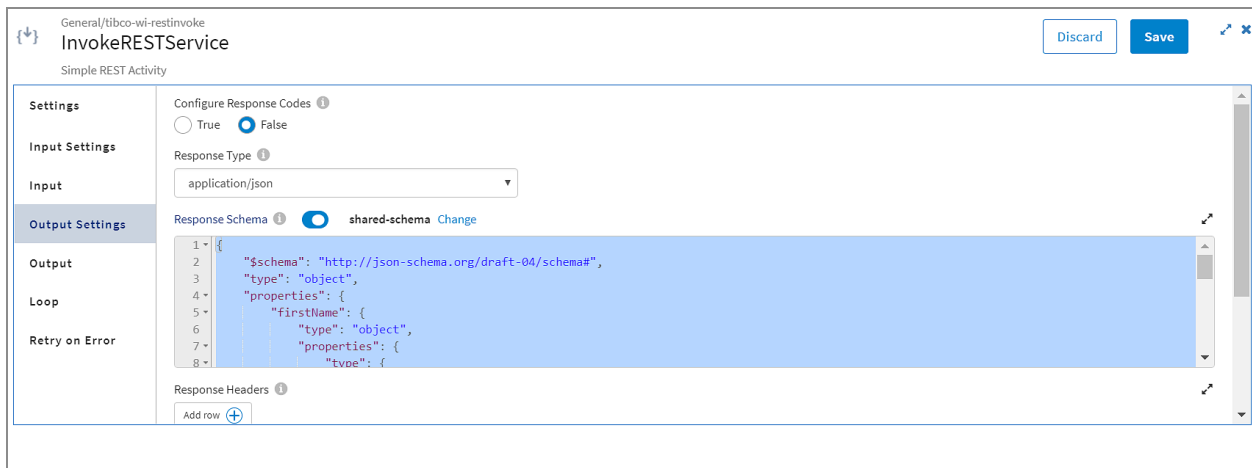
- Enter your data directly by adding parameters one at a time
- Mark parameters as required by selecting its checkbox.
- When creating a parameter, if you select its data type like an array or an object, an ellipsis (...) appears to the right of the data type. Click the ellipsis to provide a schema for the object or array.
- Use an app-level schema by selecting **Use an app-level schema**. On the **Schemas** page that appears, click **Select** beside the schema that you want to use. The name of the schema is displayed beside **Use an app-level schema** and the schema is displayed in a read-only mode.

i Note: You cannot edit an app-level schema in the **List** view if **Use an app-level schema** is selected. To edit an app-level schema, follow the instructions in the section [Editing an App-level Schema](#). You can, however, switch to another app-level schema by clicking **Change** and selecting another app-level schema. You can also unbind the app-level schema (by deselecting **Use an app-level schema**) from a trigger, activity, or the input and output of a flow. After you unbind the app-level schema, you can make changes to it using the schema editor in the **List** view.

- Click **Save** to save the changes or **Discard** to discard your changes.

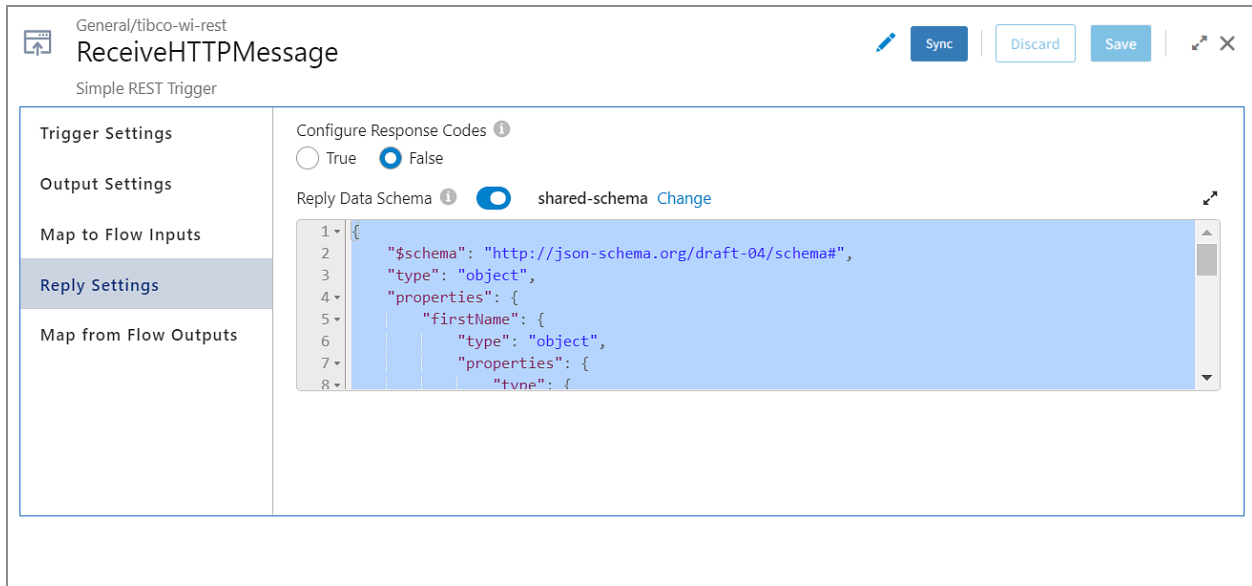
Input or Output Settings Tab of an Activity

When configuring an Activity, you can select an app-level schema on its **Input** or **Output Settings** tab. For example, the following screenshot shows an app-level schema selected in the **Response Schema** field of the **Output Settings** tab of an InvokeRESTService Activity.



Output or Reply Settings Tab of a Trigger

When configuring a trigger, you can select an app-level schema on its **Output** or **Reply Settings** Tab. For example, the following screenshot shows an app-level schema selected in the **Reply Data Schema** field of the **Reply Settings** tab of a ReceiveHTTPMessage trigger.



General/tibco-wi-rest
ReceiveHTTPMessage
Simple REST Trigger

Trigger Settings

Output Settings

Map to Flow Inputs

Reply Settings

Map from Flow Outputs

Configure Response Codes ⓘ
 True False

Reply Data Schema ⓘ shared-schema [Change](#)

```

1- {
2-   "$schema": "http://json-schema.org/draft-04/schema#",
3-   "type": "object",
4-   "properties": {
5-     "firstName": {
6-       "type": "object",
7-       "properties": {
8-         "tvne": {

```

i Note: If there is a change in the schema attached to a trigger, click **Sync** to synchronize it with the input and/or output of the flow.

Using Connectors

i Note: This section is applicable only if you have uploaded custom extensions for connectors. The **Extensions** tab displays your uploaded extensions.

To use the Flogo connectors:

Procedure

1. [Create one or more connections.](#)
2. If you do not already have an app, [create an app.](#)
3. [Create a flow.](#)
4. [Add the activities](#) about the connector you use as needed.
5. [Build the app.](#)

Creating Connections

You must create connections before using the connectors in a flow. Flogo Enterprise uses the configuration provided in these connections to connect to the respective app, data sources, systems, or SaaS.

Before you begin

You must have valid accounts for the SaaS apps to which you want to connect. To create a connection, click the **Connections** tab on the Flogo Enterprise page.

To create a connection using a connector tile:

1. If this is the first connection you are creating, click the **Create connection** link. For subsequent connections, click **Create** on the **Connections** page.
2. Click the connector tile for which you want to create a connection.
3. Follow the instructions to configure the connection when prompted.

i Note:

- You can have a maximum of four active Salesforce connections for one user at any time. If you create more than four connections for the same user, the first connection that you created gets deactivated. This limit is enforced by Salesforce.
- Make sure that the pop-up blocker in your browser is configured to always allow pop-ups from an app site. On macOS, clicking the link to the site results in the connection details page hanging, so make sure to select **Always allow pop-ups from <site>**.

Editing Connections

You can edit the name and other settings of your connection.

To edit an existing connection:

Procedure

1. In Flogo Enterprise, click the **Connections** tab to open its page.
2. In the list of existing connections, click the connection that you want to edit. Edit the

connection details in the connection details dialog box that opens.


3. Click **Save**.

i **Note:** Flogo supports automatic upgrade of activities, triggers, and connections. To view updates for connections, you must open the connection from the **Connections** page. For more information, see [Auto-Upgrade of Activities, Triggers, and Connections](#).

Deleting Connections

You can delete an existing connection.

Procedure

1. In Flogo Enterprise, click the **Connections** tab to open its page.
2. In the list of existing connections, hover over the connection name that you want to delete until you see the **Delete connection** icon () appear at the end of the row.

If the connection is being used by an app, you can see a blue icon in the **Usage** column. Hover over the icon to see which apps use the connection.

i **Note:** You cannot delete such connections.

3. Click the **Delete connection** icon.
4. On the confirmation dialog box, click **Delete connection**.

Result

The selected connection is deleted.

Using Extensions

You can create extensions for Flogo Enterprise or you can upload a Project Flogo extension into Flogo Enterprise.

You can create and contribute extensions for the following:

- activities
- triggers (you can define custom triggers that you can upload and use to create a flow)
- connectors (a connector provides configuration details to connect external apps, for example, Salesforce)
- functions (to be used inside the mapper when mapping elements)
- custom category extensions

After creating your extension, you upload its .zip file using the upload dialog box.

The extension you upload must follow the guidelines found on the GitHub page, [Building Extensions](#).

Important Considerations

Keep the following in mind before you upload your extension:

- A read-only user cannot upload an extension.
- When uploading your Activity or trigger extension, by default Flogo Enterprise compiles your extension before uploading it. If you would like to skip the compilation process, make sure to compile all the *.ts files in your extension and generate a .js file for each .ts file. The .js file must have an identical name as its corresponding .ts file.
- You are responsible for the life cycle (uploading, updating, deleting) of the extension that you contribute. Any extension that you contribute is visible and available for use only to you.
- When creating your Activity or trigger extension, if you did not specify a category for the extension, the extension is placed in the **Default** category.
- The category name for an extension must be unique. If a category by the name already exists, the upload completely overwrites the category. Out-of-the-box contributions cannot be overwritten.
- Special characters are not supported in Activity and trigger names. A validation error is displayed while uploading if any names contain special characters.

- Uploading new extension(s) to an existing category overwrites the entire category and all its contents. So, to add a new extension to an existing category while keeping the extension(s) that already exist in that category, be sure to include the existing extension(s) along with the new Activity, connection, or trigger when creating the .zip file to be uploaded.
- You cannot delete a single extension from any category other than the **Default** category. To delete a single trigger, Activity, or connector from a category, you must re-upload the whole category which includes all the extensions you want to keep minus the extension you want to delete. The same applies when editing an extension within a category - after editing an extension on your local machine, make sure to re-upload the whole category, the edited extension plus all the existing extensions in the category. Uploading only the edited extension overwrites the category causing you to lose the other extensions in the category.

An extension that you upload to Flogo Enterprise is available for use in any flow that currently exists in your app or any flow that you create later.

Creating Extensions

Flogo exposes a number of different extension points. You can easily extend the capabilities available by building your own activities. In this section, you explore the Activity contribution point and learn how to build a custom Activity in GO.

Step 1: Generate a basic framework

The easiest way to start creating Activities is to clone the content present in [TIBCO Extensions](#). The Activity built returns the concatenation of the two parameters and displays it on the console.

You must pull the following sample Activity to begin working on Flogo Core.

```
git clone https://github.com/TIBCOSoftware/tci-flogo.git
mkdir -p myNewActivity
cp -R /tci-flogo/samples/extensions/TIBCO/activity/* /myNewActivity
```

Step 2: Update the Metadata

After you have pulled an example from the Flogo core, the first step is to update the `descriptor.json` file with the required information. The file contains the metadata for the new Flogo Activity. The metadata in the file contains the following elements.

Element	Description
name	The name of the Activity This must match with the name of the folder in which the Activity has been added.
version	The version of the Activity. The semantic versioning for the activities must be used.
type	The type of contribution. For example: <i>flogo:Activity</i> in this case.
title	The application title to be displayed in the Flogo Web UI.
ref	The reference to the GO package that is used by the web UI to fetch the contribution details during the installation.
description	A brief description of the Activity. This is displayed in the Flogo Web UI.
author	The creator of the Activity.
settings	An array of name-type pairs that describe the Activity settings. <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Note:</p> <ul style="list-style-type: none"> • Activity settings are pre-compiled and can be used to increase performance. • The settings are not fetched for every invocation. </div>
input	An array of name-type pairs that describe the input to the Activity.

Element	Description
	The <i>anInput</i> parameter must be of the string type.
output	An array of name-type pairs that describe the output of the Activity. The <i>anOutput</i> parameter must be of the string type.

The updated `descriptor.json` file must look as follows:

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "name": "sample-Activity",
  "type": "flogo:Activity",
  "version": "0.0.1",
  "title": "Sample Activity",
  "description": "Sample Activity",
  "homepage": "https://github.com/project-
flogo/tree/master/examples/Activity",
  "settings": [
    {
      "name": "aSetting",
      "type": "string",
      "required": true
    }
  ],
  "input": [
    {
      "name": "anInput",
      "type": "string",
      "required": true
    }
  ],
  "output": [
    {
      "name": "anOutput",
      "type": "string"
    }
  ]
}
```

Step 3: Build the Logic

Now, you must update the .GO files available in the current directory. The . GO files in the directory are as follows:

File types	Description
Activity.go	Contains the logic behind Activity implementation in GO
Activity_test.go	Contains unit tests for the Activity
metadata.go	Contains the basic input, output, and settings metadata used by the engine

The first step is to update the metadata.go file. Define the input, output, and settings in the file. These details are used by the engine to build the Activity. Also it is used for leveraging contributions using the Flogo GO library. This enables GO developers to leverage strongly typed objects for IDE auto-completion.

The sample package of the metadata file must look like as follows:



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
import "github.com/project-flogo/core/data/coerce"

type Settings struct {
    ASetting string `md:"aSetting,required"`
}

type Input struct {
    AnInput string `md:"anInput,required"`
}

func (i *Input) FromMap(values map[string]interface{}) error {
    strVal, err := coerce.ToString(values["anInput"])
    if err != nil {
        return err
    }
}
```



```

        i.AnInput = strVal
        return nil
    }

    func (i *Input) ToMap() map[string]interface{} {
        return map[string]interface{}{
            "anInput": i.AnInput,
        }
    }

    type Output struct {
        AnOutput string `md:"anOutput"`
    }

    func (o *Output) FromMap(values map[string]interface{}) error {
        strVal, err := coerce.ToString(values["anOutput"])
        if err != nil {
            return err
        }
        o.AnOutput = strVal
        return nil
    }

    func (o *Output) ToMap() map[string]interface{} {
        return map[string]interface{}{
            "anOutput": o.AnOutput,
        }
    }
}

```

The next step is to look at the Business logic and update the Activity.go file.

The sample package of the Activity file must look like as follows:



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```

package sample

import (
    "github.com/project-flogo/core/Activity"
    "github.com/project-flogo/core/data/metadata"
)

func init() {

```

```

    //Activity.Register(&Activity{}, New) to create instances using
    factory method 'New'
    _ = Activity.Register(&Activity{})
}

var ActivityMd = Activity.ToMetadata(&Settings{}, &Input{}, &Output{})

//New optional factory method, should be used if one Activity instance
per configuration is desired
func New(ctx Activity.InitContext) (Activity.Activity, error) {

    s := &Settings{}
    err := metadata.MapToStruct(ctx.Settings(), s, true)
    if err != nil {
        return nil, err
    }

    ctx.Logger().Debugf("Setting: %s", s.ASetting)

    act := &Activity{} //add aSetting to instance

    return act, nil
}

// Activity is an sample Activity that can be used as a base to create a
custom Activity
type Activity struct {
}

// Metadata returns the Activity's metadata
func (a *Activity) Metadata() *Activity.Metadata {
    return ActivityMd
}

// Eval implements api.Activity.Eval - Logs the Message
func (a *Activity) Eval(ctx Activity.Context) (done bool, err error) {

    input := &Input{}
    err = ctx.GetInputObject(input)
    if err != nil {
        return true, err
    }

    ctx.Logger().Debugf("Input: %s", input.AnInput)

    output := &Output{AnOutput: input.AnInput}
    err = ctx.SetOutputObject(output)
}

```

```

    if err != nil {
        return true, err
    }

    return true, nil
}

```

Now, to test and build the Activity, you must get below GO packages.

```

go mod init
go mod tidy

```

Step 4: Perform Unit Testing

After you have completed the building logic of the Activity, you must now perform a unit test. Unit testing gives you an automated way to test the Activity to make sure that it works. This also lets other developers run the same tests to validate the output.

The sample package of the `Activity_test` file must look like as follows:



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```

package sample

import (
    "testing"

    "github.com/project-flogo/core/Activity"
    "github.com/project-flogo/core/support/test"
    "github.com/stretchr/testify/assert"
)

func TestRegister(t *testing.T) {

    ref := Activity.GetRef(&Activity{})
    act := Activity.Get(ref)

    assert.NotNil(t, act)
}

```

```

func TestEval(t *testing.T) {

    act := &Activity{}
    tc := test.NewActivityContext(act.Metadata())
    input := &Input{AnInput: "test"}
    err := tc.SetInputObject(input)
    assert.Nil(t, err)

    done, err := act.Eval(tc)
    assert.True(t, done)
    assert.Nil(t, err)

    output := &Output{}
    err = tc.GetOutputObject(output)
    assert.Nil(t, err)
    assert.Equal(t, "test", output.AnOutput)
}

```

To run all the test cases for your Activity, run below command:

```
go test
```

On a successful run the result must look like as follows:

```

PASS
ok      github.com/tibco/newConnector/myNewActivity    0.002s

```

Step 5: Upload the Activity in the Flogo App

Now, you can use the Activity in a Flogo app.

To install the Activity in Flogo, in the web UI, under **Environment and Tools**, go to the **Extensions** tab, and click **Upload**.

Uploading Extensions

Before you begin

When uploading an extension, you can see the logs on the screen. You can change the log levels at runtime by setting the `FLOGO_LOG_LEVEL` engine variable. Be sure to do so *before*

you begin uploading your extension. For details on the FLOGO_LOG_LEVEL engine variable, see the [Environment Variables](#) section. For more details on the environment and engine variables, see the [Configuring App Properties](#) section.

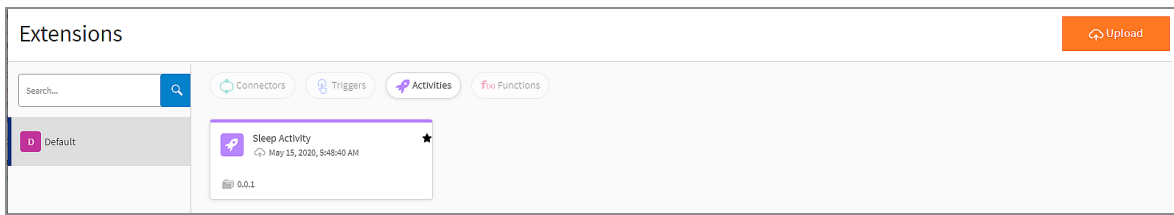
To upload an extension:

Note: This procedure assumes that you have the .zip file for your extension available for upload.

Procedure

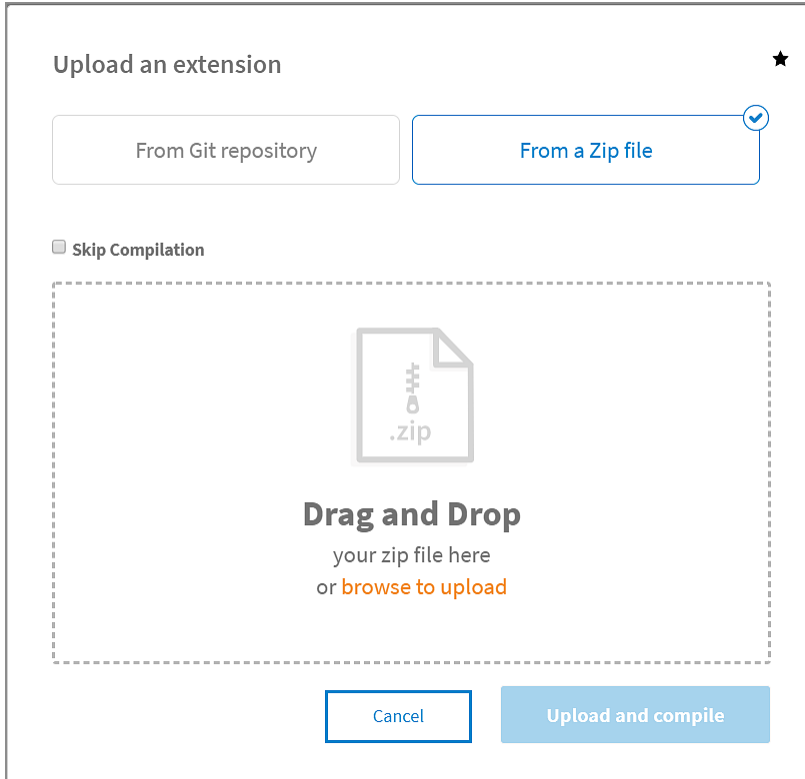
1. Click the **Extensions** tab.
2. If this is the first extension, click **Upload an extension**.

If there are existing extensions, click the **Upload** in the upper-right corner:



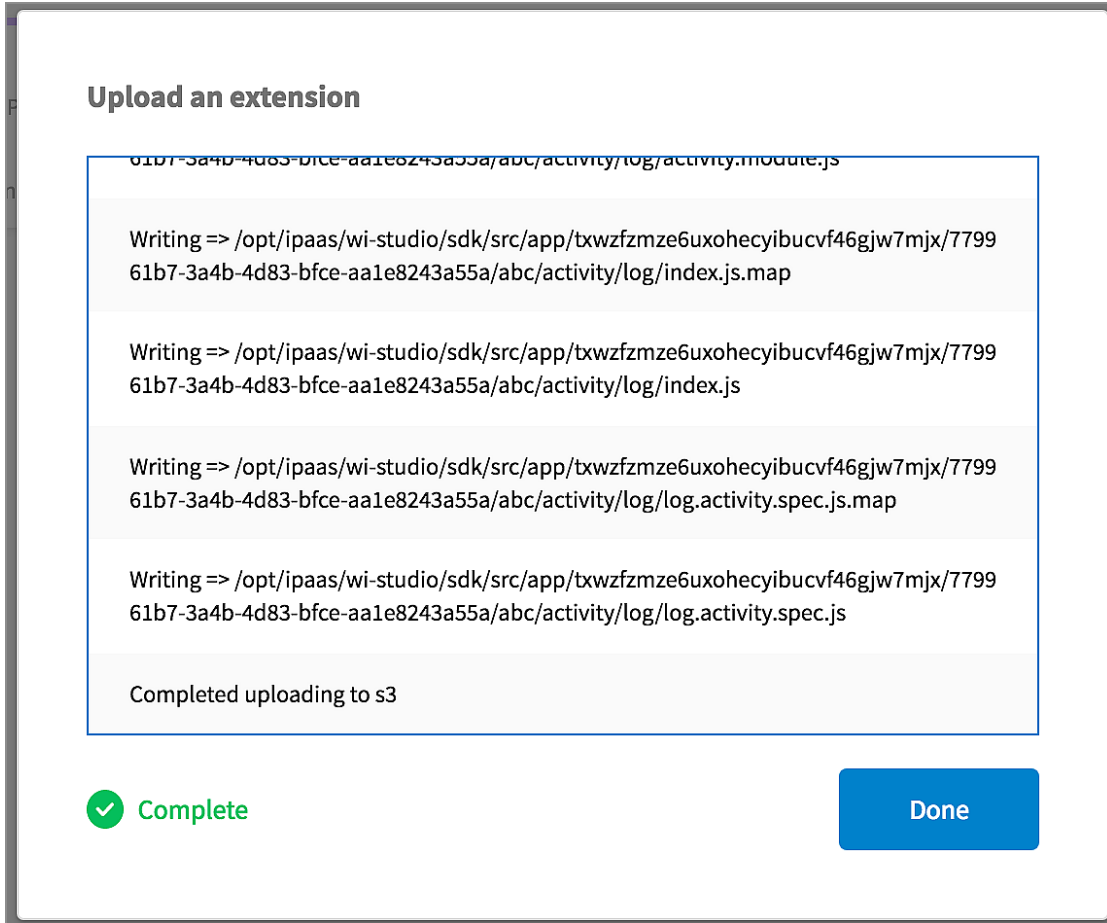
The **Upload an extension** dialog box opens. If you want to upload from the Git repository select **From Git repository**. See the section, [Pulling Extensions from an Open Source Public Git Repository](#) for more details on this.

To upload an extension residing in a .zip file locally, click **From a Zip file**.







3. Click the **browse to upload** link and navigate to your extension .zip file. Alternatively, drag the .zip file from your local machine to the area defined by a dotted line in the **Upload an extension** dialog box.
4. If you would like to skip the compilation process, select the **Skip Compilation** check box. If the check box is selected, Flogo Enterprise performs a check before uploading to make sure that every .ts file has a corresponding .js file present. If a .ts file does not have a .js file, the validation fails, and your extension does not upload.
5. Click **Upload and compile**.

Flogo Enterprise validates the contents in the .zip file. If the .zip file contains a valid folder structure, it compiles the extension code. Once the code is compiled successfully, it uploads the extension to Flogo Enterprise. You can view the progress of the upload or any errors that occur in the logs:



A **Complete** message is displayed after the extension is successfully uploaded. If there were any compilation errors during the upload, you see an error message and the upload exits. You can copy-paste the error message if required.


- Click **Done** to close the dialog box.

You can view your extension on the **Extensions** page. The newly added extension appears under the category that you specified. If you had not specified a category for the extension, it appears in the **Default** category. Connectors are denoted by the  symbol, triggers are denoted by the  symbol, activities are denoted by the  symbol, and functions have the  symbol next to them.

The new extension displays the following:

- timestamp when the extension was loaded
- name of the extension contributor

- version of the extension

i **Note:** While creating a flow, the  icon is shown on activities that are present on the **Extensions** tab.

The **Search** field that appears above the category searches within the categories for the Activity, trigger, or connector you specified in the search text box. You can filter the displayed extensions by clicking the **Triggers**, **Connectors**, or **Activities** buttons.

The extension is now available for you to use. If you uploaded an Activity, the Activity is available for use when creating a flow or editing an existing flow. The Activity appears under the category you defined for it when creating the extension. The output of the Activity is available in the mapper just as it is for any default activities that come with the Flogo Enterprise.

If you uploaded a connector, the connector is available for creating new connections on the **Add Connections > Select connection type** dialog box.

If you uploaded a trigger, the trigger is available for you to select in the **Create a Flow** dialog box. If you select the trigger, it creates the flow with your trigger.

If you uploaded a function, it is available to be used inside the mapper when mapping elements.

If you uploaded a category, it is available to use when adding any new activities while designing a flow. Triggers and connections in the category can be used as mentioned above.

Pulling Extensions from an Open Source Public Git Repository

You can upload extensions that are available from an open-source public Git repository by pulling them directly into Flogo Enterprise. This section describes how to pull the **Default** category Project Flogo extensions directly from an external public Git repository and upload it to Flogo Enterprise. Pulling from private Git repositories is currently not supported.

Keep the following in mind when pulling the contribution:

- You can download only from public repositories. Accessing private Git repositories is not supported.

- The Git repository link should be the reference of the Activity and not the URL.
- The repository link needs to be a reference of the contribution and must not begin with `http://` or `https://`, for example, to pull the LogMessage Activity from the Project Flogo Git repository, use `github.com/project-flogo/contrib/Activity/log`
- Any new default category contribution that you pull from the Git repository gets appended to the ones that already exist for the category in Flogo Enterprise. Contributions pulled and uploaded to other categories in Flogo Enterprise, overwrites the category itself. Hence, if there are any existing activities in the category, they get deleted when the category is overwritten.
- Default category extensions can only be downloaded one at a time.

To pull an extension from a public Git repository:

Procedure


1. On the **Extensions** page, click **Upload**.

The **Upload an extension** dialog box opens.

2. Click **From Git repository**.

When you select this option you are prompted to enter the location of the Git repository from which you want to pull the extension.

3. Enter the reference to the extension in the Git repository.

 **Important:** Make sure you do not enter the initial `http://`

4. Click **Import**.

Flogo Enterprise validates the format of the reference you entered in the **Git repository URL** text box. **Import** remains disabled until you enter a valid reference format. A `.zip` file for the Activity gets generated and uploaded to the **Default** category on the Extensions page in Flogo Enterprise. Once you start the process of downloading the contribution from the Git repository, you cannot cancel the process or switch to the process of uploading **From a Zip file**. You must wait for the upload process to complete, then click **Done**.

5. Click **Done**.



The extension you uploaded appears on the **Extensions** page.

Deleting Extensions or Extension Categories

From the **Extensions** page, you can delete:

- an existing extension from the **Default** category
- a custom extension category

Procedure

1. Click the **Extensions** tab.
The existing extensions are displayed on the **Extensions** page.
2. To delete an extension from the **Default** category, on the tile of the extension that you want to delete, click  and select **Delete**.
3. To delete a custom extension category, on the right side of the screen, click  and select **Delete**.

Individual items within a custom extension category cannot be deleted. The entire custom extension category must be deleted.

4. In the confirmation dialog box, click **Delete**.

Result

The selected extension or extension category is deleted.

Flow Tester

After you design a flow, use the Flow Tester to test the flow.

When designing a flow, runtime errors can go undetected until you build the app to execute the flow. It can become particularly cumbersome to test flows that start with a trigger since the triggers activate based on an external event. So, before you can test the flow, you need to configure the external app to send a message to the trigger to activate the trigger and consequently execute the flow. The Flow Tester eliminates the need to activate the trigger to execute the flow.

You provide the input to the flow in the Flow Tester. The Flow Tester executes the flow on demand without using a trigger. Each Activity executes independently and displays its logs. This lets you detect errors in the flow upfront without actually building the app.

i Note:

- The Flow Tester takes some time to start as the engine is built from scratch every time you start the Flow Tester. The time taken to start the Flow Tester depends upon the number of contributions used in the app and the resources assigned to the Docker daemon.
- The Flow Tester is disabled when Activity type contributions are missing in the flow execution.
- Expressions and functions are not evaluated in the Flow Tester. Input provided is passed as is.
- You can run the Flow Tester in the debug mode with the following features only:
 - Test run the flow
 - See the flow execution
 - Select an Activity that is executed and see the inputs and outputs
 - Change the inputs to other valid values and start the Activity from that point onwards
- You cannot:
 - Insert a debug point and stop the flow execution at a tile
 - Skip a tile from test execution

Testing Flows from the UI

You can use the Flow Tester from the Flogo Enterprise UI or you can use the CLI to run the test command in the Flow Tester. This section describes how to use the Flow Tester from the UI.

When using the Flow Tester from the UI, you must populate your test data in the Launch Configuration. Launch Configuration is a mechanism used by the Flow Tester to store your test data.

What is a Launch Configuration?

A Launch Configuration is a test configuration that contains a set of data with which to test the flow. Create a Launch Configuration to hold the test data that you want to use as input to the flow. Launch Configurations allow you to save and use your input data without having to enter it every time you want to test or retest the flow.

Blank flows use data configured on the **Input Settings** tab of the **Flow Inputs & Outputs** accordion tab as the input to the flow. Flows created with a trigger use the output of the trigger as input to the flow.

Launch Configurations are particularly useful when you want to test the flow multiple times with complex data or multiple sets of data. Create a Launch Configuration once with the data and use the Launch Configuration as input to the flow instead of manually entering the data every time you want to execute the flow. You can create multiple Launch Configurations, each containing a different set of data. A Launch Configuration can contain only one set of data. To test a flow with multiple sets of data, create multiple Launch Configurations for a flow with each containing one set of data, then test the flow with one Launch Configuration at a time.

Once you create a Launch Configuration, it automatically gets saved and is available to you until you explicitly delete it.

i Note: When exporting an app, if the app contains Launch Configurations, the Launch Configurations do **not** get exported with the app. Launch Configurations in an app must be exported independently of the app export.

Creating and Using a Launch Configuration

Launch Configurations need not be explicitly saved. They persist even after you exit Flogo Enterprise and log back in later.

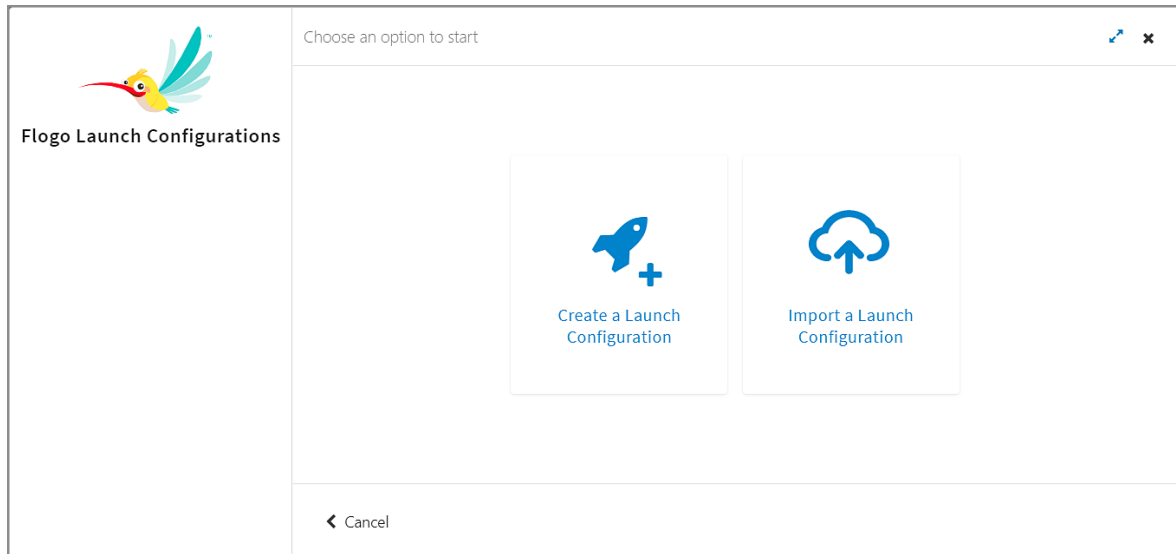
Creating your first Launch Configuration

To create the very first Launch Configuration in a flow:

Procedure

1. On the flow details page, click **Test**.

You can either start a new Launch Configuration by clicking **Create a Launch Configuration** or use an existing Launch Configuration that you had exported from another flow by clicking **Import a Launch Configuration**.



2. Click **Create a Launch Configuration**.

The Flow Tester opens with the left pane displaying the Launch Configuration name. By default, a new Launch Configuration is named "Launch Configuration x" where x stands for a number. For example, since this is the first Launch Configuration that you are creating, the name of the Launch Configuration displays as Launch Configuration 1. The next Launch Configuration you create is named Launch Configuration 2. You can edit the name in the right pane. The right pane opens the mapper which displays the flow input tree.

The screenshot shows the 'Flogo Launch Configurations' interface. The title bar reads 'Setting up Launch Configuration for flow REST'. The main configuration area includes:

- Launch Configuration name ***: A text input field containing 'Launch Configuration 1'.
- Log Level**: A dropdown menu currently set to 'INFO'.
- Using on-premise services**: A checkbox that is currently unchecked.
- Mapping settings**: A section with a search bar for 'Flow inputs' and a list of available inputs, including 'obj flowinputs'.
- A large grey area on the right with a diagram and the text 'Select an input to start mapping'.
- Navigation buttons: '< Cancel' on the bottom left and 'Next >' on the bottom right.

3. Optionally, enter a meaningful string to replace the default name in the **Launch Configuration name** text box.
4. Select the log level from the **Log Level** drop-down menu.
5. Select **Using on-premise services** if you want to test apps that connect to on-premise systems.

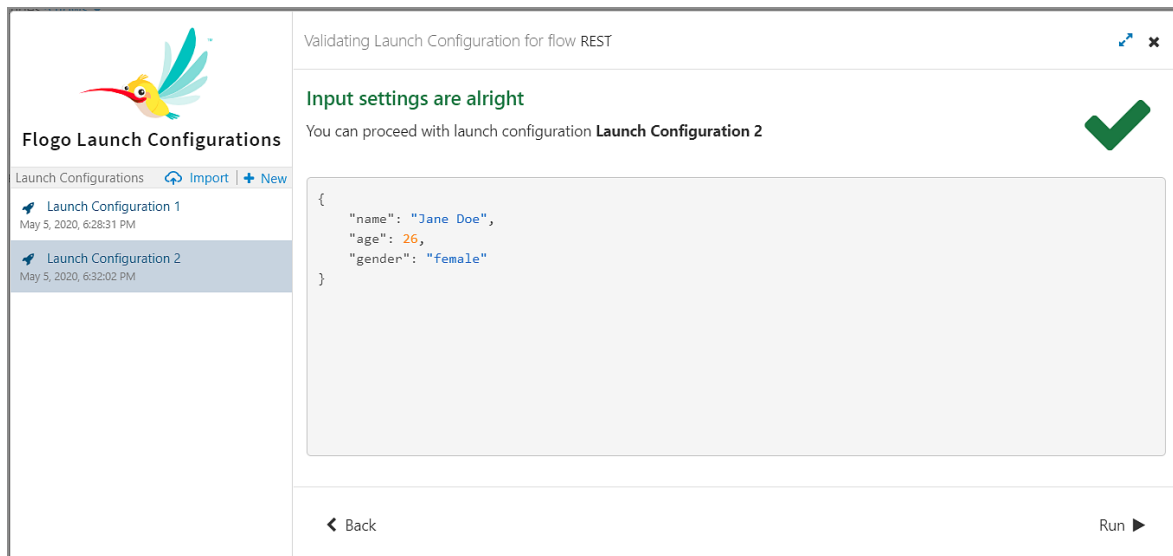
Note: Before you select this check box, enable the flogotester service for your organization using the API. For more information, see [Enabling or Disabling the TIBCO Flogo® Flow Tester for an Organization with the API](#).

6. Enter the values for the elements in the input tree. Refer to [Configuring a Launch Configuration](#) for details on entering values.

i Note: If your flow does not require an input, for example, if your flow was created with a Timer trigger that does not have an output (consequently no input to the flow), you can continue testing the flow without using the mapper in the Flow Tester.

7. Click **Next**.

The input values you entered are displayed and validated. If no errors are found you get the message, **Input settings are alright**.



8. Click **Run** to execute the flow with the input data you provided in the step above.

All the activities in the flow are executed. For details, see [What can you do using the Flow Tester?](#)

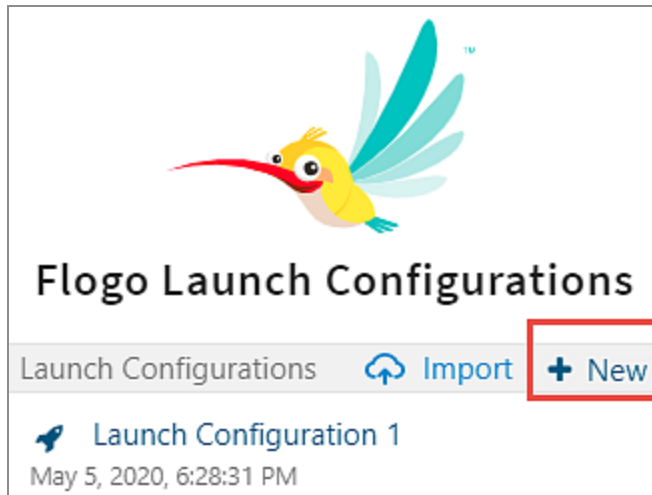
9. Click **Stop Testing**.

Creating Subsequent Launch Configurations

If you have an existing Launch Configuration:

Procedure

1. Click **New** to create another Launch Configuration.



2. Follow the procedure from [step 3](#) onwards in [Creating your first Launch Configuration](#).

What can you do using the Flow Tester?

When you use the Flow Tester to test a flow, all the activities in the flow are executed. While the flow tester is active, you cannot add or delete an Activity in the flow.

When an Activity is being executed, a blue animation is displayed around it. When the execution of the Activity is completed, the Activity is highlighted in the flow and the blue animation moves to the next Activity. Activities that have not completed execution are greyed out. This helps you see the progress made in the execution of the flow.

Attention: It is a good practice to stop testing by clicking **Stop Testing** when you finish running a flow in the Flow Tester, as the login session remains active for the entire time that you are in the testing mode.

The screenshot shows the TIBCO Flogo IDE interface. At the top, there's a navigation bar with 'Go Back to "MyApp" switch between 4 flows >' and 'test_1'. Below that, there are tabs for 'Main flow' and 'Error handler', with 'Error handler' being the active tab. The main area displays a flow diagram with a red box highlighting the 'StartSubFlow' and 'Sleep' activities. A red error icon is visible on the 'Sleep' activity. Below the flow diagram is a 'Flow logs' section with a 'Logs output' area showing a scrollable list of log messages. The logs include messages like 'Preparing the tester engine', 'Build connector done', 'Debugger running', and 'Starting TIBCO Flogo® Runtime'. A 'Copy logs' button is located to the right of the logs output area.

Handling errors

If the Activity encounters an error, it is highlighted with a red colored border and a red error icon is displayed on the the **Error handler** tab (if the error handler is run in the background). You can click the **Error handler** tab to find out till where the execution took place successfully. Note that when you navigate back to the Main flow, the red error icon is not displayed on the **Error handler** tab.



Note:

- If you start the execution from the **Error handler** tab, execution is moved on to the Main flow tab (as an error handler is always a part of the main flow).
You can, however, start a test from a tile on the **Error Handler** tab. In this case, the execution starts from the **Error Handler** tab.
- If the execution is started from a sub-flow, the execution does not move to the Main flow and acts as a normal tile execution (as a sub-flow is an independent flow).

Executing the flow from a specific Activity onwards

You can debug a specific Activity in the **Main flow** or **Error Handler** flow. If it is successful, the output is shown on the **Output** tab. If an Activity does not have any output (for example, the Return Activity), it shows the **Output** tab as blank. If the Activity is erroneous, the error is shown on the **Errors** tab.

To execute the flow from a specific Activity (and not from the beginning of the flow) with different input data, perform the following steps.



Note:

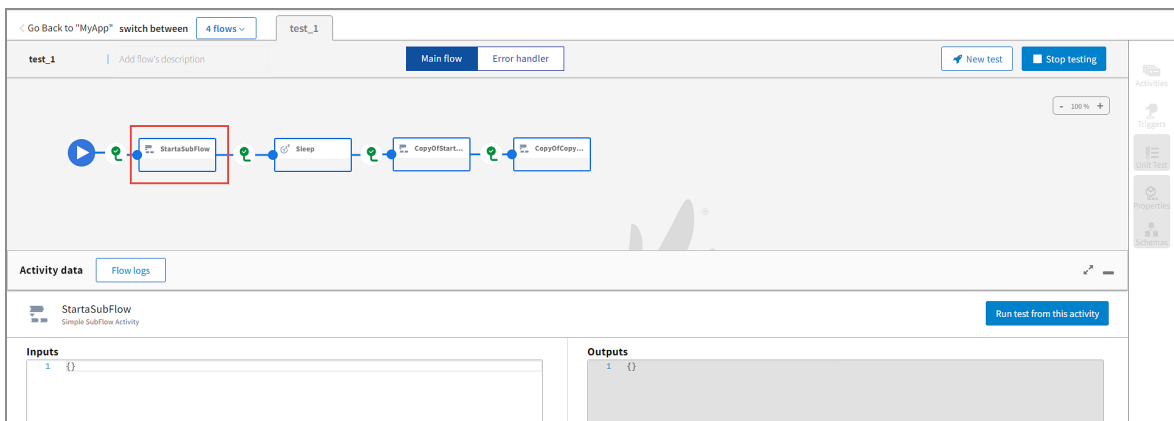
- This can be done only after the entire flow has been executed at least once.
- When you start the execution of a flow from a specific Activity in the flow, you cannot start the execution again from any Activity before the current Activity. If you need to do that, you must launch a new test.

For example, a flow includes A1 -> A2 -> A3 -> A4 -> A5 activities and execution is started from the A3 Activity onwards. In subsequent executions, you cannot start the execution from any Activity before A3; execution always starts from A3 onwards. If you want to run the flow from an Activity before A3, you must launch a new test.

Procedure

1. Select the Activity from which you want to run the flow.

The Activity is highlighted in blue. The Activity data is displayed on the **Inputs** and **Outputs** tab. If an error is returned, an **Error** tab is displayed in place of the **Outputs** tab.



2. On the **Inputs** tab, change the input values as required. You cannot do dynamic mappings here.
3. Click **Run test from this Activity**.

The execution begins from the current Activity. The logs are also displayed only for the current Activity and subsequent activities in the flow.

i Note: Once the execution starts from a tile, you cannot access preceding tasks executed in the previous runs. The previous activities are greyed out. If you want to run the flow from a previous Activity, you must launch a new test.

Logging information

As the activities are executed, the runtime engine logs for the activities are displayed in the Logs output window. The format of the logs is similar to the logs displayed while running an app binary.

To copy these logs, you can click **Copy logs**.

You can also switch from the **Flow logs** view to the Activity data view by clicking **Activity data**.

Configuring a Launch Configuration

When you click a Launch Configuration name, its mapper opens to its right. The mapper displays the input tree in the left pane.

The screenshot shows the 'Setting up Launch Configuration for flow REST' window. On the left, there is a sidebar with the Flogo logo and 'Flogo Launch Configurations'. Below the logo, there are options for 'Launch Configurations', 'Import', and 'New'. A list shows 'Launch Configuration 1' with a timestamp 'Nov 17, 2021, 2:57:54 PM'. The main area contains the following fields and sections:

- Launch Configuration name ***: A text input field containing 'Launch Configuration 1'.
- Log Level**: A dropdown menu set to 'INFO'.
- Using on-premise services**: A checkbox that is currently unchecked.
- Mapping settings**: A section with a search bar for 'Flow inputs' and a list of available inputs. One input, 'Obj flowInputs', is visible.
- Flow inputs**: A large greyed-out area with a central icon of a smartphone and a cloud, with the text 'Select an input to start mapping'.

At the bottom of the window, there are 'Cancel' and 'Next >' buttons.

Procedure

1. Select **Using on-premise services** if you want to test apps that connect to on-premise systems.

Note: Before you select this check box, enable the fLogotester service for your organization using the API. For more information, see [Enabling or Disabling the TIBCO Flogo® Flow Tester for an Organization with the API](#).

2. To configure the mapping, expand the input tree in the left pane.
3. Click an element to add a value to the element.
4. Enter the value for that element in the text box to its right.

When entering values for the elements, be aware of the following:

- The input tree for a Launch Configuration mapper displays the input you configured on the **Flow Inputs & Outputs** accordion tab for blank flows. For flows created with a trigger, it displays the output schema of the trigger.
- For flow inputs that contain only single objects, you must enter the input values at the root level. The example below shows how to enter the values for a single object, Customer:

Setting up Launch Configuration for flow flow7

Launch Configuration name *

Launch Configuration 1

Log Level

INFO

Using on-premise services

Mapping settings

Flow inputs

Flow input

obj flowinputs

obj Customer

obj Customer

```

1 {
2   "Customer": [
3     {
4       "ID": 1234,
5       "Phone": 574996,
6       "Name": "Jane Doe"
7     }
8   ]
9 }

```

< Cancel Next >

- When mapping an array of objects, the input must be provided at the array root level, which means that you must provide input for all objects in the array by clicking on the root of the array. You cannot configure the input at the array element level.

In the example below, the Customer is an array of objects. Each object within the Customer array contains ID, Phone, and Name elements. When providing values for Customer, you cannot give the input at the element (ID, Phone, or Name) level. Doing so does not specify the index of the Customer object for which you are assigning the value(s). Hence, you must assign the value to the whole Customer object. Because the Customer array has multiple objects, assign values to each object in the Customer array by separating the objects with a comma delimiter. The array size is determined based on the number of objects for which you provide values. In the example below, the array size is two because there are two objects for which values have been provided.

5. You can override app property values in the launch configuration. Properties defined in the app and those defined in a connection are listed under **properties**. Select the property whose value you want to override and specify the new app property value on the right side.

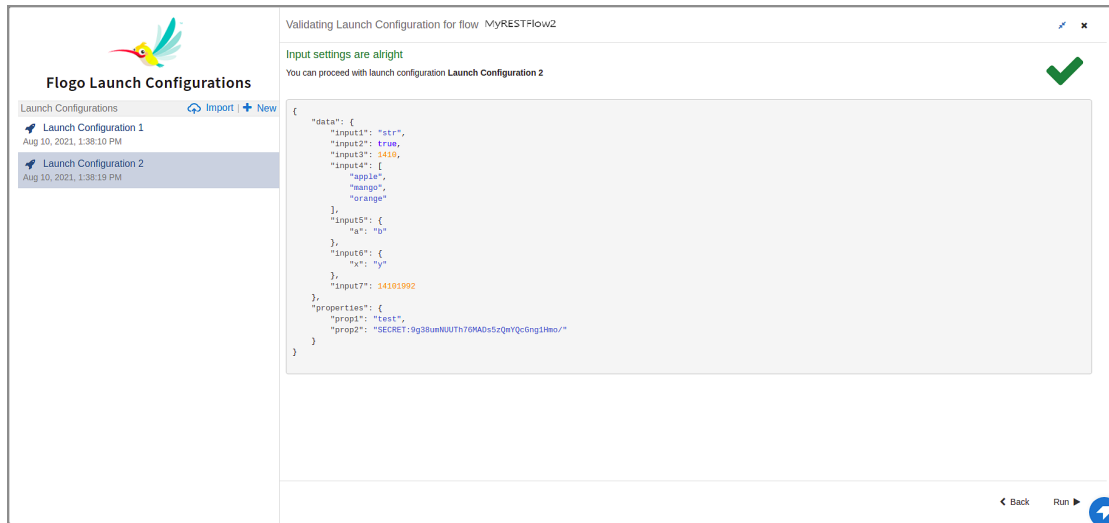
Note: For a password, you must provide an encrypted password value. For more information, see [Encrypting Password Values](#).

The screenshot shows the 'Flogo Launch Configurations' interface. The main title is 'Setting up Launch Configuration for flow FilterCakesNestedArray'. The 'Launch Configuration name' field is set to 'Launch Configuration 1'. The 'Log Level' is set to 'INFO'. The 'Using on-premise services' checkbox is unchecked. Under 'Mapping settings', the 'Flow inputs' section is expanded, showing a list of inputs: 'flowInputs' (Obj), 'properties' (Obj), 'property1' (Str), and 'property2' (No.). The 'property1' input is selected, and its value is shown as '1 "test"'. The interface includes 'Cancel' and 'Next' buttons at the bottom.

6. Click **Next**.

The mapper performs validations to ensure the validity of the JSON structure and also validates that you have entered values for all elements that are marked as required in the schema.

If there are any errors in your input, the mapper displays a list of errors. If no errors are found you get the message, **Input settings are alright**.



In your test environment, only the validation errors related to invalid JSON structure prevent you from proceeding with your testing. Errors about missing values for required elements serve as a warning but allow you to proceed with your testing. This is because it is possible that an element that is marked as a required field in the schema may not have been used in the activity at the time of testing. In that case, the element is not needed for the flow to run. But in the production environment, your app does not run successfully until you provide input values for all elements marked as required in your schema.

Exporting a Launch Configuration

There may be occasions when you want to use the same test data configurations for testing multiple flows. You have the option to create a Launch Configuration that contains this data in one flow, export the Launch Configuration, then import it into each of the other flows. The ability to export a Launch Configuration is particularly useful when the data set is very complex. In such a scenario, you can export a Launch Configuration, import it into another flow and test the flow with the imported Launch Configuration. Reusing a Launch Configuration by exporting and importing it saves you the time and effort needed to create a separate Launch Configuration for each flow.

To export a Launch Configuration:

Procedure

1. In the Flow Tester, hover your mouse cursor to the extreme right of the Launch Configuration name that you want to export.

2. Click the **Export Launch Configuration** () icon.

A file with the name `<flow-name>_<Launch Configuration-name>.json` is downloaded to your Downloads directory. You can import this file into another flow and use the Launch Configuration that you just exported. Refer to [Importing a Launch Configuration](#) for details on how to import a Launch Configuration.

i Note: The Launch Configuration name is not preserved, so the imported Launch Configuration is given a default name of "Launch Configuration x" where x stands for the next number in the series of existing Launch Configurations. For example, if you have two existing Launch Configurations in the flow, the imported Launch Configuration is named Launch Configuration 3. You have the option to edit the name to make it more meaningful.

Importing a Launch Configuration

Launch Configurations are stored as JSON files, so when you export a Launch Configuration, you export its JSON file. You import the Launch Configuration that was exported from another flow by importing the JSON file of the Launch Configuration into the flow.

i Note: The Launch Configuration name is not preserved, so the imported Launch Configuration is given a default name of "Launch Configuration x" where x stands for the next number in the series of existing Launch Configurations. For example, if you have two existing Launch Configurations in the flow, the imported Launch Configuration is named Launch Configuration 3. You have the option to edit the name to make it more meaningful.

To import a Launch Configuration, follow this procedure:

Before you begin

You must export the Launch Configuration you want to import and have its JSON file accessible before you follow the procedure below.

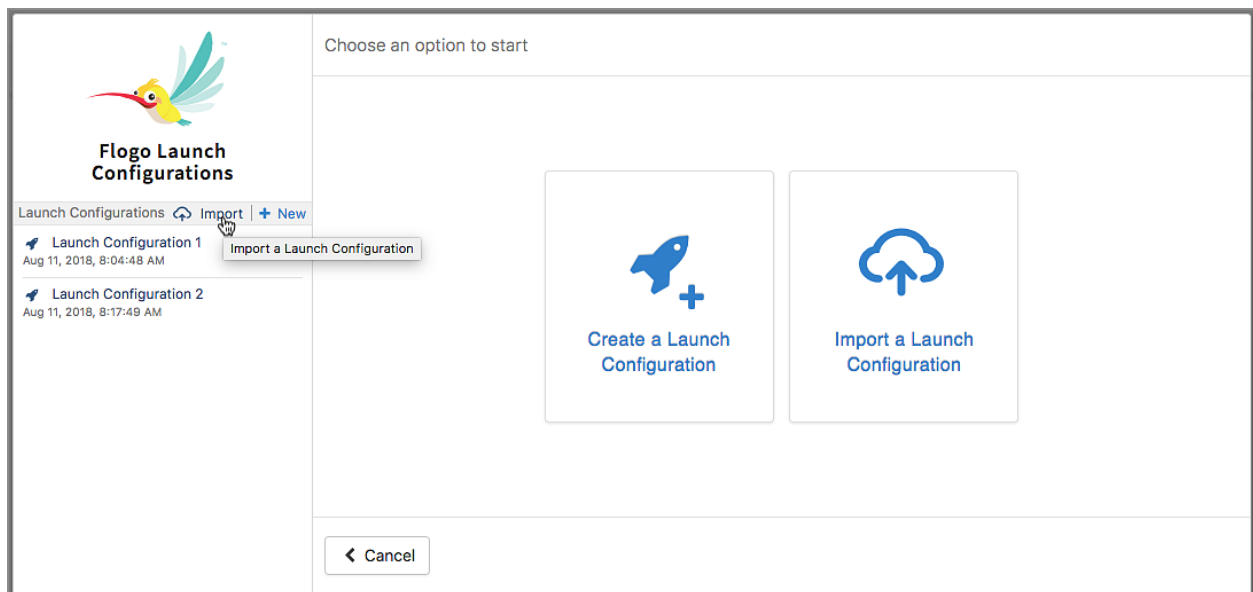
If this is the first Launch Configuration

Procedure

1. If this is the first Launch Configuration in the flow (no existing Launch Configurations), click **Test** on the flow details page.
2. Click **Import a Launch Configuration**.
3. You have the option to either drag the JSON file of the Launch Configuration you want to import, or navigate to the file using the **browse to upload** link.
4. Click **Import**. Data in the Launch Configuration being imported gets validated. In case there are any errors, they are displayed in the **Import** dialog box.

When there are existing Launch Configurations

If there are existing Launch Configurations in the flow, click **Import** in the Flow Tester and either drag the JSON file that was exported from another flow, or navigate to the file using the **browse to upload** link, then click **Import**.



Cloning a Launch Configuration


Whereas exporting and importing a Launch Configuration is useful for using the same set of data in two or more flows, cloning a Launch Configuration is useful when you want to test the same flow with two sets of data that have only minor differences.

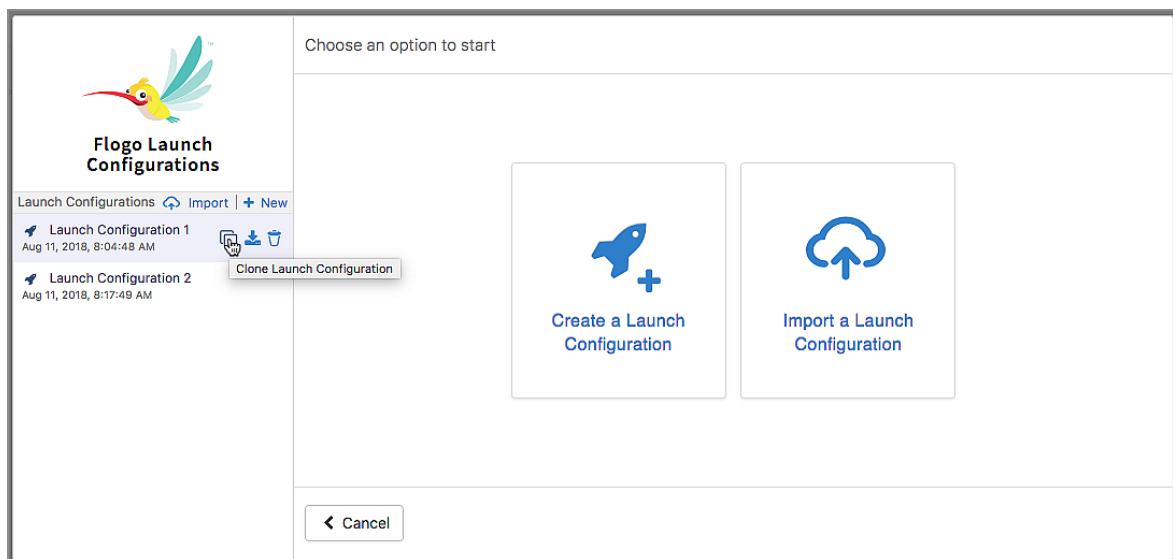
A good use case for cloning

Clone a Launch Configuration when you need to test a flow multiple times using the same input schema, but different values for one or more elements in the schema during each round of testing. You can start by creating a Launch Configuration, then cloning it, then editing the cloned Launch Configuration. You can create as many clones as needed. Each clone is a separate Launch Configuration having the same input schema. You can change the values for the elements in each cloned Launch Configuration as required. Use the original Launch Configuration for one round of testing and the cloned Launch Configuration(s) for the subsequent round(s). This saves you the effort of editing a single Launch Configuration.

To clone an existing Launch Configuration:

Procedure

1. In the Flow Tester, hover your mouse cursor to the extreme right of the Launch Configuration name that you want to clone.
2. Click the **Clone Launch Configuration** () icon. The cloned Launch Configuration is named *Copy <name-of-the-original-Launch Configuration>* by default. You can edit the name of the Launch Configuration in the **Launch Configuration name** text box.




Deleting a Launch Configuration

When you create a Launch Configuration, it automatically gets saved until you explicitly delete it.

To delete a Launch Configuration:

Procedure

1. In the Flow Tester, hover your mouse cursor to the extreme right of the Launch Configuration name that you want to delete.
2. Click the **Delete Launch Configuration** () icon.

Testing Flows from the CLI

This feature allows you to test your Flogo app using the Flogo app binary itself. Once you have built the binary for a Flogo app, you can test it using the `test` command. This feature is also useful to automate the testing process for a flow.

You can do the following from the CLI:

- List all flows in a specified app
- Generate test data for a given flow
- Test a flow against test data you specify in a JSON file
- Test a flow against test data you specify in a JSON file and generate the output of the test in an output file that you specify

Before you begin

- The app binary must be readily accessible on the machine from which you plan to test it.

Follow these steps to get help on the `test` command:

Procedure

1. Open a command prompt or terminal window depending on your platform.
2. Navigate to the folder where you stored the app binary.

3. Run the following command to get the online help on the test command:

On Windows: `<app-binary> -test`

On Macintosh: `./<app-binary> -test`

On Linux: `./<app-binary> -test`

This command outputs the usage for the test command along with some examples. Refer to [test Command](#) for details.

4. Run the command with the appropriate option to test your app. For example, if your app binary name is `MyTestApp-darwin-amd64`, to get the names of the flows in your app, run: `./MyTestApp-darwin-amd64 -test -flows`

The output of the command lists all the flows in the app.

Using the test command to test your flow from the CLI

To test a flow:

Procedure

1. Generate the input JSON file using the `-flowdata` option with the command as described (`./<app_binary> -test -flowdata <flow_name>`). This generates a JSON file (`<app-name>_<flow-name>_input.json`) with the input fields that you specified when creating the flow on the **Input** tab of the **Flow Inputs & Outputs** tab of the blank flow.

Note: You can also use the test configurations that were exported from [Launch Configuration](#) as the input file instead of generating the input file with the `-flowdata` option.

2. Modify the generated file, `<app-name>_<flow-name>_input.json`, from step 1 to set specific values for the input fields in the file.
3. Use the `<app-name>_<flow-name>_input.json` file to test your flow:

```
./<app_binary> -test -flowin <app-name>_<flow-name>_input.json -flowout
<output>.json
```

For example, if your app name is MyTestApp and the input file generated by `-flowin` is `MyTestApp_MyFlow_input.json` and the output file you specify for `-flowout` is `MyOutput.json`, the command looks as follows:

```
./MyTestApp -test -flowin MyTestApp_MyFlow_input.json -flowout
MyOutput.json
```

The test Command

Use the `test` command in Flogo Enterprise to test your Flogo app.

Options	Description and Example
<p><code>-flows</code></p> <p>SYNTAX:</p> <pre>./<binary_filename> -test -flows</pre>	<p>Lists all flows in the specified <code><binary_filename></code> app.</p> <p>Example:</p> <pre>./MyTestApp -test -flows</pre> <p>where <code>MyTestApp</code> is the app binary.</p>
<p><code>-flowdata</code></p> <p>SYNTAX:</p> <pre>./<binary_filename> -test -flowdata <flow-name></pre>	<p>Generates input fields data file for a given flow. The input test data is generated based on the Flow Inputs you provided when creating the flow (on the Input tab of Flow Inputs & Outputs).</p> <p>Example:</p> <pre>./MyTestApp -test -flowdata TestFlow</pre> <p>where <code>MyTestApp</code> is the app binary and <code>TestFlow</code> is a flow within <code>MyTestApp</code>. A JSON file with the file name with format <code><app-name>_<flow-name>_input.json</code> gets created. This file contains the generated input fields configured on the Input tab of the Flow Inputs &</p>

Options	Description and Example
<p data-bbox="207 512 310 541"><code>-flowin</code></p> <p data-bbox="207 575 310 604">SYNTAX:</p> <pre data-bbox="207 625 771 741">./<binary_filename> -test -flowin <path-to-input-file></pre>	<p data-bbox="805 296 919 325">Outputs.</p> <p data-bbox="805 359 1414 474">You can edit this file to set specific values for the input fields and use the file to test your flow using the <code>-flowin</code> option described below.</p> <hr data-bbox="805 491 1414 495"/> <p data-bbox="805 516 1386 632">Test flow against given test data contained in an input JSON file. This file must exist in the location that you specify in the command.</p> <p data-bbox="805 657 927 686">Example:</p> <pre data-bbox="805 707 1414 823">./MyTestApp -test -flowin /usr/TestFlow_input.json</pre> <p data-bbox="805 856 1386 972">where <code>MyTestApp</code> is the app binary and <code>/usr/TestFlow_input.json</code> is the path to the JSON file containing the input to the flow.</p> <div data-bbox="805 993 1414 1171" style="background-color: #f0f0f0; padding: 10px;"> <p data-bbox="824 1014 1398 1150">Note: You can also use the test configurations that were exported from Launch Configuration as the input file in this command.</p> </div>
<p data-bbox="207 1220 326 1249"><code>-flowout</code></p> <p data-bbox="207 1283 310 1312">SYNTAX:</p> <pre data-bbox="207 1333 771 1449">./<binary_filename> -test -flowin <path to test data file> -flowout <path-to-output-file-name></pre>	<p data-bbox="805 1220 1414 1451">Write flow output (if applicable) to the specified file. If a file with the specified name does not already exist in the specified location, Flogo Enterprise creates the file. If you do not specify a file name, the output gets printed to the console.</p> <p data-bbox="805 1476 927 1505">Example:</p> <pre data-bbox="805 1526 1414 1684">./MyTestApp -test -flowin TestFlow_ input.json -flowout TestFlow_ output.json</pre> <p data-bbox="805 1717 1386 1747">where <code>MyTestApp</code> is the app binary, <code>TestFlow_</code></p>

Options	Description and Example
	input.json is the file containing the input data to the flow and TestFlow_output.json is the path to the JSON file you specify to hold the output from the flow.

Unit Testing

With unit testing, you can monitor the health of your application and detect errors in individual flows or Activity levels.

While designing an application with multiple flows and activities, it becomes cumbersome to detect runtime errors at the flow and Activity levels. Using unit testing the errors at micro level are easily handled. You can run unit testing at any phase of the development cycle to verify whether activities in the process are working as expected. Using testing processes in the development stage (before you push the application to the production environment), helps detect errors and identify issues at an early stage.

Terminologies in Unit Testing

1. **Test case:** A test case is the individual unit for testing a flow. For a given set of inputs, the test case checks for a specific output for an Activity or the flow output. The expected versus actual output is compared by adding assertions to the test case. A test case can have multiple assertions added on activities and flow output. The test case is considered as passed when all the assertions in that test case pass.
2. **Assertion:** An assertion is a logical expression that evaluates to a boolean value. The expected versus actual output is compared by using an assertion expression. For the passed assertion, the expression evaluates to true. A non-boolean expression always evaluates to false.
3. **Flow Input:** Flow input is a set of input data used to run the test on the given flow. Each test case has its own set of inputs.
4. **Flow Output:** Flow output is the output generated for the given flow for the given set of inputs. Flow output can have one or more assertions.
5. **Test Suite:** A group of test cases make a test suite. An app can have multiple test

suites. The test suite is considered as passed when all the tests in the test suite are successfully run.

6. **Test Suite file:** A test suite file is an exported file that contains all the test suites for a given Flogo app. The file has .flogotest as its extension and can be exported from the studio as well as the platform API. The test suite file along with the Flogo app binary can be used to run the test cases in the stand-alone environment outside TCI.
7. **Test Result File:** A test result file with extension .testresult has the detailed execution result of the test suites, the test cases under that test suite and the assertion execution result for each test case. The test result file is generated after running the tests on the exported binary.


Role Requirements

- Admins and Users have full access to unit testing for the apps that they own.
- Admins cannot import unit test data for apps that they do not own.
- Users and read-only users have no access to unit testing for the apps that they do not own.
- Any user role cannot access unit testing if the apps are created using the platform API.

Creating and Running a Test Case

Unit testing in flogo tests smaller chunks of work in the process.

A test case is a basic building block of unit testing. A test case can have one or more assertions on the activities in the main flow, activities in the error handler, or on the flow output. One flow can have multiple test cases.

To enable the unit testing mode, click the **Unit Test** icon () that is available on the **Activities, Triggers, Unit test, Properties and Schema** panel on the right. Click the same icon to exit the unit testing mode.

Configuring Unit Test Data

To design a unit test case, you need to configure the flow inputs, activities, and flow outputs with the appropriate data. When you click any of these components, a

configuration dialog opens where you can configure the data. You can configure activities using any of the modes listed in [Unit Testing Modes](#).

Unit Testing Modes

The following tables provide a detailed explanation of the various testing modes available for configuring the unit test cases.

For Activities with output (example mapper and RESTInvoke):

Option	Description
Execute (Default)	<p>This is the default mode for all activities.</p> <p>When an activity is set to this mode, it runs as per the definition and configuration and does not have any effect in the unit test execution. You can use this option to reset your unit test configurations on the activity.</p>
Assert on Outputs	<p>Adds an assertion for flow output.</p> <p>For more information, see Creating Assertions for Flow Output.</p>
Assert on Error	<p>Adds an assertion for the flow designed in the error handler. Unit test case designed for the error handler flow is run to detect any run time exceptions or errors in the flow implementation.</p> <p>For more information, see Creating Assertions for the Error Handler.</p>
Mock Outputs	<p>Use mock data for the activities that have an output.</p> <p>For more information, see Using Mock Data.</p>
Mock Error	<p>Use mock exceptions for an activity to find out whether the exception handling is being done correctly.</p>

For activities without output (example LogMessage):

Option	Description
Execute (Default)	<p>This is the default mode for all activities.</p> <p>When an activity is set to this mode, it runs as per the definition and configuration and does not have any effect in the unit test execution. You can use this option to reset your unit test configurations on the activity.</p>
Assert on Error	<p>Adds an assertion for the flow designed in the error handler. Unit test case designed for the error handler flow is run to detect any run time exceptions or errors in the flow implementation.</p> <p>For more information, see Creating Assertions for the Error Handler.</p>
Mock Error	<p>Use mock exceptions for an activity to find out whether the exception handling is being done correctly.</p>
Skip Execution	<p>Skip an activity in unit testing if the activity does not have any output. For example, you can skip activities such as Sleep, eFTL Publish message, or StartaSubFlow.</p> <div data-bbox="776 1310 1414 1486" style="background-color: #f0f0f0; padding: 10px;"> <p>Note: To skip an activity with an output, you can mock it without any configuration data. For activities that do not have an output, you can select Skip Execution.</p> </div>

Creating a Test Case

Before you start unit testing, you must create a test case. Perform the following procedure to create a test case:

Procedure

1. On the **Flows** page, click the flow that you want to run a unit test on. The flow design page opens.
2. On the flow design page, click **Unit Test**.
3. On a pop-up layover, click **Unit Tests** to create a test case.
4. Provide **Test Name** and **Description** for your test case. Click **Create**.

You can create more test cases by clicking  beside **Test Cases**.

! **Important:** The unit test results are captured only when the flow execution successfully completes. If your test case is configured to create an exception while running any of the activities in the flow, then configure at least one of the following in your flow:

1. **Exception Handling:** Ensure that your flow has proper exception handling. This can be achieved either by configuring an error branch from the activity that can potentially raise an exception or by defining an error handler for the overall flow.
2. **Assertions on Error:** Add one or more assertions using the **Assert on Error** mode in the flow outputs. For more information, see [Creating Assertions for the Error Handler](#).

Defining Flow Input

For a particular activity that has a flow input configured in the actual process, you must assign the flow input parameters before you run a test case. You can add separate test cases for each flow input.

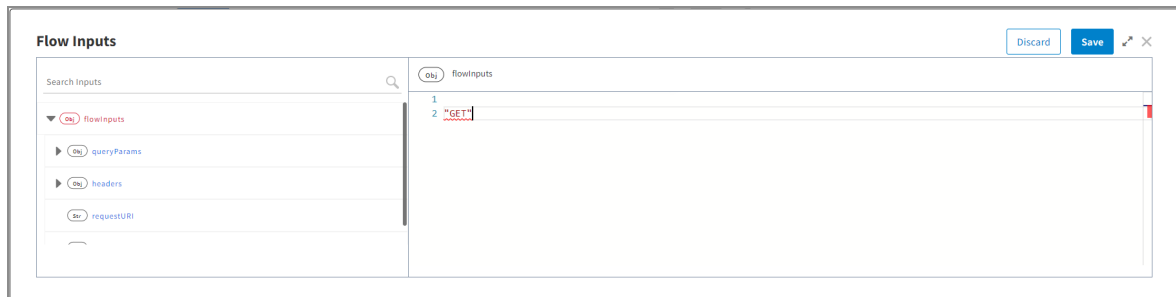
i Note:

- If the flow input is configured for the activity, then you must define the flow input value when running a test case, otherwise the test case fails. However, if the flow input is not configured for the activity, then you need not to define the flow input value when running a test case.
- For inputs containing single objects, you must enter the input values at the root level.
- For mapping an array of objects, you must provide inputs at the array root level. Click the root of the array to input values for all objects. You cannot configure the input at the array element level.

To define the flow input parameter, follow these steps:

Procedure

1. On the **Unit Test** page, under the **Main Flow** tab, click **Flow Input**.
2. Provide the required flow input details and click **Save**.

**Creating Assertions**

To compare the actual vs expected output, you can add multiple assertions on an activity, flow output, or error handler. The assertion expression always evaluates to a boolean value.

- i Note:** You cannot create assertions for the activities that do not return output. Similarly, you cannot create assertions for the flow output when it has no outputs.

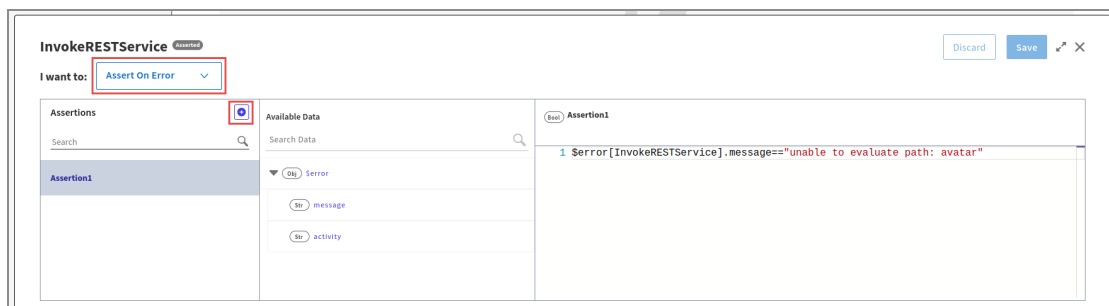
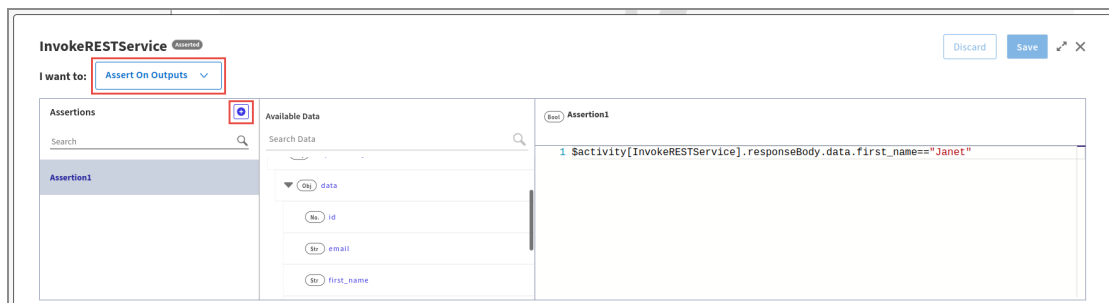
To add unit test assertions for a test case:

Procedure


1. On the **Unit Test** page, select one of the flows below to which you want to add an assertion:
 - Error Handler: On the **Error Handler** tab, click **<Activity Name>**.
 - Main Flow: On the **Main Flow** tab, click **<Activity Name>** or **Flow Output**.

The **Unit Test Data Configuration** dialog opens.

2. From the **I want to** dropdown, select one of the following assertions:
 - **Assert on Outputs**
 - **Assert on Error**



Note: The assertion types displayed in the dropdown are based on the configuration of the selected activity.

3. Click **New Assertion** or  icon to create an assertion. An assertion with the default name is created.
4. Map the data from the **Available Data** section to the appropriate values and click **Save**.

i Note:

- While asserting an object, it is recommended to assert each property individually rather than asserting the entire object.
- You cannot save changes if you delete all assertions. To remove all existing assertions, select a different mode from the **I want to** dropdown.

Creating Assertions for Flow Output

i Note: You cannot create assertions for the flow output when it has no outputs.

You can add assertions to the flow output to verify that the flow produces the correct data by comparing the actual output against predefined assertions.

To create assertions for the flow output, see [Creating Assertions](#).

Creating Assertions for the Error Handler

You can also add test cases for the flow designed in the error handler. Unit test case designed for the error handler flow is run to detect any run time exceptions or errors in the flow implementation.

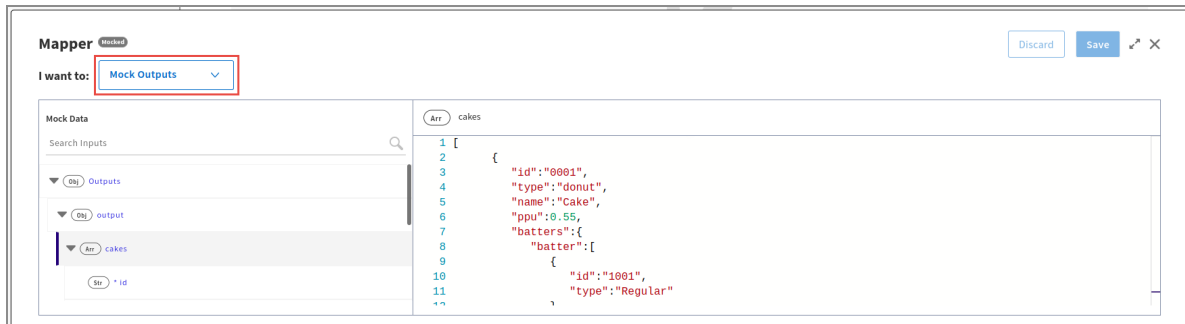
To create assertions for the error handler, see [Creating Assertions](#).

Using Mock Data

In unit testing, you can mock the data for the unit that is being tested. This is useful during unit testing so that the external dependencies are no longer a constraint to the unit under test. Using mock data the dependencies are replaced by closely controlled replacements that simulate the behavior of the real ones.

You can use the mock data for the activities that have an output. Expressions and functions are not evaluated in the values given to mock outputs, the input provided is passed as-is.

In unit testing, you can either use assertions or mock data to test the activities.

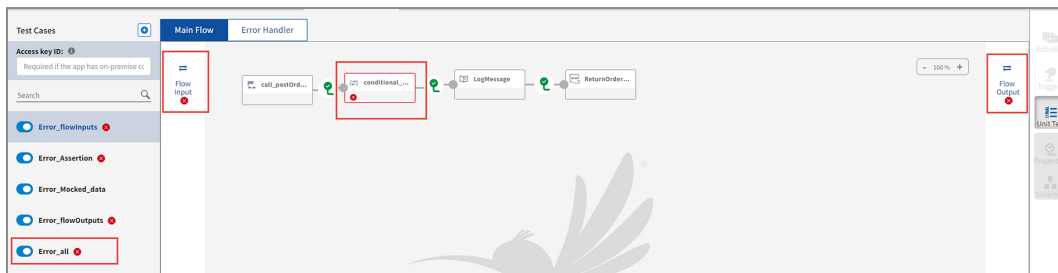


Test Case Validation

Before you run a test case, Flogo auto-validates the test cases when you open the unit testing mode.

To validate a test case, switch to unit testing mode to see if there are any validation errors. An error sign ✖ is displayed on the:


- assertion level of the activity or mock data
- flow input and flow output
- test case level of activity that has errors

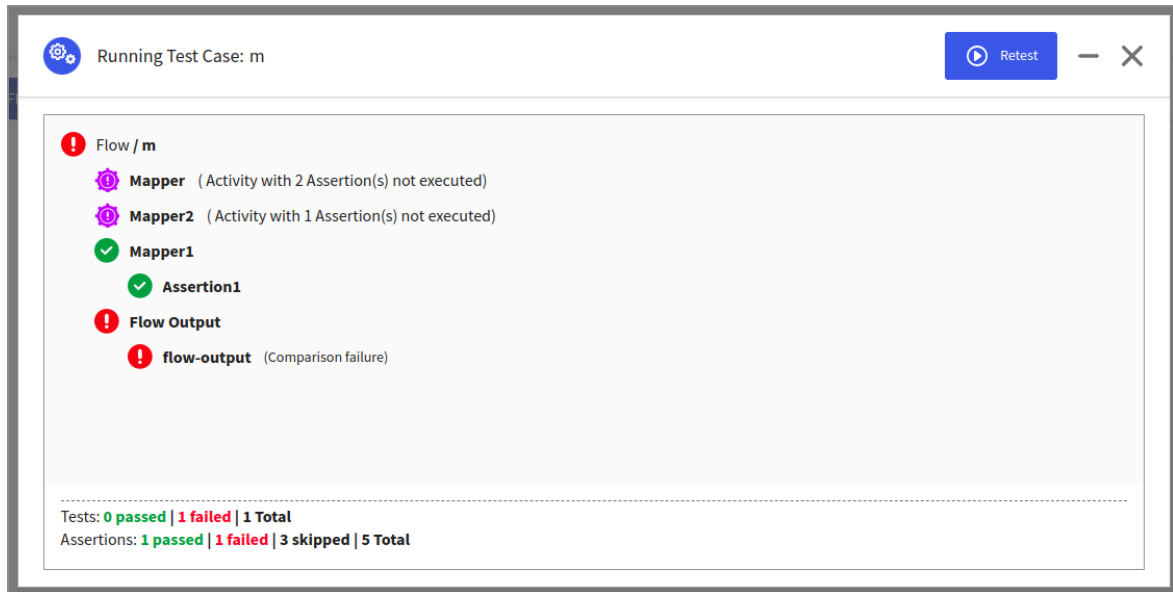


Running a Test case

After you are done with creating a test case, the test case is ready to run.




Procedure

1. To run a test case, click the **Play** icon  next to the respective test case.
2. After the test case run is completed, the result is generated.



The result window displays the total number of test cases, which include the number of passed and failed test cases. It also displays the total number of assertions and the number of passed and failed assertions on activities in the flow, activities in the error handler, and in the flow output. The result for assertions on the flow output is displayed only if the assertion is added to the flow output.

The icons on the result window are described as follows:

Icon	Description
	The assertion on the particular field has passed.
	The assertion on the particular field has failed.
	The assertion on the particular field is skipped.

**Tip:**

- You need not close the result window to modify your test case. You need to **Minimize** the window, make the changes, and **Retest** the case.
- You can enable the test cases to include in the **Run all Test Cases**. Disable the test cases to exclude it from **Run all Test Cases**.
- You can edit, delete, or copy a test case or a test assertion at any point.
- For an active unit test case, if you change app level schemas or app properties, close the session and rerun the test case.

**Note:**

- When running a binary, the test suites or test cases are run by default irrespective of whether they are enabled or disabled.
- When you export a `.flogotest` file, only the instance that you save (either assertions or mock data) is reflected in the file.

Creating and Running a Test Suite

You can use the **Test Suite** feature to combine different test cases and run them at once.

Creating a Test Suite

To create a test suite:


Procedure

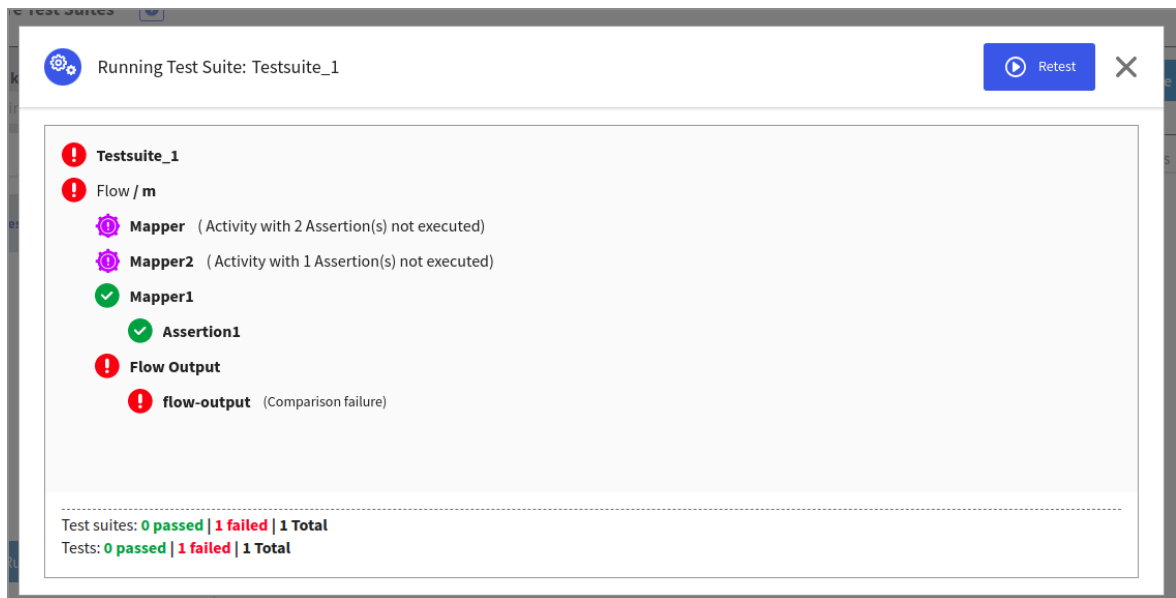
1. On the **Flows** page, click **Test Suite**. The test suite dialog opens.
2. In the **Test Suite** dialog, click **New Test Suite**. A test suite with a default name gets created.
3. You can add a test case by clicking **Add test cases** on the **Test Suite** pop-up modal.
4. Select the test cases to be added in the suite. Click **Save** and close the dialog.

Running a Test Suite

After you are done with creating a test suite, the test suite is ready to run.



Procedure

1. To run a test suite, click the **Play** icon  next to the test suite.
2. After the test suite run is completed, the result is generated.



A result window displays the total number of test suites and test cases with the number of passed and failed test suites and test cases.

The icons on the result window are described as follows:

Icon	Description
	The assertion on the particular field has passed.
	The assertion on the particular field has failed.
	The assertion on the particular field is skipped.

i Note:

- You can enable the test suites to include in the **Run all Test Suites**. Disable the test suite to exclude it from **Run all Test Suites**.
- When engine is running in unit test mode, it does not fail fast and continues on connection errors. The connections have a retry mechanism, then the start up time considerably increases. If any activity that uses connections is not mocked, the test case throws an error.

Exporting and Importing a Unit Test


After you complete designing the test cases and test suites, you can export the unit test file and import it anytime to get the unit test data in a Flogo app.

The exported file has the app version attached to it. You can maintain the unit test files based on your app versions so that you import the correct version of the unit test file matching your app versions to avoid any configuration issues.

Exporting a Unit Test

Perform the following procedure to export a unit test:

Procedure

1. On the **Apps** page, under **Flow**, click the shortcut menu .
2. Go to **Export > Unit Tests**.
A `.FLOGOTEST` file is downloaded to your system.

Importing a Unit Test

To import a unit test:

Procedure

1. On the **Apps** page, under **Flows**, click the shortcut menu .

2. Go to **Import > Unit Tests**.
3. Drag or upload the required `.FLOGOTEST` file and click **Upload**.

i Note: The unit test file is dependent on the names of the flow and activities. While importing, if the flow name or activity name in the flow does not match, the test cases of that flow and assertions on that activity gets skipped.

Enabling On-premises Services in Unit Testing

You can enable on-premises services in unit testing to run the test cases or test suites on the applications that are connecting to the on-premises services.

To enable services, while running a test Case or a test suite, first, you must provide the access Key in the **Access Key ID** text box on the **Unit Test** page. The access key is stored per application per browser session and gets auto-filled in unit test case and unit test suite once filled.

- i Note:**
- If there is no access Key provided, it is considered that there are no activities connecting to the on-premises services.
 - If you refresh the browser window, then the access Key is cleared from the text box and you need to reenter the key.

To configure the hybrid agent and generate the access key, see [Configuring the hybrid agent](#).

Unit Testing for the CI/CD

i Note: The information in this section is applicable for an app executable only.

This feature allows you to unit test your Flogo app using the app executable. Once you have built the executable for a Flogo app, you can run a unit test using the `test` command. This feature is also useful to automate the testing process for activities in development in CI/CD pipeline.

You can even generate the files using the platform API by following the below steps:

Procedure

1. Export the app JSON. For details, see [exporting an app with the API](#).
2. Build an app executable. For details, see [building an app executable](#).
3. Download the built app executable. For details, see [downloading the app executable](#).
4. Export the test suite file. For details, see [exporting a test suite file with API](#).

After you have downloaded the app executable file and the test suite file, you can build a CI/CD pipeline to run the unit test using the below mentioned commands.

Before you begin

The app executable must be readily accessible on the machine from which you plan to test it.

Follow these steps to get help on the test command:

Procedure

1. Open a command prompt or terminal window depending on your platform.
2. Navigate to the folder where you stored the app executable.
3. Run the following command to get the online help on the test command:
 - On Windows: `<app-executable> --test --test-file`
 - On Macintosh: `./<app-executable> -test --test-file`
 - On Linux: `./<app-executable> -test --test-file`

This command outputs the usage for the test command along with some examples. see [the test commands](#) for details.

4. Run the command with the appropriate option to test your app.

The output of the command generates the `.testresult` file for the unit test suites or test cases that are run.

The Test Commands

Use the test command in Flogo to run unit test on your Flogo app.

Syntax: `./<app-executable> --test --app <path to application json> --test-file <path to flogotest file> --test-suites <testsuite1, testsuite2> --output-dir <path to store .testresult file> --result-filename <custom name of .testresult file>`

Options	Description and example
<code>--test</code>	Runs a unit test on the specified <code><executable_filename></code> app.
<code>--app</code> (Optional)	Application JSON path on which the unit testing is to be done. If application JSON is not provided it takes the embedded app in the binary.
<code>--test-file</code>	The <code>.flogotest</code> file that the user has obtained by exporting a unit test file.
<code>--test-suites</code> (optional)	Test suite names that are to be run in unit testing. If test suites are not provided, tests are executed for all test suites in the test file. Example: <code>--test-suites testsuite1, testsuite2</code>
<code>--output-dir</code> (optional)	Output directory in which the test results are stored. If the output directory is not provided it stores the test result in the working directory.
<code>--result-filename</code> (optional)	Name of the unit testing output file. If the test result file name is not provided it stores as <code><AppName>.testresult</code> .

Deployment and Configuration

After you have created and validated your app, you can build an app executable to deploy and run it.

Building an App Executable

This section instructs you on how to build an app executable.

Building the App

After you have created your app, you can build it anytime. When you build the app, its deployable artifact gets created. You can download it to your local machine. Each operating system has its build target. Select the right target for your operating system when building the app. You can use the built artifact to run the app.

You can also use the TIBCO Cloud Integration API to build the app executable. For more information on the APIs, see [TIBCO Cloud™ Integration API](#).




Note: Building an app executable in TIBCO® Cloud Integration always builds the app executable with the latest version of Flogo.

Before you begin

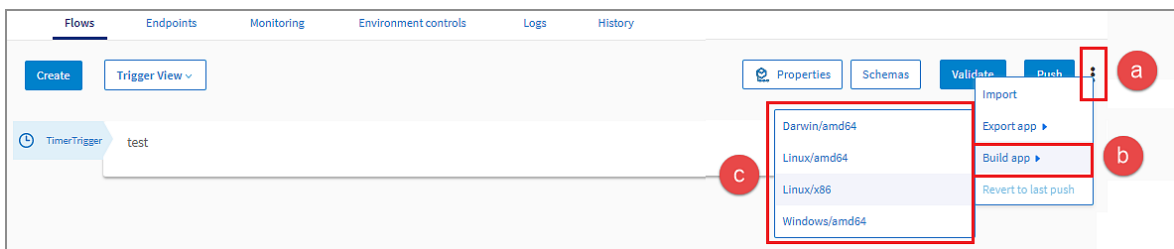
Make sure you have the following:

- The app for which an app executable needs to be created must have a trigger and a flow in it. If the app does not have a trigger and flow, the app executable is not created.
- Read through the [Considerations](#).

Procedure

1. Open the **Apps** page.
2. Click the app for which you want to build an app executable. The page for the selected app opens.
3. On the page that opens, click **Validate** and resolve errors, if any.
4. Open the shortcut  menu, click **Build app**, and select a build target option that is compatible with your operating system (such as Darwin/amd64 for the Macintosh).

Note: If you have created or pushed an app using the tibcli or platform API, the **Build App** option is not displayed as the apps are read-only apps.



The following build target options are available:

- Macintosh: **Darwin/amd64**
- 64-bit Linux: **Linux/amd64**
- 32-bit Linux: **Linux/x86**
- Microsoft Windows: **Windows/amd64**

The app begins to build. When it is built, the deployable artifact is downloaded to your local machine.

5. To confirm whether the app executable is built successfully, go to the **History** tab and check whether **Action** is displayed as **APP BUILD**.

Running the App

On the Macintosh and Linux

Procedure

1. Open a terminal.

2. Run:

```
chmod +x <app-file-name>
```

3. Run:

```
./<app-file-name>
```

On Microsoft Windows

At the command prompt, run:

```
<app-file-name>.exe
```

Considerations

- You cannot build an app executable if your app uses:
 - TIBCO Cloud™ Mesh.
If your app includes an InvokeRESTService activity that is configured to use services from TIBCO Cloud™ Mesh (by setting the **Discover services from TIBCO Cloud Mesh** option to **True**), you cannot build an app executable.
 - TIBCO Cloud™ Live Apps connectors.
 - TIBCO Cloud™ AuditSafe connectors.
 - Custom Golang code.
- SSL/TLS configuration is not enabled for inbound triggers such as GraphQL, ReceiveHTTPMessage, and Websocket triggers. If you configure SSL for these triggers in TIBCO Flogo® Enterprise and then import the app in TIBCO Cloud™ Integration, the SSL configuration is not displayed in TIBCO Cloud™ Integration.
- For the **Amazon S3 Get** and **Amazon S3 Put** activities of TIBCO Flogo® Connector for Amazon S3, **Input Type/Output Type** of **File** is not supported for a service or operation object.
If you create the app in TIBCO Flogo® Enterprise and then import the app in TIBCO Cloud™ Integration, the **File** option is not displayed in:
 - **Input Type** of the **Upload** setting of **Put** activity.

- **Output Type** of single object operation setting of **Get** activity.
- You cannot build a docker image of an app using TIBCO Cloud™ Integration - Flogo®. Instead, download the Linux app executable and then build the docker image.

Exporting App JSON from an Executable

To export the app JSON from an app executable, use the following command:

```
<app_exec> export -o <app-name>.flogo app
```

Where:

- *<app_exec>* refers to the executable of your Flogo application.
- *<app-name>* is the desired name of the exported file.

This command extracts the application's configuration and workflow details, and saves it as a JSON file.

Overriding an App's JSON File in the App Binary

While running the app binary, you can override the app binary's embedded JSON file with another JSON file by using the `-app` option. This saves you the time and effort of creating an app binary if you only want to make minor configuration or mapping changes in the app.

To do this, run the following command:

- On Windows: `<app-filename>-windows_amd64 -app <new JSON file.json>`
- On Macintosh: `<app-filename>-darwin_amd64 -app <new JSON file.json>`
- On Linux: `<app-filename>-linux_amd64 -app <new JSON file.json>`

i Note: You can modify your activities, export the `app.json` again, and run it with the same binary using the `-app` option. For example, you can make changes to an existing activity. However, if you add an activity (of the same category or a different category) and try to run it from the app binary, it does not work.

Changing the Log Level of a Running App Instance

When starting an app, you can set the log level for the app by using the `FLOGO_LOG_LEVEL` environment variable. For more information, see [FLOGO_LOG_LEVEL](#).

After the app starts, to change the log level of the running app instance without restarting the app, you can use the GET and PUT curl commands. You can perform the operations on the console or use an application such as Postman.

The Content-Type is always `application/json`, even if you have specified another Content-Type.

When the log level is changed, a message is displayed on the console.

Example

1. Start the app as follows:

```
FLOGO_HTTP_SERVICE_PORT=<port> FLOGO_LOG_LEVEL=<log_level> ./<app_name>
```

2. To get and display the log level on the console, use the GET curl command as follows:

```
curl -i -X GET http://localhost:7777/app/logger
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 18 Aug 2021 00:17:57 GMT
Content-Length: 17
{"level":"INFO"}
```

3. To change the log level, use the PUT curl command as follows:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{"level":"DEBUG"}' http://localhost:7777/app/logger
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 18 Aug 2021 00:19:05 GMT
```

```
Content-Length: 35  
{"msg":"Log level set to 'DEBUG'"}  
}
```

App Configuration Management

Flogo allows you to externalize app configuration using app properties so that you can run the same app binary in different environments without modifying your app. It integrates with configuration management systems such as Consul and AWS Systems Manager Parameter Store to get the values of app properties at runtime.

You can switch between configuration management systems without modifying your app. You can do this by running the command to set the configuration-management-system-specific environment variable from the command line. Since the environment variables are set for the specific configuration management system, at runtime, the app connects to that specific configuration management system to pull the values for the app properties.

Consul

The Consul provides a key/value store for managing app configuration externally. Flogo Enterprise allows you to fetch values for app properties from Consul and override them at runtime.

i Note: This section assumes that you have set up Consul and know-how Consul is used to storing service configuration. Refer to the Consul documentation for consul-specific information.

A Flogo app connects to the Consul server as its client by setting the environment variable, `FLOG_APPS_PROPS_CONSUL`. At runtime, you must provide the Consul server endpoint for your app to connect to a Consul server. You have the option to enter the values for the Consul connection parameters. You can either type in their values as JSON strings or create a file that contains the values and use the file as input.

Consul can be started with or without `acl_token`. If using an ACL token, make sure to have the ACL configured in Consul.

Using Consul

Below is a high-level workflow for using Consul with your Flogo app.

Before you begin


You must have access to Consul.

Set up Consul and understand how Consul is used to storing service configuration. For information on Consul, refer to the Consul documentation.

To use Consul to override app properties in your app (properties that were set in Flogo Enterprise):

Procedure

1. Export your app binary from Flogo Enterprise. Refer to [Exporting and Importing an App](#) for details on how to export the app.
2. Configure key/value pairs in Consul for the app properties whose values that you want to override. At runtime, the app fetches these values from the Consul and uses them to replace the default values that were set in the app.

3.  **Important:** When setting up the Key in Consul, make sure that the Key name matches exactly with the corresponding app property name in the **Application Properties** dialog in Flogo Enterprise. If the property name does not match exactly, a warning message is displayed, and the app uses the default value for the property that you configured in Flogo Enterprise.

4. Set the FLOGO_APP_PROPS_CONSUL environment variable to set the Consul server connection parameters. See [Setting the Consul Connection Parameters](#) for details.

Consul Connection Parameters

Provide the following configuration information during runtime to connect to the Consul server.

Property Name	Required	Description
server_address	Yes	Address of the Consul server, which could be run locally or elsewhere in the cloud.
key_prefix	No	<p>Prefix to be prepended to the lookup key. This is essentially the hierarchy that your app follows to get to the Key location in the Consul. This is helpful in case the key hierarchy is not fixed and may change based on the environment during runtime. It is also helpful in case that you want to switch to a different configuration service such as the AWS param store. Although it is a good idea to include the app name in the key_prefix, it is not required. key_prefix can be any hierarchy that is meaningful to you.</p> <p>As an example of a key_prefix, if you have an app property (for example, Message) that has two different values depending on the environment from which it is being accessed (for example, dev or test environment), your <key_prefix> for the two values can be /dev/<APPNAME>/ and /test/<APPNAME>/. At run time, the right value for Message is picked up depending on which <key_prefix> you specify in the FLOGO_APP_PROPS_CONSUL environment variable. Hence, setting a <key_prefix> allows you to change the values of the app properties at runtime without modifying your app.</p>
acl_token	No	<p>Use this parameter if you have key access protected by ACL. Tokens specify which keys can be accessed from the Consul. You create the token on the ACL tab in Consul.</p> <p>During runtime, if you use the acl_token parameter, Key access to your app is based on the token you specify.</p> <p>To protect the token, encrypt the token for the key_prefix where your Key resides and provides the encrypted value of that token by prefixing the acl_token parameter with SECRET. For example, "acl_token": "SECRET:QZL0rtN3g0EpXgUuud6jprgo/WzLR7j+Twv28/4KcP7573snZWo+hGuQauuR2o/7TJ+ZLQ==". Note that the encrypted value follows the key_prefix format.</p> <p>Provide the encrypted value of the token as the SECRET. SECRETS get decrypted at runtime. To encrypt the token, you obtain the token from</p>

Property Name	Required	Description
		<p>the Consul. Then, encrypt it using the app binary by running the following command from the directory in which your app binary is located:</p> <pre>./<app_binary> --encryptsecret <token_copied_from_Consul></pre> <p>The command outputs the encrypted token that you can use as the SECRET.</p> <p>Note: Since special characters (such as `! < > & `) are shell command directives, if they appear in the token string when encrypting the token, you must use a backslash (\) to escape such characters.</p>
insecure_connection	No	<p>Set to True if you want to connect to a secure Consul server without specifying client certificates. This should only be used in test environments.</p> <p>Default: False</p>

Setting the Consul Connection Parameters

You set the values for app properties that you want to override by creating a Key/Value pair for each property in Consul. You can create a standalone property or a hierarchy that groups multiple related properties.

Before you begin

This document assumes that you have access to Consul and are familiar with its use. To create a standalone property (without hierarchy), you simply enter the property name as the name of the Key when creating the Key in Consul. When you create a property within a hierarchy, enter the hierarchy in the following format in the **Create Key** field in Consul: <key_prefix>/<key_name> where <key_prefix> is a meaningful string or hierarchy that serves as a path to the key in Consul and <key_name> is the name of the app property whose value you want to override.

For example, in dev/Timer/Message and test/Timer/Message, dev/Timer and test/Timer are the <key_prefix> which could stand for the dev and test environments and Message is

the key name. During runtime, you provide the `<key_prefix>` value that tells your app the location in Consul from where to access the property values.

Warning: The Key name in Consul must be identical to its counterpart in the **Application Properties** dialog in Flogo Enterprise. If the key name does not match exactly, a warning message is displayed, and the app uses the default value that you configured for the property in Flogo Enterprise.

Warning: A single app property, for example, Message, is looked up by your app as either Message or `<key_prefix>/Message` in Consul. An app property within a hierarchy such as `x.y.z` is looked up as `x/y/z` or `<key_prefix>/x/y/z` in Consul. Note that the dot in the hierarchy is represented by a forward slash (/) in Consul.

After you have configured the app properties in Consul, you need to set the environment variable, `FLOGO_APP_PROPS_CONSUL`, with the Consul connection parameters for your app to connect to the Consul. When you set the environment variable, it triggers the app to run, which connects to the Consul using the Consul connection parameters you provided and pulls the app property values from the `key_prefix` location you set by matching the app property name with the `key_name`. Hence, the Key names must be identical to the app property names defined in the **Application Properties** dialog in Flogo Enterprise.

You can set the `FLOGO_APP_PROPS_CONSUL` environment variable either by directly entering the values as a JSON string on the command line or placing the properties in a file and using the file as input to the `FLOGO_APP_PROPS_CONSUL` environment variable.

Entering the Consul Parameter Values as a JSON String

To enter the Consul parameters as a JSON string, enter the parameters as key/value pairs using the comma delimiter. The following examples illustrate how to set the values as JSON strings. You would run the following from the location where your app resides:

An example when not using security without tokens enabled:

```
FLOGO_APP_PROPS_CONSUL="{\"server_address\": \"http://127.0.0.1:8500\"}" ./Timer-darwin-amd64
```

Where `Timer-darwin-amd64` is the name of the app binary.

An example when tokens are enabled and app properties are within a hierarchy:

```
FLOGO_APP_PROPS_CONSUL="{\"server_address\":\"http://127.0.0.1:8500\",\"key_prefix\":\"/dev/Timer\",\"acl_token\":\"SECRET:b0UaK3bTyD9wN+ZJkm\lKRmojhAv+\"}"
```

Where `/dev/Timer` is the path and `SECRET` is the encrypted value of the token obtained from the Consul.

This command directs your app to connect to the Consul at the `server_address` and pull the values for the properties from the Consul and run your app with those values.

Refer to the [Consul Connection Parameters](#) section for a description of the parameters. Refer to [Encrypting Password Values](#) for details on how to encrypt a value.

Setting the Consul Parameter Values Using a File

To set the parameter values in a file, create a `.json` file, for example, `consul_config.json` containing the parameter values in key/value pairs. Here is an example:

```
{
  "server_address": "http://127.0.0.1:32819",
  "key_prefix": "/dev/<APPNAME>/",
  "acl_token": "SECRET:b0UaK3bTyD9wN+ZJkm\lKRmojhAv+"
}
```

Place the `consul_config.json` file in the same directory that contains your app binary.

Run the following from the location where your app binary resides to set the `FLOGO_APP_PROPS_CONSUL` environment variable. For example, to use the `consul_config.json` file from the example above, run:

```
FLOGO_APP_PROPS_CONSUL=consul_config.json ./<app_binary_name>
```

The command extracts the Consul server connection parameters from the file and connects to the Consul to pull the app properties values from the Consul and runs your app with those values.

Consul properties can also be run using docker by passing the same arguments for the docker image of a binary app. For more information, see [Building the App](#).

Overriding an App Property at Runtime

While using config management services like Consul or AWS Params store, you can update or override an app property at runtime without restarting or redeploying the app.

Note: Currently, this functionality is only available for app properties mapped in activities. It is not available for app properties in triggers and connections.

Before you begin

Set the following environment variables:

Environment Variable	Description
FLOGO_APP_PROP_RECONFIGURE=true	Specifies that app properties can be updated or overridden at runtime.
FLOGO_APP_PROP_SNAPSHOTS=true	Used along with FLOGO_APP_PROP_RECONFIGURE. If you do not want your application to pick the updated app properties dynamically for a running flow, set this variable to true. The updated values are effective only for new flows and not existing ones.
FLOGO_HTTP_SERVICE_PORT=<port number>	Specifies the service port. For apps running in TCI, you do not need to specify the port. The default is 7777.
FLOGO_APP_PROPS_CONSUL="{\"server_address\":\"http://127.0.0.1:8500\"}"	Specifies the Consul server address.

Overriding Values by Specifying New Values

Procedure

1. In the Flogo app, create an app property and map it to the activities as required.

2. Create the same key as the app property and add some value.
3. Run the app with the environment variables in the "Before you begin" section. The app takes all the configured values.
4. Update the values.
5. To reconfigure the app property values, use the API as follows:

```
curl -X PUT localhost:7777/app/config/refresh
```

A successful response is returned from the API.

6. Open the app property update logs to verify that the new app property values are used by the activities.

Overriding Values by Specifying New Values in the API Directly

You can specify the new values of app properties directly through the body of the reconfigure API. This method takes priority over any other resolver specified.

Example:

```
curl -X PUT -H "Content-Type: application/json" -d '{"Property_1":"Value"}' localhost:7777/app/config/refresh
```

Important Considerations

- If the same property exists in Consul, the property from the body of the reconfigure API is used.
- Any new request on the API does not save property values provided on a previous request.
- Properties mentioned in an earlier request and not mentioned in the new request take values if present from other resolvers mentioned or the last saved value.
- Properties that are not mentioned in any resolver take the value from TIBCO Cloud Integration.

AWS Systems Manager Parameter Store

AWS Systems Manager Parameter Store is a capability provided by AWS Systems Manager for managing configuration data. You can use the Parameter Store to centrally store configuration parameters for your apps.

Your Flogo app connects to the AWS Systems Manager Parameter Store server as its client. At runtime, you are required to provide the Parameter Store server connection details by setting the `FLOGO_APP_PROPS_AWS` environment variable for your app to connect to the Parameter Store server. You have the option to enter the values for the Parameter Store connection parameters either by typing in their values as JSON strings or by creating a file that contains the values and using the file as input.

Using the Parameter Store

Below is a high-level workflow for using AWS Systems Manager Parameter Store with your Flogo app.

Before you begin

This document assumes that you have an AWS account, have access to the AWS Systems Manager, and know how to use the AWS Systems Manager Parameter Store. Refer to the AWS documentation for the information on the AWS Systems Manager Parameter Store.

Overview

To use the Parameter Store to override app properties set in Flogo Enterprise:

1. Build an app binary that has the app properties already configured in Flogo Enterprise. For more information on building an app binary, see [Building the App](#).
2. Configure the app properties that you want to override in the Parameter Store. At runtime, the app fetches these values from the Parameter Store and uses them to replace the default values that were set in the app.
3. Set the `FLOGO_APP_PROPS_AWS` environment variable to set the Parameter Store connection parameters from the command line.

When you run the command for setting the `FLOGO_APP_PROPS_AWS` environment variable, it runs your app, connects to the Parameter Store, and fetches the overridden values for the app properties from the Parameter Store. Only the values

for properties that were configured in the Parameter Store are overridden. The remaining app properties get their values from the **Application Properties** dialog.

See the [Setting the Parameter Store Connection Parameters](#) and [Parameter Store Connection Parameters](#) sections for details.

Parameter Store Connection Parameters

To connect to the AWS Systems Manager Parameter Store, provide the configuration below at runtime.

Property Name	Required	Data Type	Description
access_key_id	Yes	String	<p>Access ID for your AWS account. To protect the access key, an encrypted value can be provided in this configuration. See Encrypting Password Values section for information on how to encrypt a string.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;"> <p>Note: The encrypted value must be prefixed with SECRET: For example, SECRET:b0UaK3bTyD9wN+ZJkmLKRmojhAv+</p> </div> <p>This configuration is optional if <code>use_iam_role</code> is set to true.</p>
secret_access_key	Yes	String	<p>Secret access key for your AWS account. This account must have access to the Parameter Store. To protect the secret access key, an encrypted value can be provided in this configuration. See the Encrypting Password Values section for information on how to encrypt a string.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;"> <p>Note: The encrypted value must be prefixed with SECRET: For example, SECRET:b0UaK3bTyD9wN+ZJkmLKRmojhAv+</p> </div> <p>This configuration is optional if <code>use_iam_role</code> is set to true.</p>

Property Name	Required	Data Type	Description
region	Yes	String	Select a geographic area where your Parameter Store is located. This configuration is optional if <code>use_iam_role</code> is set to <code>true</code> and your Parameter Store is configured in the same region as the running service. When running in AWS services (for example, EC2, ECS, EKS), this configuration is optional if the Parameter Store is in the same region as these services.
param_prefix	No	String	<p>This is essentially the hierarchy that your app follows to get to the app property location in the Parameter Store. It is the prefix to be prepended to the lookup parameter. This is helpful in case the parameter hierarchy is not fixed and may change based on the environment during runtime.</p> <p>This is also helpful in case that you want to switch to a different configuration service such as the Consul KV store.</p> <p>As an example of a <code>param_prefix</code>, if you have an app property (for example, <code>Message</code>) that has two different values depending on the environment from which it is being accessed (for example, <code>dev</code> or <code>test</code> environment), your <code>param_prefix</code> for the two values can be <code>/dev/<APPNAME/</code> and <code>/test/<APPNAME/</code>. At run time, the right value for <code>Message</code> is picked up depending on which <code>param_prefix</code> you specify in the <code>FLOGO_APP_PROPS_AWS</code> environment variable. Hence, setting a <code>param_prefix</code> allows you to change the values of the app properties at runtime without modifying your app.</p>
use_iam_role	No	Boolean	Set to <code>true</code> if the Flogo app is running in the AWS services (such as EC2, ECS, EKS) and you want to use the IAM role (such as instance role or task role) to fetch parameters from the Parameter Store. In

Property Name	Required	Data Type	Description
			that case, <code>access_key_id</code> and <code>secret_access_key</code> are not required.
<code>session_token</code>	No	String	Enter session token if you are using temporary security credentials. Temporary credentials expire after a specified interval. For more information, see the AWS documentation.

Setting the Parameter Store Connection Parameters

You can use the AWS Systems Manager Parameter Store to override the property value set in your Flogo app. You do so by creating the property in the Parameter Store and assigning it the value with which to override the default value set in the app. You can create a standalone property or a hierarchy (group) in which your property resides.

Before you begin

This document assumes that you have an AWS account and the Parameter Store and are familiar with its use. Refer to the AWS documentation for more information on the Parameter Store.

To create a standalone property (without hierarchy), you simply enter the property name when creating it. To create a property within a hierarchy enter the hierarchy in the following format when creating the property: `<param_prefix>/<property_name>`, where `<param_prefix>` is a meaningful string or hierarchy that serves as a path to the property name in Parameter Store and `<property_name>` is the name of the app property whose value you want to override.

For example, in `dev/Timer/Message` and `test/Timer/Message/dev/Timer` and `test/Timer` are the `<param_prefix>` which could stand for the dev and test environments respectively, and `Message` is the key name. During runtime, you provide the `<param_prefix>` value, which tells your app the location in the Parameter Store from where to access the property values.

Warning: The parameter name in the Parameter Store must be identical to its counterpart (app property) in the **Application Properties** dialog in Flogo Enterprise. If the parameter names do not match exactly, a warning message is displayed, and the app uses the default value that you configured for the property in Flogo Enterprise.

Warning: A single app property, for example, Message, is looked up by your app as either Message or <param_prefix>/Message in the Parameter Store. An app property within a hierarchy such as x.y.z is looked up as x/y/z or <param_prefix>/x/y/z in the Parameter Store. Note that the dot in the hierarchy is represented by a forward slash (/) in the Parameter Store.

After you have configured the app properties in the Parameter Store, you need to set the environment variable, FLOGO_APP_PROPS_AWS, with the Parameter Store connection parameters for your app to connect to the Parameter Store. When you set the environment variable, it triggers your app to run, which connects to the Parameter Store using the Parameter Store connection parameters you provided and pulls the app property values from the param_prefix location you set by matching the app property name with the param_name. Hence, the property names must be identical to the app property names defined in the **Application Properties** dialog in Flogo Enterprise.

You can set the FLOGO_APP_PROPS_AWS environment variable either by manually entering the values as a JSON string on the command line or placing the properties in a file and using the file as input to the FLOGO_APP_PROPS_AWS environment variable.

If your Container is Not running on ECS or EKS

If the container in which your app resides is running external to ECS, you must enter the values for access_key_id and secret_access_key parameters when setting the FLOGO_APP_PROPS_AWS environment variable.

Entering the Parameter Store Values as a JSON string

To enter the Parameter Store connection parameters as a JSON string, enter the parameters and their value using the comma delimiter. The following example illustrates how to set the values as JSON strings. This would be run from the location where your app resides:


```
FLOGO_APP_PROPS_AWS="{\"access_key_id\":\"SECRET:XXXXXXXXXXXXX\",
\"secret_access_key\":\"SECRET:XXXXXXXXXXXXX\",
\"region\":\"us-west-2\",
\"session_
token\":\"SECRET:1UBrEIEzye8W1mmx7NLAiQzopmp58kUa02XdpmxYqVvkGKUrdN+wgCeH3
mxZ\"
\"param_prefix\":\"/MyFlogoApp/Dev/\"}"
```

Where `/MyFlogoApp/Dev/` is the `param_prefix` (path to the properties) and `SECRET` is the encrypted version of the key or `key_id` obtained from the Parameter Store.

This connects to the Parameter Store, pulls the values for the properties, and overrides the default values that were set in the app.

Refer to the [Parameter Store Connection Parameters](#) section for a description of the parameters.

Setting the Parameter Store values using a file

To set the parameter values in a file, create a `.json` file, for example, `aws_config.json` containing the parameter values. Here is an example:

```
{
  \"access_key_id\": \"SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+\",
  \"param_prefix\": \"/MyFlogoApp/dev/\",
  \"secret_access_key\": \"SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+\",
  \"region\": \"us-west-2\",
  \"session_
  token\":\"SECRET:1UBrEIEzye8W1mmx7NLAiQzopmp58kUa02XdpmxYqVvkGKUrdN+wgCeH3
  mxZ\"
}
```

Place the `aws_config.json` file in the same directory, which contains your app binary.

Run the following from the location where your app binary resides to set the `FLOGO_APP_PROPS_AWS` environment variable. For example, to use the `aws_config.json` file from the example above, run:

```
FLOGO_APP_PROPS_AWS=aws_config.json ./<app_binary_name>
```

This connects to the Parameter Store, pulls the overridden app properties values from the Parameter Store, and runs your app with those values.

If your Container is running on ECS or EKS

In case your Flogo apps are running in ECS and intend to use the EC2 instance credentials, set `use_iam_role` to `true`. The values for `access_key_id` and `secret_access_key` are gathered from the running container. Ensure that the ECS task has permission to access the param store.

The IAM role that you use must have permissions to access the parameter(s) from the AWS Systems Manager Parameter Store. The following policy must be configured for the IAM role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ssm:GetParameters",
        "ssm:GetParametersByPath",
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

The following is an example of how to set the `FLOGO_APP_PROPS_AWS` environment variable when your container is running on ECS. Notice that the values for `access_key_id` and `secret_access_key` are omitted:

```
FLOGO_APP_PROPS_AWS="{\"use_iam_role\":true, \"region\": \"us-west-2\"}"
./Timer-darwin-amd64
```

AWS AppConfig

AWS AppConfig is a feature provided by AWS System Manager, which lets you create, manage, and quickly deploy application configurations. You can use AWS AppConfig to simplify the task of configuring changes in application configuration, validating the changed configurations, deploying the new configurations and monitoring it.

Using AWS AppConfig, you can override the Flogo app properties at runtime. Your Flogo app retrieves configuration data by establishing the connection with AWS AppConfig. To enable the connection between your Flogo app and AWS AppConfig, you are required to set

the value of `FLOGO_APP_PROPS_AWS_APPCONFIG` to `True`. Here, the session retrieves the data from AppConfig only once at the start of the session.

Using the AppConfig

Below is a high-level work flow for using AWS Systems Manager AppConfig with your Flogo app.

Before you begin

This document assumes that you have an AWS account, have access to the AWS Systems Manager, and know how to use the AWS Systems Manager AppConfig. Refer to the AWS documentation for the information on the AWS Systems Manager AppConfig.

Overview

1. Build an app executable that has the app properties already configured in Flogo. For more information on building an app executable, see [Building an App Executable](#).

In case of TCI, for a new app, you need to set the engine variables for the Flogo app before pushing it to TCI. For an existing app you can configure the engine variables and push the updates to the app in the TCI.

2. Configure AWS AppConfig to work with your Flogo application. To define the properties in AWS AppConfig:
 - a. Create an application in AWS Appconfig to organize and manage configuration data.
 - b. Select the environment of the application in the Appconfig same as that of the environment of your Flogo app.
 - c. Create a configuration profile.

A configuration profile enables AWS AppConfig to access your hosted configuration versions in its stored location. You can store configurations in YAML, JSON, or as text documents in the AWS AppConfig hosted configuration store.

Refer to AWS documentation for detailed procedure to set up the AWS AppConfig.

3. Configure the app properties that you want to override in the AppConfig. At runtime,

the app fetches these values from the AppConfig and uses them to replace the default values that were set in your Flogo app.

4. Set the value of the parameter `FLOGO_APP_PROPS_AWS_APPCONFIG` to `True` to establish the connection between your Flogo app and AWS AppConfig.

Note: If you change the app properties values in AWS AppConfig, then you need to repush the app to TCI or re-execute the app executable.

AppConfig Client Configuration

IAM role that you would be using to fetch the configuration details must have permissions to access configurations from AWS AppConfig. For the same, Following policy must be configured for IAM role:

Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "appconfig:GetLatestConfiguration",
        "appconfig:StartConfigurationSession",
        "appconfig:ListApplications",
        "appconfig:GetApplication",
        "appconfig:ListEnvironments",
        "appconfig:GetEnvironment",
        "appconfig:ListConfigurationProfiles",
        "appconfig:GetConfigurationProfile",
        "appconfig:GetConfiguration",
        "appconfig:ListDeployments",
        "appconfig:GetDeployment"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

To connect to the AWS Systems Manager AppConfig, provide below configuration at runtime.

Property Name	Required	Data Type	Description
FLOGO_APP_PROPS_AWS_APPCONFIG	Yes	Boolean	Set this as True to enable the AWS AppConfig support feature.
AWS_APPCONFIG_PROFILE_NAME	Yes	String	This is name of the configuration profile created while defining the properties in AppConfig.
AWS_APPCONFIG_ENV_NAME	Yes	String	This is name of the environment provided while creating application in the AppConfig.
AWS_APPCONFIG_APP_IDENTIFIER_NAME	No	String	Set app identifier name for AWS AppConfig. If the name is not set, it takes the name as that of your Flogo app. It is required only if your AWS AppConfig app identifier name does not match with the Flogo app name.
AWS_APPCONFIG_REGION	No	String	Select AWS region where your Appconfig is located.

Property Name	Required	Data Type	Description
			This field is not required when your app binary (executable) is running on AWS EC2 instance in the same region as that of your AppConfig region. For all other cases, you must set the region.
AWS_APPCONFIG_ACCESS_KEY_ID	No	String	<p>If the access key ID is not provided, it is picked up by following the AWS default credentials provider chain.</p> <p>For flogo app deployment on TCI, you must provide this value.</p>
AWS_APPCONFIG_SECRET_ACCESS_KEY	No	String	<p>If the secret access key is not provided, it is picked up by following the AWS default credentials provider chain.</p> <p>For flogo app deployment on TCI, you must provide this value.</p>
AWS_APPCONFIG_SESSION_TOKEN	No	String	Set this if you want to use your session token for AWS AppConfig API calls.
AWS_APPCONFIG_ASSUME_ROLE_ARN	No	String	Set the assume role ARN if you want to use

Property Name	Required	Data Type	Description
			assumed role to fetch the values from AWS AppConfig.

✔ **Tip:** For sensitive fields such as ACCESS_KEY_ID, SECRET_ACCESS_KEY, and SESSION_TOKEN an encrypted value can be provided in this configuration. See the [Encrypting Password Values](#) section for information on how to encrypt a string.

ⓘ **Note:** The encrypted value must be prefixed with SECRET: For example, SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+

Environment Variables

Flogo Enterprise allows you to externalize the configuration of app properties using environment variables.

Using environment variables with app properties is a two-step process:

Procedure

1. Create one environment variable per app property.
2. Set the FLOGO_APP_PROPS_ENV=auto environment variable, which directs it to fetch the values of the app properties for which you have created environment variables.

ⓘ **Note:** App binaries that were generated from a version of Flogo Enterprise older than 2.4.0 do not support app properties override using environment variables. For example, if you attempt to run an older app binary from Flogo Enterprise 2.4.0 (which supports the environment variable functionality) and override app properties in the app using environment variables, the binary runs normally but the app property override gets ignored.

Exporting App Properties to a File

You can export the app properties to a JSON file or a `.properties` file. The exported JSON file can be used to override app property values. The `.properties` file can be used to create a ConfigMap in Kubernetes. When using the exported properties file, the values in the properties file get validated by the app during runtime. If a property value in the file is invalid, you get an error saying so and the app proceeds to use the default value for that property instead.

Exporting the app properties to a JSON file

Exporting the app properties to a JSON file allows you to override the default app property values during app runtime. It is useful if you want to test your app by plugging in different test data with successive test runs of your app. You can set the app properties in the exported file to a different value during each run of the app. The default app property values get overridden with the values that you set in the exported file.

To export the app properties to a JSON file, run the following command from the directory where your app resides:

```
./<app-binary-name> -export props-json
```

The properties get exported to `<app-binary-name>-props.json` file.

Exporting app properties to a `.properties` file

You cannot use a `.properties` file format to override the app properties that were externalized using environment variables. The `.properties` file is useful when creating the ConfigMap in Kubernetes. To export the app properties to a `.properties` file, run the following command from the directory where your app resides:

```
./<app-binary-name> -export props-env
```

The properties get exported to `<app-binary-name>-env.properties` file. The names of the app properties appear in all uppercase in the exported `env.properties` file. For example, a property named `Message` appears as `MESSAGE`. A hierarchy such as `x.y.z` appears as `X_Y_Z`.

Using a JSON File to Override App Property Values

To override an app prop using a JSON file:

Procedure

1. In the JSON file, make sure that the app property which you want to override is set as follows:

```
"<property>":"<value>"
```

For example:

```
{
  "IntegerOverrideVal":453,
  "StringOverridingValue":"hello",
  "BoolValue":true
}
```



Note: Only for certificates, the format of the property must be:

```
"<property>":"<encoded_value>"
```

To get the encoded value of the contents, you can use

<https://www.base64encode.org/> or any other base64 encoding tool.

2. Execute the binary of the app using the FLOGO_APP_PROPS_JSON environment variable as follows:

```
FLOGO_APP_PROPS_JSON=/<filepath>/<JSON filename>.json ./<binary>
```

Example: Overriding a Certificate Using a JSON File

You can override a server key and certificate using an app property. You would, typically, need to override a certificate if the existing certificate has expired or you want to use a custom certificate. You can directly override the certificate at runtime instead of re-configuring the app. In such a case:

Procedure

1. In the JSON file, set the ServerKey and ServerCertificate app properties as follows:

```
{
  "ServerKey":"LS0tLS1CRUdJTiBQVklWQVRFIEtFWS0tLS0tCk1JSUV2Zz0l",
```

```
"ServerCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1J",
}
```

2. Execute the binary of the app using the FLOGO_APP_PROPS_JSON environment variable as follows:

```
FLOGO_APP_PROPS_JSON=/home/john/Downloads/appPropOverride.json
./RestSSLService-linux_amd64
```

Overriding Security Certificate Values

The use of environment variables to assign new values to your app properties at runtime is a handy method that you can use to test your app with multiple data sets.



Warning: Using environment variables to override app properties in Lambda apps is not currently supported.

Follow these steps to set up the environment variables and use them during app runtime.

Step 1: Create environment variables for your app properties

You start by creating one environment variable for each app property that you want to externalize. To do so, run:

```
export <app-property-name>="<value>"
```

For example, if your app property name is `username`, run `export username="abc@xyz.com"` or `export USERNAME="abc@xyz.com"`

A few things to note about this command:

- Since special characters (such as `` | < > & @ ``) are shell command directives, if they appear in *value*, enclosing the *value* in double-quotes tells the system to treat such characters as literal values instead of shell command directives.
- The *app-property-name* must match the app property exactly or it can use all uppercase letters. For example, the app property, `Message`, can either be entered as `Message` or `MESSAGE`, but not as `message`.
- If you want to use a hierarchy for your app property, be sure to use underscores (`_`) between each level instead of the dot notation. For example, for an app property

named `x.y.z`, the environment variable name should be either `x_y_z` or `X_Y_Z`.

Step 2: Set `FLOGO_APP_PROPS_ENV=auto` environment variable

To use the environment variables during app runtime, set the `FLOGO_APP_PROPS_ENV=auto` environment variable.

To do so, run:

```
FLOGO_APP_PROPS_ENV=auto ./<app-binary>
```

For example, `FLOGO_APP_PROPS_ENV=auto MESSAGE="This is variable 1." LOGLEVEL=DEBUG ./Timer-darwin-amd64`



Note: When setting variables of type password be sure to encrypt its value for security reasons. For more information, see [Encrypting Password Values](#).

Setting the `FLOGO_APP_PROPS_ENV=auto` directs your app to search the list of environment variables for each app property by matching the environment variable name to the app property name. When it finds a matching environment variable for a property, the app pulls the value for the property from the environment variable and runs the app with those values. Hence, it is mandatory that the app property name exactly matches the environment variable name for the property.

App properties that were not set as environment variables pick up the default values set for them in the app. A warning message similar to the following is displayed in the output: `<property_name> could not be resolved. Using default values.`

Example: Overriding a Certificate Using an Environment Variable

You can override a server key and certificate using an app property. You would, typically, need to override a certificate if the existing certificate has expired or you want to use a custom certificate. You can directly override the certificate at runtime instead of reconfiguring the app. In such a case:

1. Export the base64 encoded values of the content of the file in the terminal itself as follows:

```
export ServerCertificate=<base64encodedCertificateFileContent>
```

```
export ServerKey=<base64encodedKeyFileContent>
```

2. Set the FLOGO_APP_PROPS_ENV=auto environment variable as follows:
FLOGO_APP_PROPS_ENV=auto ./<app-binary>

Encrypting Password Values

When entering passwords on the command line or in a file, it is always a good idea to encrypt their values for security reasons. Flogo Enterprise has a utility that you can use to encrypt passwords.

Before you begin

You must have the password to be encrypted handy to run the utility. To encrypt a password, run the following:

Procedure

1. Open a command prompt or a terminal.
2. Navigate to the location of the app binary and run the following command:

```
./<app_binary> --encryptsecret <value_to_be_encrypted>
```

The command outputs the encrypted value, which you can use when setting the password in a file or setting the password from the command line or using environment variables. For example, `export PASSWORD="SECRET:t90Ixj+QYCMFbqCEo/UnELlPPhrClMzv"`.

Note that the password value is enclosed in double-quotes. Since special characters (such as ``!|<, >, &, ``) are shell command directives, if such characters appear in the encrypted string, using double quotes around the encrypted value directs your system to treat them as literal characters. Also, the encrypted value must be preceded by `SECRET:`

Keep in mind that when you run the `env` command to list the environment variables, the command does not output the environment variable for the password.

Container Deployments

You can run Flogo apps as containerized apps in Docker containers and use Kubernetes to deploy, manage and scale the apps.

Kubernetes

You can package a Flogo app binary in a docker image, then push the docker image to a container registry and run the Flogo apps on a Kubernetes cluster as a pod.

For information on deploying apps in a Kubernetes environment, see [Deploying Flogo apps to a Kubernetes](#).

Deploying Flogo Apps to Kubernetes

You can deploy your Flogo apps to a Kubernetes Cluster running locally on bare metal servers, on VMs in hybrid cloud environments, or on fully managed services provided by various cloud providers such as Amazon EKS, Azure Container Service, or Google Kubernetes Engine. Refer to the Kubernetes documentation for more information. To do so, you must create a docker image locally for your app, then push the image to a container registry. When you apply the appropriate app deployment configuration to the Kubernetes cluster, one or more docker containers get created from the docker image that is encapsulated in one or more Kubernetes pods based on the deployment configuration.

Before you begin

You must have:

- The Kubernetes cluster running on your choice of environment
- Docker 1.18.x or greater installed on your machine
- `kubectl` installed on your machine

Procedure

1. Build a docker image for your app. You can use one of the following ways to build a docker image:

- **Using the UI:**

- a. Build a docker image using the Flogo Enterprise UI. For details, see [Building the App](#).
- b. Tag the generated docker image from the command line:

```
docker tag <image-id> <app-name>:<version>
```

the app tag must be in the format, *<app-name>:<app-version>*.

- **From a Linux binary:**

- a. Build a Linux binary using the Linux/amd64 option. For details, see [Building the App](#).
- b. Provide run permission to the app binary: `chmod +x <app-binary>`
- c. Create a docker file. For example:

```
FROM <OS-version> # for example, FROM alpine:3.7
WORKDIR /app
ADD <app-binary> <path-to-app-in-docker-container> # for example, ADD
flogo-rest-linux_amd64 /app/flogo-rest
CMD ["/app/flogo-rest"]
```

- d. Build the docker image using the docker file. Run the following command:

```
docker build -t <app-tag> -f <path-to-Dockerfile> .
```

the app tag must be in the format *<app-name>:<app-version>*

- **From the CLI:**

- a. Export your app as a JSON file (for example, `flogo-rest.json`) by clicking **Export app** on the flow details page.
- b. Build a Linux binary for the app from the CLI. Open a command prompt and change directory to `<FLOGO_HOME>/<version>/bin` and run:

```
builder-<platform>_<arch> build -p linux/amd64 -f <path-
to-the-.json-file>
```

This generates a linux app binary.

- c. Provide run permission to the app binary:

```
chmod +x <app-binary>
```

- d. Create a docker file. For example:

```
FROM <OS-version> # for example, FROM alpine:3.7
WORKDIR /app
ADD <app-binary> <path-to-app-in-docker-container> # for example, ADD
flogo-rest-linux_amd64 /app/flogo-rest
CMD ["/app/flogo-rest"]
```

- e. Build the docker image using the docker file. Run the following command:

```
docker build -t <app-tag> -f <path-to-Dockerfile> .
```

The app tag must be in the format `<app-name>:<app-version>`

2. Run the docker image locally to verify that all looks good:

```
docker run -it -p 9999:9999 <app-tag>
```

3. Authenticate docker with the container registry where you want to push the docker image.
4. Tag the docker image by running the following command:

```
docker tag <app-tag> <CONTAINER_REGISTRY_URI>/<app-tag>
```

the app tag must be in the format, `<app-name>:<app-version>`

5. Push the local docker image to the container registry by running the following command:

```
docker push <CONTAINER_REGISTRY_URI>/<app-tag>
```



Note: Refer to the documentation for your container registry for the exact commands to authenticate docker, tag docker image, and push it to the registry.

6. To deploy your app on Kubernetes, run your app by creating a Kubernetes deployment object. Follow these steps to do so:
 - a. Create a YAML file. For example, the YAML file below describes a deployment that runs the `gcr.io/<GCP_PROJECT_ID>/<docker-image-name>:<tag>` docker image on the Google Cloud.

```

apiVersion: apps/v1 # for versions before 1.9.0 use
apps/v1beta2
kind: Deployment
metadata:
  name: flogo-app-deployment
spec:
  selector:
    matchLabels:
      app: flogo-app
  replicas: 2 # tells deployment to run 2 pods matching the
template
  template:
    metadata:
      labels:
        app: flogo-app
    spec:
      containers:
        - name: flogo-app
          image: gcr.io/<GCP_PROJECT_ID>/<docker-image-
name>:<tag>
          ports:
            - containerPort: 9999

```

- b. Create a Kubernetes deployment by running the following command:

```
kubectl apply -f deployment.yaml
```

Using ConfigMaps with a Flogo App

Flogo apps running in Kubernetes can use ConfigMaps for the app configuration through environment variables. When you bind the ConfigMap with your pod, all the properties in the ConfigMap get injected into the pod as environment variables. If your pod has multiple containers, you can specify the container into which you want to inject the environment variables in the `.yaml` file of the app. When running the app in Kubernetes, you use the ConfigMap. You can create a ConfigMap using a `.property` file that was exported from your Flogo app.

To create a ConfigMap when running your app in Kubernetes:

! **Important:** If you update the app properties in Flogo Enterprise, you must recreate the ConfigMap and repush the app for your changes to take effect in Kubernetes.

Procedure

1. Export the Flogo app properties to a `.properties` file. Refer to the section [Exporting App Properties to a File](#) for details.
2. Update the generated `.properties` file as desired.
3. Create a ConfigMap using the `.properties` file. Run the following command:

```
kubectl create configmap <name-of-configmap-file-to-be-created> --from-env-file=<exported-app-prop-filename>.properties
```

For example, if your exported file name is `Timer-env.properties` and you want the generated ConfigMap to be called `flogo-rest-config` the command would be similar to the following:

```
kubectl create configmap flogo-rest-config --from-env-file=Timer-env.properties
```

4. Update the Kubernetes deployment configuration YAML file for the app to let your app know that you want to use environment variables. Add the following:

```
env:
  - name: "FLOGO_APP_PROPS_ENV"
    value: "auto"
  envFrom:
  - configMapRef:
      name: <name-of-the-configmap>
```

i **Note:** Refer to the Kubernetes documentation for instructions on how to configure a pod to use ConfigMaps.

5. Build the docker image for the app binary by running the following command:

```
docker build -t <CONTAINER_REGISTRY_URI>/<app-tag>
```

6. Push the resulting image to the container registry using the following command:

```
kubectl apply -f <appname>.yaml
```

Managing Sensitive Information Using Kubernetes Secrets

You can resolve the values of the app properties in a Flogo app deployed on Kubernetes using Kubernetes Secrets. Kubernetes secret object lets you store and manage sensitive information like passwords or keys. This section explains how a secret can be used with a Kubernetes pod.

For more information on Kubernetes secrets, refer to the Kubernetes documentation.

Configuring the Secrets

To use the Kubernetes secrets in a Flogo app, you must set `FLOGO_APP_PROPS_K8S_VOLUME` with the `volume_path` configuration parameter at runtime:

- The secret key name must match the app property name. For example, if the property is `DB_PASS`, the secret key name must be `DB_PASS`. For example:

```
echo -n 'flogo123>./DB_PASS.txt  
kubectl create secret generic my-first-secret --from-file=./DB_  
PASS.txt'
```

where `DB_PASS.txt` contains the password for the database and `DB_PASS` is set as a property in the Flogo app.

- If you want to use a hierarchy for your app property, ensure that you use an underscore (`_`) between each level instead of the dot notation in the name of the secret. For example, for an app property named `x.y.z`, the name of the secret must be `x_y_z`.

Specifying the Path of the Volume Where the Secrets are Mounted

To specify the path to the volume where the secrets are mounted, you can specify the `volume_path` parameter in a JSON file or as a JSON string.

In a JSON File

1. Set the `volume_path` parameter in a `.json` file. For example, `k8s_secrets_config.json` contains:

```
{  
  "volume_path": "/etc/test"  
}
```

2. Set the path to the `.json` file in the `FLOGO_APP_PROPS_K8S_VOLUME` environment variable. For example:

```
FLOGO_APP_PROPS_K8S_VOLUME=k8s_secrets_config.json
```

As a JSON String

Set the `FLOGO_APP_PROPS_K8S_VOLUME` environment variable as a JSON string as follows:

```
FLOGO_APP_PROPS_K8S_VOLUME="{\"volume_path\": \"\/etc\/test\"}"
```

Sample YAML File



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
---  
apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:
```

```

labels:
  app: sampleapp
  name: sampleapp
  namespace: default
spec:
  template:
    metadata:
      labels:
        app: sampleapp
    spec:
      containers:
        -
          env:
            -
              name: FLOGO_APP_PROPS_K8S_VOLUME
              value: "{\"volume_path\": \"/etc/test\"}"
            -
              name: FLOGO_APP_PROPS_ENV
              value: auto
          envFrom:
            -
              configMapRef:
                - name: first-configmap
              image: "gcr.io/<project_name>/sampleapp:latest"
              imagePullPolicy: Always
            - name: sampleapp
          volumeMounts:
            -
              mountPath: /etc/test
              name: test
              readOnly: true
      volumes:
        -
          name: test
          secret:
            secretName: my-first-secret

```

Amazon Elastic Container Service (ECS) and Fargate

You can package a Flogo app binary in a docker image, push the docker image to Amazon ECR, then run, manage, and scale the Flogo app in Docker containers using Amazon ECS and AWS Fargate.

Deploying a Flogo App to Amazon ECS and Fargate

Procedure

1. Build a Flogo app as a docker image.
2. Push the Flogo docker image to Amazon Elastic Container Registry (ECR) as follows:
 - a. Authenticate Docker to the ECR Registry using the following command. For more information, refer to the AWS documentation for [Registry Authentication](#).

```
aws ecr get-login
```

- b. Tag the Flogo app Docker image with the ECR registry, repository, and optional image tag name combination:

```
docker tag <flogo_app_docker_image> <aws_account_
id>.dkr.ecr.<region>.amazonaws.com/<ecr_repository_name>:<tag>
```

- c. Push the tagged Docker image to the ECR registry:

```
docker push <aws_account_
id>.dkr.ecr.<region>.amazonaws.com/<ecr_repository_name>:<tag>
```

3. Create a cluster in which to run your apps. For more information on how to create an Amazon ECS Cluster, refer to the AWS documentation for [Creating Cluster](#).
4. Create a task definition. The task definition defines what docker image to run and how to run it. For more information on how to create a task definition, refer to the AWS documentation available for [Creating Task Definition](#).
5. Run the app in containers. After creating the task definition, you can open the app containers either by manually running tasks or by creating a service using the Amazon ECS Service Scheduler. For more information on how to create a service, refer to the AWS documentation available at [Creating Service](#).

Pivotal Cloud Foundry

You can deploy a Flogo app binary to the Pivotal Application Service (PAS) of Pivotal Cloud Foundry (PCF) using the Binary Buildpack. For more information, see the section [Deploying a Flogo App to Pivotal Application Service](#).

Deploying a Flogo App to Pivotal Application Service

After installing the Cloud Foundry Command Line Interface (cf CLI), you can push a Flogo app to the Pivotal Application Service. For more information on Pivotal Cloud Foundry, Pivotal Application Service, and its CLI, refer to the Pivotal Cloud Foundry documentation.

Before you begin

- Run the following command to ensure that the Cloud Foundry command-line client is installed successfully:

```
$ cf version
```

This command returns information about the currently installed version of the Cloud Foundry command-line client. For example:

```
cf version 6.42.0+0cba12168.2019-01-10
```

- Run the following command to authenticate yourself in the Pivotal Cloud Foundry:

```
$ cf login
```

Building a Linux Binary

From the UI

To build a Linux binary from the UI:

Procedure

1. From the UI, build a Linux binary using the Linux/amd64 option. See the [Building the App](#) section for details.
2. Provide run permission to the app binary: `chmod +x <app-binary>`
3. Follow the steps in the appropriate section below.

From the CLI

To build a Linux binary from the CLI:

Procedure

1. Export your app as a JSON file (for example, `flogo-rest.json`) by clicking **Export app** on the flow details page.
2. Build a Linux binary for the app from the CLI. Open a command prompt and change the directory to `<FLOGO_HOME>/<version>/bin` and run:

```
builder-<platform>_<arch> build -p linux/amd64 -f <path-to-the-.json-file>
```

This generates a Linux app binary.

3. Provide run permission to the app binary: `chmod +x <app-binary>`
4. Follow the steps in the appropriate section below.

Without Using a manifest.yml File

Procedure

1. Create a temporary folder.
2. Copy the `linux/amd64` binary of the app, which you had created in [Building a Linux Binary](#) and save it to the temporary folder created in step 1.

Note:

- Ensure that you do not save the binary to a path that already contains other files and directories.
- In your Flogo app, for a REST trigger, ensure that the port is set to 8080 in the trigger configuration.

3. In a command window, navigate to the path where you saved the binary and run the following command:

```
$ cf push <NAME_IN_PCF> -c './<APP_BINARY_NAME>' -b binary_
```

```
buildpack -u none
```

For example:

```
cf push test1 -c ./Timer-linux_amd64 -b binary_buildpack -u none
```

For the `-u` argument, depending on the health check, provide value as `none`, `port`, `http`, or `process`. For example, if the app is a REST API exposing an HTTP endpoint, use `port` after `-u`.

i Note: In your Flogo app, for a REST trigger, ensure that the port is set to 8080 in the trigger configuration.

4. After successfully deploying the app to the Pivotal Application Service, you can check the log of the app using the following command:

```
$ cf logs <APP_NAME_IN_PCF> --recent
```

Using a manifest.yml File

Procedure

1. Create a temporary folder.
2. Copy the `linux/amd64` binary of the app, which you had created in [Building a Linux Binary](#) and save it to the temporary folder created in step 1.

i Note: In your Flogo app, for a REST trigger, ensure that the port is set to 8080 in the trigger configuration.

You have two options:

- If you do not mention `Path` in the `manifest.yml` file, you must have both `manifest.yml` and the app binary in the same directory.
- If you have the `manifest.yml` file and the app binary in different directories, you must mention the following in the `manifest.yml` file:


```
path: <app binary path>
```

3. Create a manifest file in YAML. The following manifest file illustrates some YAML conventions:

```
# this manifest deploys REST APP to Pivotal Cloud Foundry

applications:
- name: REST_APP
  memory: 100M
  instances: 1
  buildpack: binary_buildpack
  command: ./REST-linux_amd64
  disk_quota: 100M
  health-check-type: port
```

 **Note:** REST-linux_amd64 indicates the name of app binary.

4. Save the manifest.yml file and run the following command in the same directory:

```
$ cf push
```

Result

The Flogo app is successfully pushed to the Pivotal Cloud Foundry.

Using Spring Cloud Configuration to Override App Properties

You can use Spring Cloud Configuration to override the properties of Flogo apps running on Pivotal Cloud Foundry.

To do so:

1. [Create a Repository and Properties File on Github](#)
2. [Setup Spring Cloud Configuration on Pivotal Cloud Foundry](#)
3. [Use Spring Cloud Configuration Service with Flogo](#)

Create a Repository and Properties File on Github

Procedure

1. Create a repository on Github.
2. In the repository created in step 1 above, create properties file with the following file naming convention:

```
<APP_NAME>-<PROFILE>.properties
```

For example, if a Flogo app name is PCFAPP and the profile name is DEV, the properties file name must be PCFAPP-DEV.properties.

3. Populate the <APP_NAME>-<PROFILE>.properties file with the key-value pairs for the overridden app properties.

i Note:

- The name of the property must match the name of the app property. For example, if the app property is named `Message`, define the property in the properties file as:

```
Message="<value>"
```

- If the properties are in a group, define the property as:

```
<groupname>.<propertyname> = <value>
```

For example, if a property, `username`, is under the `email` group and its value is `xyz@abc.com`, define the property in the `.properties` file as:

```
email.username=xyz@abc.com
```

Setup Spring Cloud Configuration on Pivotal Cloud Foundry

Set up an instance of Config Server for Pivotal Cloud Foundry with the Git repository created above using Spring Cloud Services on Pivotal Cloud Foundry. Refer to Spring Cloud Services for PCF documentation for detailed instructions.

Using Spring Cloud Configuration Service with Flogo

Procedure

1. Bind the service instance of Spring Cloud Config Server to your Flogo app.
2. Navigate to the setting of the pushed app.
3. Under **User Provided Env Variables**, add the following environment variable:

```
FLOGO_APP_PROPS_SPRING_CLOUD = {"profile":"<PROFILE_NAME>"}
```

4. Restage the app and see the logs using the following command:

```
$ cf logs <APP_NAME_IN_PCF> --recent
```

Microsoft Azure Container Instances

You can deploy a Flogo app to a Microsoft Azure container instance using a Flogo app docker image. For more information, refer to the section, [Deploying Flogo Apps to Microsoft Azure Container Instances](#).

Deploying a Flogo App to a Microsoft Azure Container Instance

Before you begin

- Create a Microsoft Azure account.
- Download and install Microsoft Azure CLI.
- Create a docker image of the Flogo app that needs to be deployed to the Microsoft Azure Container Instance.
- For information on Microsoft Azure commands, refer to the Microsoft Azure documentation.

Procedure

1. Create a new resource group using the following command:

```
az group create -l <location> -n <name-of-group>
```

2. If you have not created an Azure Container Registry, create one using the following command. This Azure Container Registry stores all the images that are pushed to the registry.

```
az acr create -n <name-of-registry> -g <name-of-group> --sku  
<pricing-tier-plan> --admin-enabled true
```

i Note: You must set `--admin-enabled` to `true`.

3. Log in to Azure Container Registry using the following command:

```
az acr login -n <name-of-registry>
```

4. Tag and push the Flogo app docker image to Azure Container Registry using the following commands:

```
docker tag <app-tag> <CONTAINER_REGISTRY_URI>/<app-tag>  
docker push <CONTAINER_REGISTRY_URI>/<app-tag>
```

5. Create an Azure Container instance using the following command:

```
az container create  
-g <name-of-resource-group>  
--name <name-of-container>  
--image <name-of-image>  
--environment-variables <name=value name=value FLOGO_APP_PROPS_  
ENV=auto>  
--dns-name-label <dns-name-label-for-container-group>  
--ip-address Public  
--ports <port-to-open>  
--registry-login-server <name-of-container-image-registry-login-  
server>  
--registry-username <username>  
--registry-password <password>  
#NOTE: If--environment-variables FLOGO_APP_PROPS_ENV=auto is not
```

**set, the environment variables are not detected at Flogo runtime.
#NOTE: IP Address must be explicitly set to Public.**

For example:

```
az container create
-g flogodemo
--name flogoapp
--image flogoacr.azurecr.io/acs_flogo:latest
--environment-variables prop_str=azure FLOGO_APP_PROPS_ENV=auto --
dns-name-label flogoappazure
--ip-address Public
--ports 9999
--registry-login-server flogoacr.azurecr.io
--registry-username <username>
--registry-password <password>
#where prop_str is the app property defined in the flogo app which
is being overridden from this command
```

6. Get container logs using the following commands:

```
az container logs --resource-group <name-of-resource-group> --name
<name-of-container>
```

Deploying a Flogo App to a Microsoft Azure Container Instance Using a YAML File



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

Procedure

1. Create a YAML file as follows:

```
--- apiVersion: 2018-10-01
location: <location>
```

```

name: <name-of-YAML-file>
properties:
containers:
-
name: fe-app-yaml
properties:
environmentVariables:
-
name: <name-of-app-property>
value: <value-of-app-property>
-
name: <name-of-app-property>
value: <value-of-app-property>
-
name: <name-of-app-property>
secureValue: <value-of-app-property>
#NOTE: secureValue must be used for passwords
-
name: FLOGO_APP_PROPS_ENV
value: auto
#NOTE: If the environment variable FLOGO_APP_PROPS_ENV is not set to "auto", the environment variables are not detected at Flogo runtime.
image: "<image>"
ports:
-
port: <port-number>
resources:
requests:
cpu: 1
memoryInGb: <memory>
imageRegistryCredentials:
-
password: <password>
server: <server>
username: <username>
ipAddress: <IP-address>
ports:
-
port: <port-number>
protocol: <protocol>
type: Public
#NOTE: IP Address must be explicitly set to Public.

```

```
osType: <OS>
tags: ~
type: <type>
```

2. Run the following commands:

```
az container create --resource-group <name-of-resource-group> --
file <name-of-YAML-file>
az container show -g <name-of-resource-group> -n <name-of-
container>
```

3. After the app is deployed, you can access the app endpoint by accessing the public IP address of the Azure container instance followed by the resource path.

```
<IP-address>:<port>/<resource-path>
```

Google Cloud Run

You can package a Flogo app binary in a docker image, push the image to Google Container Registry, then deploy the app to Google Cloud Run.

i Note: Only apps with REST and GraphQL triggers work in Google Cloud Run.

Deploying a Flogo App to Google Cloud Run

Before you begin

- A Google Cloud account.
For more information, see [Google Cloud](#).
- Setup the Google Cloud command-line tool.

Create or import REST app

Design a new REST app using the UI or import an existing one into the UI.

Build and push a docker image to the container registry

- From the UI:
 - Create a Docker image of the app.
 - Push the Docker image to Google Container Registry.
For more information, see [Push Docker Image](#).
- From CLI:
 - Build a Linux/Amd64 binary using the CLI. For more information, see [Building the App from the CLI](#).
 - Create a Docker file of the app and copy it along with the app binary.
 - From the directory where the binary and docker files are placed, run the following command:

```
gcloud builds submit --tag gcr.io/[PROJECT-ID]/[IMAGE_NAME]:  
[IMAGE_TAG]
```

For example:

```
gcloud builds submit --tag gcr.io/227xxx/flogo-helloworld:1.0
```

Deploy app on Cloud Run

You can deploy the app to Cloud Run using the CLI or the Console. This section describes how to deploy the app using the CLI. For more information on deploying the app using the Console, refer [Cloud Run](#).

Procedure

1. Deploy the Flogo app using the following command:

```
gcloud beta run deploy --image gcr.io/[PROJECT-ID]/[IMAGE_NAME]:  
[IMAGE_TAG]
```

For example:


```
gcloud beta run deploy --image gcr.io/227xxx/flugo-helloworld:1.0
Please specify a region:

[1] us-central1

[2] cancel

Please enter your numeric choice: 1

To make this the default region, run `gcloud config set run/region
us-central1`.

Service name (helloworld):

Allow unauthenticated invocations to [helloworld] (y/N)? y
##NOTE: At this prompt, only if you enter Y, you are allowed to
hit an endpoint without authentication.

Deploying container to Cloud Run service [helloworld] in project
[227xxx] region [us-central1]

✓ Deploying new service... Done.

✓ Creating Revision...

✓ Routing traffic...

✓ Setting IAM Policy...

Done.

Service [helloworld] revision [helloworld-695fa56d-97d2-46b9-b037-
```

```
2dfada50aca5] has been deployed and is serving traffic at  
https://helloworld-pae7vs5yaq-uc.a.run.app
```

2. Make a call using the URL returned in the output. For example, you can make a call to the following URL returned in step 2:

```
https://helloworld-pae7vs5yaq-uc.a.run.app/greetings/Flogo
```

Red Hat OpenShift

You can package a Flogo app binary in a docker image, then push the docker image to a container registry and run the Flogo apps on Red Hat OpenShift.

Deploying a Flogo App to Red Hat OpenShift

Before you begin

- Ensure that you have a Red Hat OpenShift account and that the Red Hat OpenShift environment is set up to deploy the app.
- Ensure that the Red Hat OpenShift CLI is installed on your machine.
- Ensure that the image of the Flogo app is pushed to the Red Hat OpenShift internal registry or any other public registry such as Docker Hub.

Procedure

1. Build a docker image for your app. You can build a docker image in one of the following ways.
 - **Using the UI:**
 - a. Build a docker image.
 - b. Tag the generated docker image from the command line: `docker tag <image-id> <app-name>:<version>` The app tag must be in the format `<app-name>:<app-version>`
 - **From a Linux binary:**

- a. Build a Linux binary using the Linux/amd64 option. For more information, see [Building the App](#).
- b. Provide execute permission to the app binary: `chmod +x <app-binary>`
- c. Create a docker file. For example:

```
FROM <OS-version> # for example, FROM alpine:3.7
WORKDIR /app
ADD <app-binary> <path-to-app-in-docker-container> # for example, ADD
flogo-rest-linux_amd64 /app/flogo-rest
CMD ["/app/flogo-rest"]
```

- d. Build the docker image using the docker file. Run the following command:

```
docker build -t <app-tag> -f <path-to-Dockerfile> .
```

The app tag must be in the format `<app-name>:<app-version>`

- **From the CLI:**

- a. Export your app as a JSON file (for example, `flogo-rest.json`) by clicking **Export app** on the flow details page.
- b. Build a Docker image containing the app using the builder command from the CLI. Open a command prompt and change directory to `<FLOGO_HOME>/<version>/bin` and run:

```
builder-<platform>_<arch> build -f <path-to-the.json-file>
-docker -n <docker_image_name>:<tag>
```

For example:

```
builder_linux_amd64 build -f flogo-rest.json -docker -n
flogo-rest:v1
```

For more information on the builder command, refer to the section, [Builder command](#).

2. Run the docker image locally to verify that everything is fine:

```
docker run -it -p 9999:9999 <app-tag>
```

3. Authenticate docker with the container registry where you want to push the docker image.
4. Tag the docker image. Run:

```
docker tag <app-tag> <CONTAINER_REGISTRY_URI>/<app-tag>
```

The app tag must be in the format `<app-name>:<app-version>`

5. Push the local docker image to the container registry by running the following command:

```
docker push <CONTAINER_REGISTRY_URI>/<app-tag>
```

i Note: Refer to the documentation for your container registry for the exact commands to authenticate docker, tag docker image, and push it to the registry.

6. Login to OpenShift from command line:

```
oc login --token=<Your token> --server=https://<host address>:<port>
```

For example:

```
oc login --token=<Your token> --server=https://api.ca-central-1.starter.openshift-online.com:6443
```

7. Create a project in Red Hat OpenShift:

```
oc new-project <PROJECT_NAME>
```

8. Deploy the app on Red Hat OpenShift using a YAML file. For a sample YAML file, see [Sample YAML File: Red Hat OpenShift](#).

```
oc create -f <YAML filename>
```

9. To get information about pods, run the following command:

```
oc get pods
```

The following is a sample output of the command:

```
# oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
flogo-graphql-599b9b4947-z4wqv      1/1     Running  0           3d
http-config-68595b747c-f7k6s       1/1     Running  0           23h
rest-basic-app-tmg5r                1/1     Running  0           1d
```

- To get the logs of a particular pod, run the following command:

```
oc logs <pod name>
```

The following is a sample output of the command:

```
# oc logs flogo-graphql-599b9b4947-z4wqv
TIBCO Flogo® Runtime - 2.7.0 (Powered by Project Flogo™ - master)
TIBCO Flogo® connector for General - 1.1.0.227
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Starting TIBCO Flogo® Runtime
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2019-11-02T03:33:46.191Z INFO [flogo.trigger.graphql] - Initializing GraphQL Trigger - [GraphQLTrigger]
2019-11-02T03:33:46.192Z INFO [flogo.trigger.graphql] - Starting GraphQL Server...
2019-11-02T03:33:46.192Z INFO [flogo.trigger.graphql] - Secure:[false] Port:[7879] Path:[/graphql]
2019-11-02T03:33:46.192Z INFO [flogo.engine] - Starting app [ graphql-app ] with version [ 1.1.0 ]
2019-11-02T03:33:46.192Z INFO [flogo.engine] - Engine Starting...
2019-11-02T03:33:46.192Z INFO [flogo.engine] - Starting Services...
2019-11-02T03:33:46.192Z INFO [flogo] - ActionRunner Service: Started
2019-11-02T03:33:46.192Z INFO [flogo.engine] - Started Services
2019-11-02T03:33:46.192Z INFO [flogo.engine] - Starting Application...
2019-11-02T03:33:46.192Z INFO [flogo] - Starting Triggers...
2019-11-02T03:33:46.193Z INFO [flogo] - Trigger [ GraphQLTrigger ]: Started
2019-11-02T03:33:46.193Z INFO [flogo] - Triggers Started
2019-11-02T03:33:46.193Z INFO [flogo.engine] - Application Started
2019-11-02T03:33:46.193Z INFO [flogo.engine] - Engine Started
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Runtime started in 9.821752ms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2019-11-02T03:33:46.193Z INFO [flogo] - Management Service started successfully on Port[7777]
```

- To access the endpoint of an app, run the following command:

```
oc get svc -o wide
```

The following is a sample output of the command:

```
oc get svc -o wide
NAME      TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE      SELECTOR
http-config  LoadBalancer  172.30.160.35   ac64e203feeb1-597152199.ca-central-1.elb.amazonaws.com  80:31970/TCP,8090:32445/TCP  23h     app=http-config
rest-basic-app  LoadBalancer  172.30.191.133  a852474a5fedd1-987646022.ca-central-1.elb.amazonaws.com  80:31277/TCP              1d       app=rest-basic-app
```

- From the output, note the external IP and port. Access the endpoint using the following URL:

```
http:<external IP>:<port>/<resource_context_path>
```

Sample YAML File: Red Hat OpenShift

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

The following is a sample YAML file for a REST app:

```
apiVersion: v1
kind: Service
metadata:
  name: flogo-rest
  labels:
    app: flogo-rest
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 9999
    name: app
  selector:
    app: flogo-rest
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: flogo-rest
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: flogo-rest
    spec:
      containers:
      - name: flogo-rest
        image: <DOCKER_REPOSITORY_NAME>/<APP_IMAGE_NAME>
        ports:
        - containerPort: 9999
```

Serverless Deployments

Developing for Lambda

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). Lambda functions automatically run pieces of code in response to specific events while also managing the resources that the code requires to run. Refer to the AWS documentation for more details on AWS Lambda.

Creating a Connection with the AWS Connector

You must create AWS connections before you use the Lambda trigger or Activity in a flow.

i Note: AWS Lambda is supported on the Linux platform only.

To create an AWS connection:

Procedure

1. In Flogo Enterprise, click **Connections** to open its page.
2. Click the **AWS Connector** card.
3. Enter the connection details. Refer to the section AWS Connection Details for details on the connection parameters.
4. Click **Save**.

Your connection gets created and is available for you to select in the drop-down menu when adding a **Lambda** Activity or trigger.

AWS Connection Details

To establish the connection, you must specify the following configurations in the **AWS Connector** dialog.

The **AWS Connector** dialog contains the following fields:

Field	Description
Name	Specify a unique name for the connection that you are creating. This is displayed in the connection drop-down list for all the activities.
Description	A short description of the connection.
Custom Endpoint	<p>(Optional) To enable the AWS connection to an AWS or AWS compatible service running at the URL specified in the Endpoint field, set this field to True.</p> <p>This field is not supported in TIBCO Flogo® Connector for Amazon Glacier.</p>
Endpoint	<p>This field is available only when Custom Endpoint is set to True.</p> <p>Enter the service endpoint URL in the following format: <code><protocol>://<host>:<port></code>. For example, you can configure a MinIO cloud storage server endpoint.</p>
Region	Region for AWS connection.
Authentication Type	<p>Select one of the following authentication types as required:</p> <ul style="list-style-type: none"> • AWS Credentials: Use this authentication to connect to AWS resources using access key, secret key, and assumed role. • Default Credentials: Use this authentication to use a role configured AWS resource such as EC2, ECS, or EKS without configuring the AWS credentials. Credentials are loaded using the AWS default credentials provider chain. <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note: To use Default Credentials as the Authentication Type in TIBCO Flogo® Connector for Amazon SQS and AWS Lambda, create an AWS connection using the Authentication Type as AWS Credentials and override AWS Credentials to Default Credentials at runtime.</p> </div>
Access key ID	<p>Access key ID of the AWS account (from the Security Credentials field of IAM Management Console).</p> <p>For more information, see the AWS documentation.</p>

Field	Description
Secret access key	<p>Enter the secret access key. This is the access key ID that is associated with your AWS account.</p> <p>For more information, see the AWS documentation.</p>
Session token	<p>(Optional) Enter session token if you are using temporary security credentials. Temporary credentials expire after a specified interval. For more information, see the AWS documentation.</p>
Use Assume Role	<p>This enables you to assume a role from another AWS account. By default, it is set to False (indicating that you cannot assume a role from another AWS account).</p> <p>When set to True, provide the following information:</p> <ul style="list-style-type: none"> • Role ARN - Amazon Resource Name of the role to be assumed • Role Session Name - Any string used to identify the assumed role session • External ID - A unique identifier that might be required when you assume a role in another account • Expiration Duration - The duration in seconds of the role session. The value can range from 900 seconds (15 minutes) to the maximum session duration setting that you specify for the role <p>For more information, see the AWS documentation.</p>

Creating a Flow with Receive Lambda Invocation Trigger

The **Receive Lambda Invocation** trigger allows you to create a Flogo flow to create and deploy as a Lambda function on AWS.

Refer to the "Receive Lambda Invocation Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details on the trigger.

To create a flow with the **Receive Lambda Invocation** trigger:

Procedure

1. Create an app in Flogo.

2. Click the app name on the Apps page to open it.
3. Click **Create a Flow**.
The **Create a Flow** dialog box opens.
4. Enter a name for the flow in the **Flow Name** text box.
Flow names within an app must be unique.
5. Optionally, enter a brief description of what the flow does in the **Description** text box and click **Save**.
A flow gets created. Click the flow name to open the flow page.
6. From the **Triggers** palette, select **Receive Lambda Invocation** and drag it to the triggers area.
7. To configure a trigger, enter the JSON schema or JSON sample data for the operation. This is the schema for the request payload.
8. Click **Continue**.
A flow beginning with the **ReceiveLambdaInvocation** trigger gets created.
9. Click the **ReceiveLambdaInvocation** trigger tile and configure its properties. See the "ReceiveLambdaInvocation" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details.

Deploying a Flow as a Lambda Function on AWS

After you have created the flow, you can deploy it as a Lambda function on AWS.

Before you begin

Note the following points:

- The flow must be configured with the **ReceiveLambdaInvocation** trigger.
- If the execution role name is not provided in the **ReceiveLambdaInvocation** trigger, then the Lambda function is created with the default **AWSLambdaBasicExecutionRole** role. It has the following Amazon CloudWatch permissions:
 - Allow: logs:CreateLogGroup
 - Allow: logs:CreateLogStream

- Allow: `logs:PutLogEvents`

If a non-existing execution role is provided, then the user whose AWS credentials are used in the AWS connection should have the following permissions:

- `iam:CreateRole`
- `sts:AssumeRole`

To deploy a Flogo app as a Lambda function, user role can have access to following `AWSLambda_FullAccess` policy which has all the required access.

To deploy a flow as a Lambda function on AWS:

Procedure

1. Build your Flogo app (`<myApp>`) with the `Linux/amd64` target. This is because Lambda deployments are Linux-based and building the binary for `Linux/amd64` generates the appropriate artifact to deploy in your AWS Lambda function. Refer to [Building the App](#) for details on how to build an app.
2. Add execution permission to the native `Linux/amd64` executable file that you built. Run `chmod +x <myApp>-linux_amd64`
3. You can deploy the `<myApp>-linux_amd64` in one of two ways:
 - If you are using a Linux environment to design, build, and deploy your apps, you can directly run the following command:

```
<LambdaTriggerBinary> --deploy lambda --aws-access-key <secret_key>
```

For example, `myApp-Linux64 --deploy lambda --aws-access-key xxxxxxxxx`

i Note: Ensure that the `aws-access-key` is identical to the one configured in the Flogo UI for the selected AWS Connection. This is used for validation with the `aws-access-key` configured as part of the AWS Connection within the UI and the value provided here does not overwrite the `aws-access-key` used while designing the app.

This approach of deploying to AWS Lambda works only on Linux platforms.

- If you are using a non-Linux environment to design, build, and deploy apps, then use this approach:
 - a. Build your Flogo app (`<myApp>`) with the `Linux/amd64` target.
 - b. Rename the Flogo executable file to `bootstrap`. This is mandatory per new

`provided.al2` and `provided.al2023` runtimes.

- c. Compress the executable file and rename it to `<myFunctionName>.zip`.
- d. From the AWS Lambda UI, create a Lambda function with Amazon Linux 2023 runtime.
- e. Create a role or attach an existing role in the Execution role.
- f. Click **Create function**.
- g. Go to **Code source**, click **Upload from** and upload the compressed file.

After successful deployment, the Lambda function is created in the AWS Lambda console.

- To override app properties used in a Lambda app during runtime, create a `.properties` or `.json` file containing the properties and their values to override, then use the command:

```
./<Lambda-app-name> --deploy --env-config <app-property-file-name>.properties
```

For example:

```
./MyLambdaApp --deploy --env-config MyLambdaApp-env.properties
```

where *MyLambdaApp* is the Lambda app name and *MyLambdaApp-env.properties* is the properties file name.

All properties in the `.properties` or `.json` file are passed to Lambda as environment variables.

Deploying a Flow as a Lambda Function on AWS using AWS CLI

Procedure

1. Build your Flogo App (*<myApp>*) with the `Linux/amd64` target. This is because Lambda deployments are Linux-based and building the binary for `Linux/amd64` generates the appropriate artifact to deploy in your AWS Lambda function.
2. Rename the Flogo executable to `bootstrap`. This is mandatory as per new `provided.al2` and `provided.al2023` runtimes.
3. Compress the executable file and rename it to `myFunction.zip`.
4. Run the AWS CLI:

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures x86_64 \  
--role arn:aws:iam::111122223333:role/lambda-ex \  
--region us-west-2 \  
--zip-file fileb://myFunction.zip
```

Creating a Flow with AWS API Gateway Lambda Trigger

The **AWS API Gateway Lambda** trigger allows you to invoke Lambda functions as REST APIs. A flow created in an app using the AWS API Gateway trigger is deployed as a Lambda function.

Refer to the "AWS API Gateway Lambda Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details on the trigger.

To create a flow with the **AWS API Gateway Lambda** trigger:

Procedure

1. Create an app in Flogo.
2. Click the app name on the Apps page to open it.
3. Click **Create a Flow**.
The **Create a Flow** dialog box opens.
4. Enter a name for the flow in the **Flow Name** text box.
Flow names within an app must be unique.
5. Optionally, enter a brief description of what the flow does in the **Description** text box and click **Next**.
A flow gets created. Click the flow name to open the flow page.
6. From the **Triggers** palette, select **Receive Lambda Invocation** and drag it to the triggers area.
7. Provide the method, resource path, and JSON schema for the operation.
8. Click **Continue**.
A flow beginning with the **AWSAPIGatewayLambda** trigger is created.
9. Click **Copy schema** or **Just add the trigger**.

10. Click the **AWSAPIGatewayLambda** trigger tile and configure its properties. See the "AWS API Gateway Lambda Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details.

What to do next

Deploy the flow on AWS. For instructions on how to do so, see [Deploying a Flow as a Lambda Function on AWS](#).

Creating a Flow with S3 Bucket Event Lambda Trigger

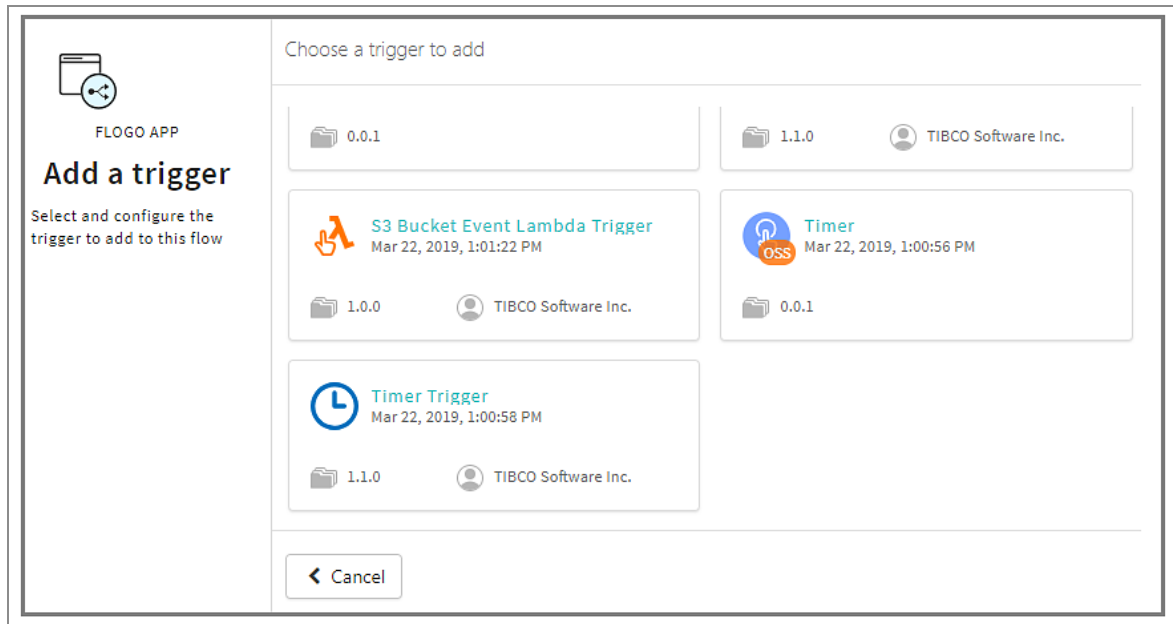
The **S3 Bucket Event Lambda** trigger allows you to create a flow using the operations or events that are performed on an S3 bucket trigger, a Lambda function.

i Note: Creating a new event or updating an existing event in the S3 Bucket Event Lambda trigger and re-pushing the app deletes existing Events on AWS S3.

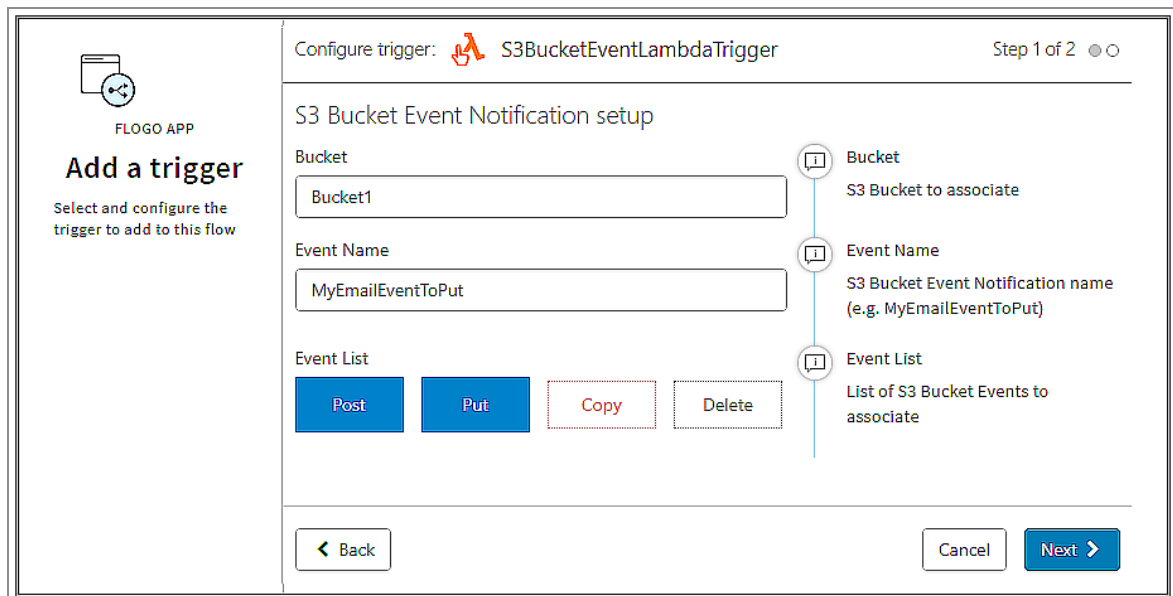
Refer to the "S3 Bucket Event Lambda Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details on the trigger. To create a flow with the **S3 Bucket Event Lambda** trigger:

Procedure

1. Create an app in Flogo Enterprise.
2. Click the app name on the apps page to open its page.
3. Click **Create a Flow**.
The **Create a Flow** dialog box opens.
4. Enter a name for the flow in the **Flow Name** text box.
Flow names within an app must be unique. An app cannot contain two flows with the same name.
5. Optionally, enter a brief description of what the flow does in the **Description** text box and click **Next**.
6. Click **Start with a trigger**.
7. Under **Choose a trigger to add**, click **S3 Bucket Event Lambda Trigger**.



- Provide the bucket name, event name, and the list of events to be performed. See the "S3 Bucket Event Lambda Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details.



- Provide any prefix or suffix object filters.

The screenshot shows the 'Configure trigger' interface for 'S3BucketEventLambdaTrigger' in the FLOGO APP. The interface is titled 'Step 2 of 2'. On the left, there is a sidebar with the FLOGO APP logo and the text 'Add a trigger' and 'Select and configure the trigger to add to this flow'. The main area is titled 'S3 Bucket Object Prefix and Suffix filter'. It contains two input fields: 'Object Prefix filter' and 'Object Suffix filter'. To the right of each field is a help icon and a description: 'Object Prefix filter Optional S3 Bucket Prefix to filter (e.g. images/)' and 'Object Suffix filter Optional S3 Bucket Suffix to filter (e.g. .jpg)'. At the bottom, there are three buttons: 'Back', 'Cancel', and 'Continue' (highlighted in blue with a checkmark).

10. Click **Continue**.

A flow beginning with the **S3 Bucket Event Lambda** trigger is created.

11. Click **Copy schema** or **Just add the trigger**.

12. Click the **S3 Bucket Event Lambda** trigger tile and configure its properties. See the "S3 Bucket Event Lambda Trigger" section in the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details.

13. Create a flow containing the Business logic of the Lambda function that you want to trigger using the S3 Bucket Event Lambda trigger.

What to do next

Deploy the flow on AWS. For instructions, see [Deploying a Flow as a Lambda Function on AWS](#).

S3 Bucket Event Lambda Trigger

Use the **S3 Bucket Event Lambda** trigger to trigger a Lambda function when a supported event occurs on the associated S3 bucket.

Trigger Settings

Note:

- Creating a new event or updating an existing event in the S3 Bucket Event Lambda trigger and re-pushing the app deletes existing Events on AWS S3.
- You can have only one S3 trigger in an app. An app that has an S3 trigger cannot contain any other triggers including another S3 trigger. The S3 trigger supports multiple handlers (flows), so you can have multiple flows in the app that are attached to the same S3 trigger. You can also have blank flows in the app which can serve as subflows for the flows that are attached to the S3 trigger.
- For overriding app properties, use the FLOGO_APP_PROPS_JSON environment variable only. You cannot override app properties using the FLOGO_APP_PROPS_ENV environment variable.

Field	Description
AWS Connection Name	(Mandatory) Name of the AWS connection that you want to use for deploying the flow.
Execution Role Name	Permission of the Lambda function to execute. The role must be assumable by Lambda and must have CloudWatch logs permission execution role. By default, Cloud watching is enabled.
Bucket	Name of the S3 bucket with which the trigger is to be associated. This bucket must be an existing one.
Event name	Name of the S3 bucket event notification.
Event list	A list of operations to be performed on the S3 bucket. Supported operations are POST, PUT, COPY, and DELETE.
Object prefix filter	(Optional) The prefix is to be used to filter the S3 bucket.

Field	Description
	For example, images/
Object suffix filter	(Optional) The suffix is to be used to filter the S3 bucket. For example, .jpg

Map to Flow Inputs

Map the flow output to the trigger reply on this tab. The tab displays the following fields.

Field	Description
Function	Information about the Lambda function
Context	Envelope information about this invocation
Identity	Identity for the invoking users
ClientApp	Metadata about the calling app
S3Event	Default schema of S3 bucket event trigger. It can be mapped with the flow input to pass the key values to the flow.

Deploying a Flogo App to Microsoft Azure Functions

After you have designed a Flogo app or imported an existing one, you can deploy it to Microsoft Azure Functions as a custom Docker container. You can do this by using the Microsoft Azure portal or do it by using the CLI.

Before you begin

Make sure you have a Microsoft Azure account with an active subscription and you can log in to the [Azure Portal](#). For more information on getting a Microsoft Azure account, see [Microsoft Azure](#).

Creating the Azure Function App in the Azure Portal

Before you begin

Install the following:

- To push images to the Azure Container registry, install the latest version of Azure CLI.
- Install Docker. For the supported versions, see the *Readme*.

Procedure

1. Build a Docker image of your Flogo app.
2. If you are using an Azure container registry, log in to the repository created on the Azure container registry.
3. Tag and push the Docker image of the Flogo app to the repository in the Azure container registry. For example:

```
docker tag flogo/hello-world:latest myregistry.azurecr.io/flogo-hello-world:latest
```

4. In the Azure portal, create a new Azure Function app. While creating the Azure Function app, in the **Instance details** dialog box, select **Docker Container** as the **Publish mode**.
5. After the Azure Function app is created, go to **Settings > Container Settings (Classic)** on the left navigation pane.
6. On the right pane, select the image **Source**, enter other details, and click **Save**.

✔ **Tip:** If you select **Registry Settings > Registry Source** as **Azure Container Registry**, and you face issues while selecting the **Registry**, verify the Repository permissions for the repository created for the Flogo app. Update the **Azure Container Registry** and enable Admin user; this enables the Azure function to access the images in the repositories.

7. If you are using a trigger port other than 80 or 8080, navigate to **Settings > Configuration** and click **New application setting**. Specify:
 - **Name:** WEBSITES_PORT

- **Value:** <your trigger port>
8. Click **Save**. The app is restarted and changes made are reflected in the app.
 9. To copy the URL for your app and check whether it is working, go to the **Overview** menu.

i Note: Do NOT add the port declared in the trigger settings to the URL. If the URL does not work, restart the app manually.

10. For an app with app properties, to override the app properties, add the following to **Configuration > Application settings**:
 - All the app properties
 - **FLOGO_APP_PROPS_ENV=auto**
11. Save the settings. The app restarts when you save the properties.

Creating the Azure Function App from the Azure CLI

Before you begin

Install the following:

- **Visual Studio Code:** For more information, see [Visual Studio Code](#).
- **Azure Functions extension for Visual Studio Code:** For more information, see [Visual Studio Code - Azure Functions](#).
- **Azure Functions Core Tools (version 3.x or higher):** For more information, see [Local Azure Functions](#).

Procedure

1. Create a new directory (for example, flogo-func-project) and open it.

```
cd flogo-func-project
```

2. Create a new function project using the following command:

```
func init --worker-runtime custom --docker
```

The `--docker` option generates a Dockerfile for the project.

3. Add a new function from a template using the following command:

```
func new --name <your-app-name> --template "HTTP trigger"
```

Here:

`--name` argument is the unique name of your function

`--template` argument specifies the template based on which the function is created

Example:

```
func new --name hello-world --template "HTTP trigger"
```

4. Download or build the binary for your HTTP trigger app and copy it into the directory you created earlier.

For example, copy `hello-world.json` to the `flogo-func-project` directory.

5. If it is not an executable file, make it executable by running the following command:

```
chmod +x <binary-filename>
```

6. Add the following script to your project folder with the name `start.sh`:

start.sh

```
#!/usr/bin/env sh
echo "Starting function..."
PORT=${FUNCTIONS_CUSTOMHANDLER_PORT} ./hello-world-linux_amd64
```

7. Make it executable by running the following command:

```
chmod +x start.sh
```

8. To update the default app prefix from `api` to your prefix, edit the `function.json` file. For example, you can change the prefix to `hello-world`.

function.json

```
{
  "bindings":
    [
      {
        "authLevel": "anonymous",
        "type": "httpTrigger",
        "direction": "in",
        "name": "req",
        "methods": ["get", "post"],
        "route": "books/{bookID}"
      },
      {
        "type": "http",
        "direction": "out",
        "name": "res"
      }
    ]
}
```

9. To add customHeaders in the extensions, edit the host.json file:

host.json

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "version": "2.0",
  "logging":
    {
      "applicationInsights":
        {
          "samplingSettings":
            {
              "isEnabled": true,
              "excludedTypes": "Request"
            }
        }
    },
}
```

```

"extensionBundle":
  {
    "id": "Microsoft.Azure.Functions.ExtensionBundle",
    "version": "[2.*, 3.0.0)"
  },
"customHandler":
  {
    "description":
      {
        "defaultExecutablePath": "start.sh",
        "workingDirectory": "",
        "arguments": []
      },
    "enableForwardingHttpRequest": true
  },
"extensions":
  {
    "http":
      {
        "routePrefix": ""
      }
  }
}

```

- To change the path, edit the Dockerfile:

Dockerfile

```

FROM mcr.microsoft.com/azure-functions/dotnet:3.0-appservice
ENV AzureWebJobsScriptRoot=/home/site/wwwroot \
    AzureFunctionsJobHost__Logging__Console__IsEnabled=true
COPY . /home/site/wwwroot

```

Your folder structure should now look similar to the following:

```

.
├── Dockerfile
├── hello-world-app
│   └── function.json
└── hello-world-rest-trigger-linux_amd64

```

```
├─ host.json
├─ local.settings.json
└─ start.sh
```

11. Test your app locally by running the following command:

```
func start
```

The output returns an URL for the app.

12. Test whether the app works by navigating to the URL provided in the output. For example:

```
http://localhost:7071/hello-world
```

13. Publish your app to Microsoft Azure. For more information, see [Publish the project to Azure](#).

Deploying a Flogo App in Knative

You can create and deploy a Flogoapp as a Knative service. For information on Knative, see the [Knative documentation](#).

A Flogoapp running inside a Docker container is called by a Knative service. For the app to be called by the Knative service, the app must be exposed over an HTTP port. In this section, a REST Trigger is used to expose the app over an HTTP port.

Before you begin

Make sure you meet the following requirements:

- Install the following components by using the instructions from [Getting Started with Knative](#):
 - Kind (Kubernetes in Docker)
 - Kubernetes CLI (kubectl)
 - Knative CLI (kn)
 - Knative "Quickstart" environment

- Create a Knative service and make sure you can ping the service endpoint. For details, see [Deploying your first Knative Service](#) and [Ping your Knative Service](#).

i Note: This section uses a **Knative on Kind** setup to explain the procedure. However, you can also set it up on minikube and Docker Desktop. For more information, see [Setup Knative on Minikube](#) and [Setup Knative on Docker Desktop](#).

Procedure

1. Configure a sample Flogo app with a REST Trigger exposed with a port. You can use the default REST Trigger port, 9999.

i Important:

- Only apps with HTTP endpoints can be deployed as a Knative service. Hence, a REST Trigger is used in this procedure.
- An app with multiple endpoints on different ports cannot be deployed as a Knative service.

The screenshot shows the configuration window for a 'ReceiveHTTPMessage' trigger. The 'Trigger Settings' section is highlighted with a red box, showing the 'Port' field set to 9999. The 'Handler Settings' section is also highlighted with a red box, showing the 'Path' field set to /hello/knative. The 'Method' is set to GET. The 'Output Validation' is set to False. The 'Configure Using API Specs' is set to False. The 'Output Settings', 'Map to Flow Inputs', 'Reply Settings', and 'Map from Flow Outputs' sections are also visible but not highlighted.

2. Build the Flogo app for **Linux/amd64** platform and save the binary file locally. For more information on building the app binary, see [Building the App](#).
3. Give executable permissions to the app binary:

```
chmod a+x <app_executable>
```

4. Build a Docker image for the Flogo app and tag it:

```
docker build --file Dockerfile -t dev.local/flogoknative:1.0.0 .
```

5.

i Note: Make sure you tag the image in the following format:
 dev.local/<image name>:<tag>
 Instead of the latest tag, use a tag such as 1.0.0.

Here is the sample Dockerfile used in the above command:

```
FROM alpine:3.8
RUN apk add --no-cache ca-certificates
WORKDIR /app
ADD <app_executable> /app/flogoapp
RUN chmod a+x /app/flogoapp
ENTRYPOINT ["/app/flogoapp"]
```

The Docker image is built:

```
~/Desktop docker build --file Dockerfile -t dev.local/flogoknative:1.0.0 .
[+] Building 0.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 370B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/alpine:3.8
=> [1/5] FROM docker.io/library/alpine:3.8@sha256:2bb501e6173d9d006e56de5bce2720eb06396803300fe1687b58a7ff32bf4c14
=> [internal] load build context
=> => transferring context: 619B
=> CACHED [2/5] RUN apk add --no-cache ca-certificates
=> CACHED [3/5] WORKDIR /app
=> CACHED [4/5] ADD knative-linux_amd64_9999 /app/flogoapp
=> CACHED [5/5] RUN chmod a+x /app/flogoapp
=> exporting to image
=> exporting layers
=> writing image sha256:ca9a72f11d624155eeb178ae6d3d38255a3418d1d07beed696346d35d75e751f
=> naming to docker.io/library/flogoknative
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```

6. Confirm that the Docker image is built successfully:

```
docker images | grep knative
```

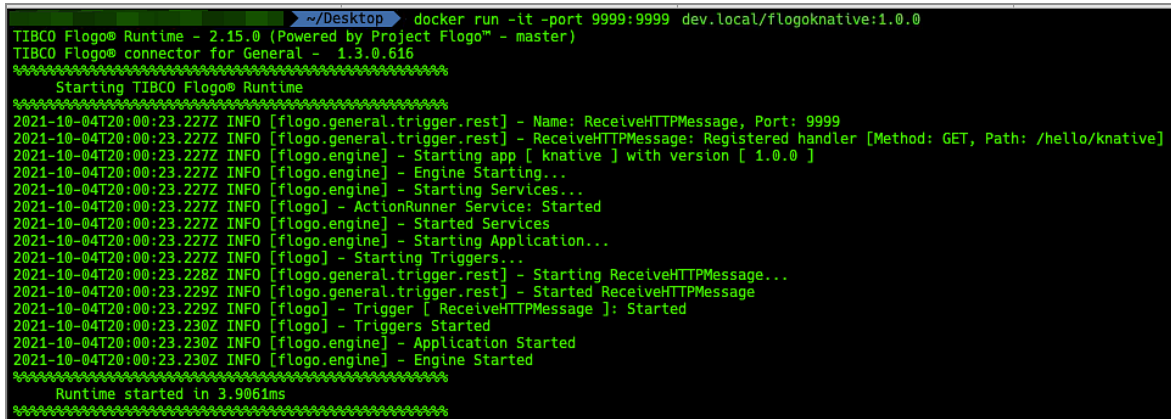
The details of the flogoknative image are displayed:

```
~/Desktop docker images | grep knative
dev.local/flogoknative 1.0.0 833e5670f64b 33 hours ago 52.4MB
dev.local/flogoknative V02 833e5670f64b 33 hours ago 52.4MB
flogoknative latest 833e5670f64b 33 hours ago 52.4MB
dev.local/flogoknative V01 77d6fb92eb00 5 days ago 28.7MB
flogoknative V01 77d6fb92eb00 5 days ago 28.7MB
dev.local/flogoknative latest cbe4bea6b96a 7 days ago 28.7MB
helloknative-go latest 078ead9e3f1a 3 weeks ago 13.5MB
```

7. To test the Docker image:

```
docker run -it -p 9999:9999 dev.local/flogoknative:1.0.0
```

The Flogo runtime logs should be displayed as follows:



```
~/Desktop docker run -it -port 9999:9999 dev.local/flogoknative:1.0.0
TIBCO Flogo® Runtime - 2.15.0 (Powered by Project Flogo™ - master)
TIBCO Flogo® connector for General - 1.3.0.616
*****
Starting TIBCO Flogo® Runtime
*****
2021-10-04T20:00:23.227Z INFO [flogo.general.trigger.rest] - Name: ReceiveHTTPMessage, Port: 9999
2021-10-04T20:00:23.227Z INFO [flogo.general.trigger.rest] - ReceiveHTTPMessage: Registered handler [Method: GET, Path: /hello/knative]
2021-10-04T20:00:23.227Z INFO [flogo.engine] - Starting app [ knative ] with version [ 1.0.0 ]
2021-10-04T20:00:23.227Z INFO [flogo.engine] - Engine Starting...
2021-10-04T20:00:23.227Z INFO [flogo.engine] - Starting Services...
2021-10-04T20:00:23.227Z INFO [flogo] - ActionRunner Service: Started
2021-10-04T20:00:23.227Z INFO [flogo.engine] - Started Services
2021-10-04T20:00:23.227Z INFO [flogo.engine] - Starting Application...
2021-10-04T20:00:23.227Z INFO [flogo] - Starting Triggers...
2021-10-04T20:00:23.228Z INFO [flogo.general.trigger.rest] - Starting ReceiveHTTPMessage...
2021-10-04T20:00:23.229Z INFO [flogo.general.trigger.rest] - Started ReceiveHTTPMessage
2021-10-04T20:00:23.229Z INFO [flogo] - Trigger [ ReceiveHTTPMessage ]: Started
2021-10-04T20:00:23.230Z INFO [flogo] - Triggers Started
2021-10-04T20:00:23.230Z INFO [flogo.engine] - Application Started
2021-10-04T20:00:23.230Z INFO [flogo.engine] - Engine Started
*****
Runtime started in 3.9061ms
*****
```

8. Load the Docker image into the default knative cluster:

```
kind load docker-image dev.local/flogoknative:1.0.0 --name knative
```

The Flogo Docker image is loaded inside the knative cluster and is used by Knative to create the service.

9. Create the Knative service:

```
kn service create helloflogo --image dev.local/flogoknative:1.0.0 -
-port 9999 --revision-name=<any revision name>
```

i Note: The port should be the same as what the Flogo is listening to. In this case, 9999.

A service is created and an URL is generated. You should see messages similar to the following:

```

~/Desktop ~$ kn service create helloflogo --image dev.local/flogoknative:1.0.0 --port 9999 --revision-name=rakshit
Creating service 'helloflogov1' in namespace 'default':
0.026s The Configuration is still working to reflect the latest desired specification.
0.082s The Route is still working to reflect the latest desired specification.
0.119s Configuration "helloflogov1" is waiting for a Revision to become ready.
3.235s ...
3.287s Ingress has not yet been reconciled.
3.364s Waiting for load balancer to be ready
3.564s Ready to serve.

Service 'helloflogov1' created to latest revision 'helloflogov1-rakshit' is available at URL:
http://helloflogov1.default.127.0.0.1.nip.io

```

The networking layer, routes, ingress, and load balancer are configured for the Knative service.

To see a list of services, execute the following command:

```
kn service list
```

NOTE: If you notice errors during any of these steps, the service is not created successfully. For troubleshooting tips, see [Troubleshooting Tips](#).

- Append the REST Trigger endpoint path (specified in step 1) to the generated service URL and hit the endpoint using a browser or curl. For curl, the format of the command is `curl <URL returned in previous step>/hello/knative`. You should see the Flogo return message as the response:

```

~/Desktop ~$ curl -v http://helloflogov1.default.127.0.0.1.nip.io/hello/knative
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to helloflogov1.default.127.0.0.1.nip.io (127.0.0.1) port 80 (#0)
> GET /hello/knative HTTP/1.1
> Host: helloflogov1.default.127.0.0.1.nip.io
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< access-control-allow-origin: *
< content-length: 24
< content-type: text/plain; charset=UTF-8
< date: Mon, 04 Oct 2021 20:14:46 GMT
< x-request-id: 533ac258-f522-4714-8071-6817fe3aaf73
< x-server-instance-id: 254c2d8d960f2d66ffa2667d4c806b1f
< x-envoy-upstream-service-time: 1390
< server: envoy
<
+ Connection #0 to host helloflogov1.default.127.0.0.1.nip.io left intact
hello flogo from knative* closing connection 0

```

Troubleshooting Tips

Error Message	Probable Solution
Errors while creating a kn service: <ul style="list-style-type: none"> configuration does not have any ready revision 	Check whether executable permissions are given to the Flogo app before building the Docker image.

-
- RevisionMissing
-

Error while creating a kn service:
IngressNotConfigured/reconciled

Delete your Knative cluster and recreate it using the following command:

```
kind delete cluster --name knative
```

After deleting the cluster, to reinstall it, follow the steps mentioned in [Getting Started with Knative](#).

Error after creating the kn service and running the `kn service list` command:

RevisionMissing

Make sure you tag the image in the following format:

```
dev.local/<image name>:<tag>
```

Instead of the `latest` tag, use a tag such as `1.0.0`.

After you tag the image, load it in `kind` and then create the service again.

Monitoring

This section contains information about how to monitor your apps.

About the TIBCO Flogo Enterprise Monitoring App

Using the Flogo Enterprise Monitoring app, you can monitor Flogo Enterprise apps that are running in your environment. The Flogo Enterprise Monitoring app collects metrics of flows and triggers from all running apps that are registered with it. In the UI of the app, you can visualize the metrics.

The Flogo Enterprise Monitoring app can also be used with TIBCO Flogo® Flow State Manager to collect information about the state of all run flows of a Flogo app. For more

information on how to use the Flogo Enterprise Monitoring app with TIBCO Flogo® Flow State Manager, see [About TIBCO Flogo® Flow State Manager](#) .

How to Set Up and run the Flogo Enterprise Monitoring App

The Flogo Enterprise Monitoring app is available as a ZIP file. It can run as a standalone app or in a container, such as Docker or Kubernetes. However, you must run the Flogo Enterprise Monitoring app on the same container platform where the Flogo Enterprise apps are running.

How Registration Works in the Flogo Enterprise Monitoring App

Flogo Enterprise apps must be registered with the Flogo Enterprise Monitoring app to be able to view its app metrics. After an app is registered, the Flogo Enterprise Monitoring app can monitor and fetch the instrumentation statistics for the app.

The Flogo Enterprise Monitoring app stores the app registration details in a data store. Currently, the only data store supported is of the type `File`. The app registration details include app name, app host, app instrumentation port, app version, runtime version under which the app is running, and app tags. App tags are custom tags that help you provide additional information about the app. You can set them specific to an app.

Note:

- A Flogo app can have one or more instances and they can be registered with the Flogo Enterprise Monitoring app.
- Each app instance is identified as unique based on the app name and app version.

API Key for Additional Security

For additional security, the Flogo Enterprise Monitoring app can also be started using a secret key called the API key. The API key must be provided while starting the Flogo Enterprise Monitoring app and the same API key must also be provided while starting the Flogo app. The Flogo app registers with the Flogo Enterprise Monitoring app using the API key provided. If an API key is not provided, the Flogo app is not registered with the Flogo Enterprise Monitoring app.

Using the Flogo Enterprise Monitoring App

Using the Flogo Enterprise Monitoring app to monitor Flogo apps involves the following steps:

Procedure

1. Run the Flogo Enterprise Monitoring app. You can run the app in one of two ways:
 - Run the app as a standalone app. See [Running the Flogo Enterprise Monitoring App](#).
 - Run the app in Docker. See [Running the Flogo Enterprise Monitoring App on Docker](#).
2. Register the Flogo app to be monitored using the Flogo Enterprise Monitoring app. See [Registering an App with the Flogo Enterprise Monitoring App](#).
3. Access the UI of the Flogo Enterprise Monitoring app by using a browser and view the statistics of the Flogo apps. See [Viewing Statistics of Apps](#).

Running Flogo Enterprise Monitoring as a Standalone App

You can run the Flogo Enterprise Monitoring app as a standalone app or in a container such as Docker or Kubernetes. This section explains how to run the Flogo Enterprise Monitoring app as a standalone app.

Before you begin

The Flogo Enterprise Monitoring app is installed as described in the "Installing TIBCO Flogo[®] Enterprise Monitoring App" section of *TIBCO Flogo[®] Enterprise Installation*.

Procedure

1. Navigate to the `flogomon/bin` folder.
2. Run `startup.sh` (on macOS or Linux) or `startup.bat` (on Windows).

Result

The web server for the Flogo Enterprise Monitoring app is started.

What to do next

Register the Flogo app to be monitored with the Flogo Enterprise Monitoring app. See [Registering a Flogo App with the Flogo Enterprise Monitoring App](#).

Running the TIBCO Flogo Enterprise Monitoring App On Docker

You can run the Flogo Enterprise Monitoring app as a standalone app or in a container such as Docker or Kubernetes. This section explains how to run the Flogo Enterprise Monitoring app on Docker.

Before you begin

The Flogo Enterprise Monitoring app is installed as described in the "Installing TIBCO Flogo[®] Enterprise Monitoring App" section of the *TIBCO Flogo[®] Enterprise Installation*.

Procedure

1. Navigate to the `flogomon` folder.
2. Build the Docker image by running the `Dockerfile` command or `Dockerfile_alpine` command as follows:

```
docker build -t flogomon -f Dockerfile .
```

```
docker build -t flogomon -f Dockerfile_alpine .
```

3. To get a list of the most recently created Docker images, run:

```
docker images
```

4. Run the Flogo Enterprise Monitoring app using the following commands. For a list of configuration properties that can be used while running these commands, refer to [Configuring the Flogo Enterprise Monitoring App](#).
 - **With persistent volumes and API key:**


```
docker run -e FLOGO_MON_DATA_DIR=<path where applist.json file
must be stored> -e FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port>
-v <path to persistent volumes>:/opt/flogomon/data -e FLOGO_
MON_API_KEY=<secret API key> -it -p 7337:7337 <name of Docker
image of Flogo Enterprise Monitoring application>
```

Here, `-p` specifies the port on which the Flogo Enterprise Monitoring app must be started. The default port is 7337 and it can be configured using the `FLOGO_MON_SERVER_PORT` property.

i Note: Use `-e FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port>` only if you want to use TIBCO Flogo® Flow State Manager.

For example:

```
docker run -e FLOGO_MON_DATA_DIR=/opt/flogomon/data -e FLOGO_
FLOW_SM_ENDPOINT=http://localhost:9091 -v
/home/testuser/flogo:/opt/flogomon/data -it -p 7337:7337
flogomon:latest
```

```
docker run -e FLOGO_MON_DATA_DIR=/opt/flogomon/data -e FLOGO_
FLOW_SM_ENDPOINT=http://192.168.4.12:9091 -v
/home/testuser/flogo:/opt/flogomon/data -it -p 7337:7337
flogomon:latest
```

- **Without persistent volumes and API key:**

```
docker run -e FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port> -it -
p 7337:7337 <name of Docker image of Flogo Enterprise
Monitoring application>
```

Here, `-p` specifies the port on which the Flogo Enterprise Monitoring app must be started. The default port is 7337 and it can be configured using the `FLOGO_MON_SERVER_PORT` property.

i Note: Use `-e FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port>` only if you want to use the TIBCO Flogo® Flow State Manager.

For example:

```
docker run -e FLOGO_FLOW_SM_ENDPOINT=http://localhost:9091 -it  
-p 7337:7337 flogomon:latest
```

```
docker run -e FLOGO_FLOW_SM_ENDPOINT=http://192.168.4.12:9091 -  
it -p 7337:7337 flogomon:latest
```

Result

The web server for the Flogo Enterprise Monitoring app is started.

What to do next

Register the app to be monitored with the Flogo Enterprise Monitoring app. See [Registering an App with the Flogo Enterprise Monitoring App](#).

Running the Flogo Enterprise Monitoring Application On Kubernetes

You can run the Flogo Enterprise Monitoring app as a standalone app or as a container on Kubernetes. This section explains how to run the Flogo Enterprise Monitoring app on Kubernetes.

When the Flogo Enterprise Monitoring app is started on Kubernetes, it monitors Flogo apps added to a Kubernetes cluster. If a Flogo app is found, the app is registered with the Flogo Enterprise Monitoring app. The YAML file of the app must include some configuration details required for registering the app with the Flogo Enterprise Monitoring app. For details, refer to [Configurations in the Flogo App's YAML File](#). After a Flogo app is registered, the Flogo Enterprise Monitoring app is available in the App List on the Summary page.

Before you begin

The Flogo Enterprise Monitoring app is installed as described in the "*Installing TIBCO Flogo® Enterprise Monitoring App*" section of the *TIBCO Flogo® Enterprise Installation*.

An overview of the procedure is given below:

Procedure

1. [Grant Access Using ClusterRole.](#)
2. [Configure the ServiceAccount.](#)
3. [Link the ServiceAccount to the ClusterRole using ClusterRoleBinding.](#)
4. [Link the Flogo App to the Flogo Enterprise Monitoring Application.](#)
5. [Specify configurations in the Flogo App's YAML File.](#)

Granting Access Using ClusterRole

To monitor pods registered in the Kubernetes cluster, the Flogo Enterprise Monitoring app requires access to the List, Watch, and Get verbs for all pods across all namespaces. To grant access, and update the YAML file as shown in the following sample file.

Sample YAML file showing ClusterRole

Cluster

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: flogo-mon-cluster-role
rules:
- apiGroups: ["*"]
  resources: ["pods"]
  verbs: ["list","get","watch"]
```

Configuring the Service Account

Configure a service account for a pod as shown in the following sample YAML file.

Sample YAML file showing ServiceAccount

Service

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
name: flogo-mon-service-account
```

Linking the ServiceAccount to the ClusterRole

Link the ServiceAccount to the ClusterRole using ClusterRoleBinding as shown in the following sample YAML file.

Sample YAML file to add a ClusterRoleBinding

Deployment

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: flogo-mon-service
subjects:
- kind: ServiceAccount
  name: flogo-mon-service-account
  namespace: default
roleRef:
  kind: ClusterRole
  name: flogo-mon-cluster-role
  apiGroup: rbac.authorization.k8s.io
```

Linking the Flogo App to the Flogo Enterprise Monitoring Application

1. Link the Flogo Monitoring Deployment to the service account created in the previous steps. See the sample deployment YAML.
2. In the Flogo Monitoring Deployment YAML, provide the FLOGO_APP_SELECTOR label with a value as a key-value pair. For example, appType=flogo.



Note: All Flogo apps which are required to be linked with the Flogo Enterprise Monitoring app must specify this label.

Sample YAML file for Deployment

Deployment

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flogo-mon-service
spec:
  selector:
    matchLabels:
      app: flogo-mon-service
  replicas: 1
  template:
    metadata:
      labels:
        app: flogo-mon-service
    spec:
      containers:
        - name: flogo-mon-service
          image: flogomon:v1
          imagePullPolicy: Never
          env:
            - name: "FLOGO_APP_SELECTOR"
              value: "appType=flogo"
          ports:
            - containerPort: 7337
      serviceAccountName: flogo-mon-service-account
```

Configurations in the Flogo App's YAML File

To register an app with the Flogo Enterprise Monitoring app, provide the following configuration details in the app's YAML file.

- **Labels:** The deployment must have a label with the same value provided in the FLOGO_APP_SELECTOR environment variable. For example, if FLOGO_APP_SELECTOR has the value as appType=flogo, the Flogo app must have a label with the key as appType and Name as flogo. The Flogo Enterprise Monitoring app attempts to register the

app with this label only. If the label is not provided, the app is ignored.

- **Annotations:** The following annotations are mandatory:
 - **app.tibco.com/metrics:** Setting this annotation to `true` registers the app with the Flogo Enterprise Monitoring app and enables the metrics collection on the app. Setting the annotation to `false` deregisters it from the Flogo Enterprise Monitoring app and turns off the metrics collection.
 - **app.tibco.com/metrics-port:** Provide the HTTP port for the app. This port must be the same as the one specified by the `FLOGO_HTTP_SERVICE_PORT` environment variable. If an invalid value is set, the app is ignored.

Sample YAML File

App



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```

apiVersion: v1
kind: Service
metadata:
  name: flogoapp
  labels:
    app: flogoapp
spec:
  type: LoadBalancer
  ports:
    - port: 9999
      protocol: TCP
      name: appport
      targetPort: 9999
  selector:
    app: flogoapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flogoapp
spec:
  selector:
    matchLabels:

```

```

    app: flogoapp
  replicas: 2
  template:
    metadata:
      labels:
        app: flogoapp
        appType: flogo
      annotations:
        app.tibco.com/metrics: 'true'
        app.tibco.com/metrics-port: '7777'
    spec:
      containers:
        - name: flogoapp
          image: flogoapp:v1
          imagePullPolicy: Never
          ports:
            - containerPort: 9999
            - containerPort: 7777
          env:
            - name: "FLOGO_HTTP_SERVICE_PORT"
              value: "7777"

```

Configuring the Flogo Enterprise Monitoring App

The following properties can be set when running the Flogo Enterprise Monitoring app as described in [Running the Flogo Enterprise Monitoring App](#).

i Note: These properties can also be set in the `flogomon/config/config.env` file. If you have set the properties while starting the app, the values in the `config.env` file are ignored, and the values specified during the startup take precedence.

Property	Description
FLOGO_MON_DATA_DIR	The Flogo Enterprise Monitoring app uses a file-based data store. This property provides the folder where the <code>applist.json</code> file must be stored. If you run a Docker app with persistent volumes, the <code>applist.json</code> is created at the location specified as persistent volume.

Property	Description
FLOGO_MON_RETRY_INTERVAL	<p>The interval (in seconds) after which the Flogo Enterprise Monitoring app retries to ping all instances of the Flogo app registered with the Flogo Enterprise Monitoring app. For example, if an app is down or the network is slow, the Flogo Enterprise Monitoring app tries to collect monitoring data after the value specified in this property.</p> <p>Default value: 30s</p>
FLOGO_MON_RETRY_COUNT	<p>Number of times the Flogo Enterprise Monitoring app retries to ping all the instances before removing the instance from the datastore.</p> <p>For example, if an app is down or the network is slow, the Flogo Enterprise Monitoring app tries to collect monitoring data the number of times specified in this property.</p> <p>Default value: 5</p>
FLOGO_MON_API_KEY	<p>The API Key that is used by the Flogo app to register with the Flogo Enterprise Monitoring app. The API key must be provided when starting the Flogo Enterprise Monitoring app and the same API key must also be provided when starting the app. The app registers with the Flogo Enterprise Monitoring app using the API key provided. If an API key is not provided, the app is not registered with the Flogo Enterprise Monitoring app.</p> <p>Default value: Blank</p>
FLOGO_MON_SERVER_PORT	<p>The port on which the Flogo Enterprise Monitoring app must be started.</p> <p>Default value: 7337</p>
FLOGO_MON_LOG_LEVEL	<p>The log level for the Flogo app.</p> <p>Default value: INFO</p>

Property	Description
Properties related to TIBCO Flogo® Flow State Manager	
FLOGO_FLOW_SM_ENDPOINT	<p>The endpoint of Flogo Flow State Manager. The format to set the property is:</p> <pre>FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port></pre> <p>For example:</p> <pre>FLOGO_FLOW_SM_ENDPOINT=http://localhost:9091</pre>
	<p>Note: This property needs to be set when starting the app binary after Flogo Flow State Manager is up and running.</p>

Registering a Flogo App with the Flogo Enterprise Monitoring App

After a Flogo app is registered with the Flogo Enterprise Monitoring app, the collection of instrumentation statistics starts automatically. To register a Flogo app with the Flogo Enterprise Monitoring app, start the app with the following properties:

- `FLOGO_HTTP_SERVICE_PORT=<instrumentation port>`: This property specifies the port required to enable the app instrumentation.
- `FLOGO_APP_MON_SERVICE_CONFIG`: This property specifies details of the Flogo Enterprise Monitoring app to the Flogo app.

```
FLOGO_APP_MON_SERVICE_CONFIG={"host":"<Host of Flogo Enterprise Monitoring app>","port":"<Port of Flogo Enterprise Monitoring app>","tags":["<Tag 1>","<Tag 2>"],"apiKey":"<API Key>"}
```

Option	Description
Host	Host of the Flogo Enterprise Monitoring app.
Port	Port of the Flogo Enterprise Monitoring app.

Option	Description
Tags (Optional)	Custom tags that help you provide additional information about the Flogo app; you can set them specific to an app. For example, you can specify whether it is a REST app or whether it is running in Kubernetes, and so on.
apiKey (Optional)	For additional security, the Flogo Enterprise Monitoring app can also be started using a secret key called API key. The API key must be provided while starting the Flogo Enterprise Monitoring app and the same API key must also be provided while starting the Flogo app. The app registers with the Flogo Enterprise Monitoring app using the API key provided. If an API key is not provided, the app is not registered with the Flogo Enterprise Monitoring app.

Examples

- If the Flogo Enterprise Monitoring app is running on `localhost` on port 7337 and the app instrumentation port is 7777, start the Flogo app as:

```
$FLOGO_HTTP_SERVICE_PORT=7777 FLOGO_APP_MON_SERVICE_CONFIG="{\"host\": \"localhost\", \"port\": \"7337\"}" ./App1
```

- If the Flogo Enterprise Monitoring app is running on `localhost` on port 7337, the app instrumentation port is 7777, and you want to start the Flogo Enterprise Monitoring app based on an API Key `APIkey1`, start the app as:

```
$FLOGO_HTTP_SERVICE_PORT=7777 FLOGO_APP_MON_SERVICE_CONFIG="{\"host\": \"localhost\", \"port\": \"7337\", \"apiKey\": \"<value specified when starting the Flogo Enterprise Monitoring app>\"}" ./app_linux_amd64
```

- If the Flogo Enterprise Monitoring app is running on `localhost` on port 7337, the app instrumentation port is 7777, and you want to provide additional tags (named `onpremise` and `testing`), start the app as:

```
$FLOGO_HTTP_SERVICE_PORT=7777 FLOGO_APP_MON_SERVICE_CONFIG="{\"host\": \"localhost\", \"port\": \"7337\", \"tags\": {\"onpremise\": true, \"testing\": true}}\"}" ./App1
```

```
{\"host\": \"localhost\", \"port\": \"7337\", \"tags\": [\"onpremise\", \"testing\"]}\" ./App1
```

- On Microsoft Windows, if the Flogo Enterprise Monitoring app is running on localhost on port 3000 and the app instrumentation port is 7775, start the app as:

```
set FLOGO_HTTP_SERVICE_PORT=7775
set FLOGO_APP_MON_SERVICE_CONFIG=
{\"host\": \"localhost\", \"port\": \"3000\", \"appHost\": \"instance1\"}
flogo-windows_amd64.exe
```

- On Linux and Mac, if the Flogo Enterprise Monitoring app is running on localhost on port 7337, the app instrumentation port is 7777, start the app as:

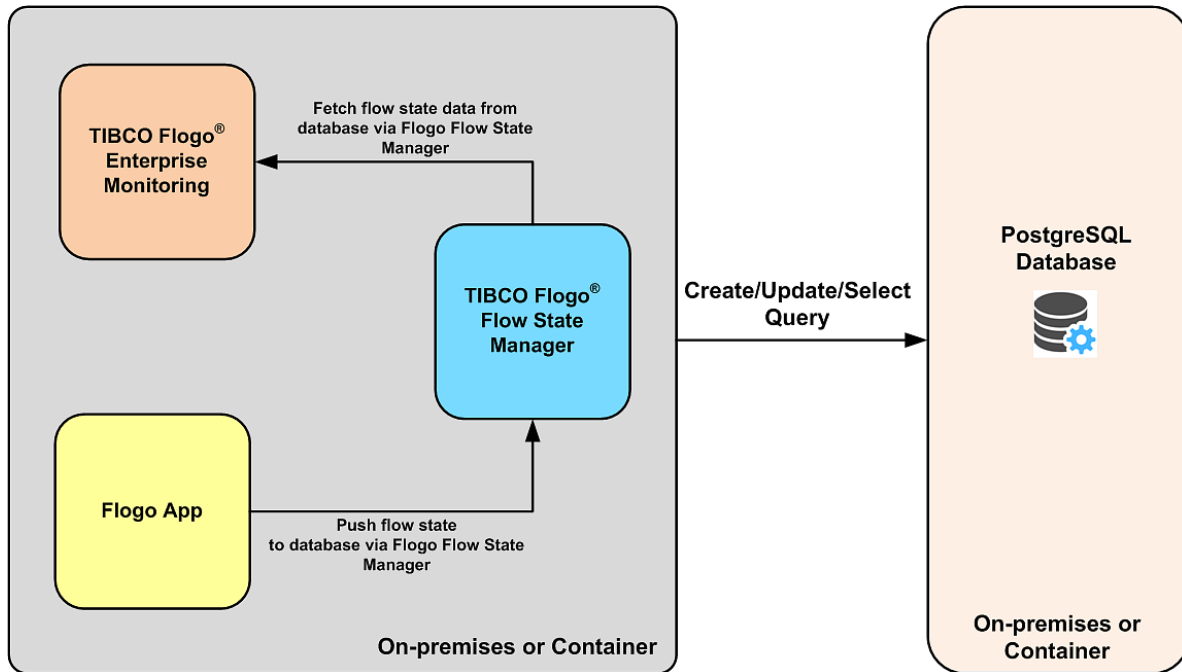
```
$FLOGO_HTTP_SERVICE_PORT=7777 FLOGO_APP_MON_SERVICE_CONFIG="
{\"host\": \"localhost\", \"port\": \"7337\", \"apiKey\": \"<value
specified when starting the Flogo Enterprise Monitoring app>\"}"
./app_linux_amd64
```

What to do next: View the statistics of the app on the UI of the Flogo Enterprise Monitoring app. See [Viewing Statistics of Apps](#).

About TIBCO Flogo® Flow State Manager

Using Flogo® Flow State Manager and the TIBCO Flogo® Enterprise Monitoring App, you can collect information about the state of all executed flows of a Flogo app.

Flogo Flow State Manager acts as an interface between a Flogo app and the TIBCO Flogo® Enterprise Monitoring application. It collects data from a Flogo app and then persists the collected data to a supported database (currently, PostgreSQL). When it receives a request from the TIBCO Flogo® Enterprise Monitoring application, Flogo Flow State Manager collects data from the database and passes it on to the TIBCO Flogo® Enterprise Monitoring application for displaying on the UI.



Flogo Flow State Manager is available as a compressed file. For more information about installing Flogo Flow State Manager, see *TIBCO Flogo® Enterprise Installation*.

For more information about TIBCO Flogo® Enterprise Monitoring App, see [About the TIBCO Flogo Enterprise Monitoring App](#).

Using Flogo Flow State Manager

Before you begin

Make sure you meet the following requirements:

- Install the PostgreSQL database. For more information, see [PostgreSQL](#).
- Optionally, download and install a PostgreSQL management tool such as PGAdmin. For more information, see [PGAdmin](#).

Using Flogo Flow State Manager involves the following steps:

Procedure

1. Configure the PostgreSQL database as described in [Using Flogo Flow State Manager](#).
2. Run Flogo Flow State Manager. You can run the app in one of two ways:

- Run the app as a standalone app. See [Running Flogo Flow State Manager as a Standalone App](#).
 - Run the app in Docker. See [Running Flogo Flow State Manager on Docker](#).
3. Start the Flogo Enterprise Monitoring app by specifying the host and port of the Flogo Flow State Manager. See [Starting Flogo Enterprise Monitoring with Details of Flogo Flow State Manager](#).
 4. Start the Flogo app binary. Information about the state of all executed flows of a Flogo app is displayed on the [Executions Page](#).

Configuring the PostgreSQL Database

All execution data from the Flogo app is stored in the PostgreSQL database. Set up the PostgreSQL database for accepting data from the Flogo app as follows:

Procedure

1. Start the PostgreSQL service as docker container. For example:

```
docker run -d --name my_postgres -v my_dbdata1:/var/lib/postgresql/data -p 54320:5432 -e POSTGRES_PASSWORD=<password> -e POSTGRES_USER=<user> postgres
```

2. Start the PGAdmin portal as a Docker container:

```
docker run -p 9990:80 -e PGADMIN_DEFAULT_EMAIL=<email address> -e PGADMIN_DEFAULT_PASSWORD=<pgadmin_password> -d dpage/pgadmin4
```

3. Configure the PostgreSQL server in the PGAdmin admin portal with the following details. Note that you must use the same parameter values while configuring config.json for Flogo Flow State Manager.

- Host: IP of the local machine
- PORT: 54320 (same host and port used while starting PostgreSQL service as docker container)
- User: <user> (configured while starting PostgreSQL server)
- Password: <password> (configured while starting PostgreSQL server)
- Maintenance database: same as <user> (if not specifically mentioned while starting PostgreSQL server)

4. Create the steps table by using <flogo_flow_state_

```
manager.tar>\config\postgres\steps.sql.
```

i Note: If you are running the `steps.sql` script in a terminal, convert the script content to a single continuous line.

5. Create the `flowstate` table by using `<flogo_flow_state_manager.tar>\config\postgres\flowstate.sql`.

6.

i Note: If you are running the `flowstate.sql` script in a terminal, convert the script content to a single continuous line.

Running Flogo Flow State Manager as a Standalone App

Procedure

1. Start Flogo Flow State Manager by executing the binary for your operating system:
 - `flowstatemanager-windows_amd64` (Windows executable)
 - `flowstatemanager-linux_amd64` (Linux executable)
 - `flowstatemanager-darwin_amd64` (Mac executable)
2. Copy the `<flogo_flow_state_manager.tar>\config\postgres\config.json` into the `bin` directory. If the `config.json` file exists in any other directory, you can also set the `FLOW_STATE_CONFIG` environment variable to point to the location as follows:

```
FLOW_STATE_CONFIG=<file path>
```

3. Update the values in `config.json` as follows:

⚠ Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "exposeRecorder": true,
```

```

"port": "<The port on which you want to start the flow state manager
binary>",
"persistence": {
"type": "postgres",
"name": "pg-server-1",
"description": "",
"host": "<The IP address where Postgres is running>",
"port": "<port on which the Postgres database server is running>",
"databaseName": "postgres",
"user": "<user value configured while starting PostgreSQL server>",
"password": "<password value configured while starting PostgreSQL server>",
"Maintenance database": <same as <user>, if not specifically
mentioned while starting postgresQL>
"maxopenconnection": "0",
"maxidleconnection": "2",
"conmaxlifetime": "0",
"maxconnectattempts": "3",
"connectionretrydelay": "5",
"tlsparam": "VerifyCA",
"cacert": "",
"clientcert": "",
"clientkey": ""
}
}

```

Running Flogo Flow State Manager on Docker

To run the Flogo Flow State Manager in a Docker container:

Procedure

1. Update <flogo_flow_state_manager.tar>\config\postgres\config.json as per your Postgres installation.

! **Important:** Postgres is not accessible over 'localhost' when Flogo Flow State Manager is running on Docker. You must use the machine's IP address.

2. Go to the root folder (packaging) and run:

```
docker build -t flogostatemanager:1.0.0 -f ./deployments/Dockerfile
.
```

3. Start the Flogo Flow State Manager service by mounting a volume for config.json:

```
docker run -p 8099:8099 -v <parent>
absolutePath>/flowstatemanager/packaging/config/postgres/config.jso
n:/opt/flogo/sm/config.json flogostatemanager:1.0.0
```

Running Flogo Flow State Manager on Kubernetes

Procedure

1. Update <flogo_flow_state_manager.tar>\config\postgres\config.json as per your Postgres installation.

! **Important:** Postgres is not accessible over 'localhost' when Flogo Flow State Manager is running on Docker. You must use the machine's IP address.

2. Go to the root folder (packaging) and run:

```
docker build -t flogostatemanager:1.0.0 -f ./deployments/Dockerfile
.
```

3. Push the Flogo Flow State Manager Docker image to the Docker registry.
4. Update the <flogo_flow_state_manager.tar>/deployments/k8s/deployment.yml as per the required configuration. For example, image name, version, port values, and so on.
5. Deploy the Flogo Flow State Manager service in the Kubernetes cluster:


```
<flogo_flow_state_manager.tar>/deployments/k8s/deploy.sh
```

This command creates the required configmap and applies the `deployment.yml` configuration to define the deployment and service component for Kubernetes.

- To undeploy the Flow State Manager service in k8s cluster:

```
<flogo_flow_state_manager.tar>/deployments/k8s/undeploy.sh
```

Configuring Flogo Flow State Manager

Property	Description
FLOGO_FLOW_SM_ENDPOINT	<p>The endpoint of Flogo Flow State Manager. The format to set the property is:</p> <pre>FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port></pre> <p>For example:</p> <pre>FLOGO_FLOW_SM_ENDPOINT=http://localhost:9091</pre> <p>Note: This property needs to be set when starting the app binary after Flogo Flow State Manager is up and running. It also needs to be set when running the Flogo Enterprise Monitoring app.</p>
FLOGO_FLOW_STATE_ASYNC_INVOCATION	<p>Specifies whether the Flogo Flow State Manager must be invoked asynchronously or not. Enabling the property also helps to increase the throughput of the app. The format to set the property is:</p> <pre>FLOGO_FLOW_STATE_ASYNC_INVOCATION=true</pre> <p>Note: This property needs to be set when starting the app binary after Flogo Flow State Manager is up and running.</p> <p>Default value: false</p>

Starting Flogo Enterprise Monitoring with Details of Flogo Flow State Manager

Start Flogo Enterprise Monitoring with the host and port details of the Flogo Flow State Manager:

Procedure

Start the TIBCO Flogo® Enterprise Monitoring app. When starting the app, use the `FLOGO_FLOW_SM_ENDPOINT` environment variable to specify the host and port of the Flogo Flow State Manager. For example:

```
docker run -it -e FLOGO_MON_DATA_DIR=/opt/flogomon/data -e FLOGO_FLOW_SM_ENDPOINT=http://<host>:<port> -v ~/temp:/opt/flogomon/data -p 7337:7337 <fe-mon docker image name>
```

Procedure

Check the console log to verify that a successful connection has been established with Flogo Flow State Manager.

If you notice a connection error in the log, verify whether the Flogo Flow State Manager is running and the host/port details are configured correctly.

Starting the App Binary

Start the app binary after Flogo Flow State Manager is up and running.

```
export FLOGO_FLOW_SM_ENDPOINT=http://localhost:9091
FLOGO_HTTP_SERVICE_PORT=7777
FLOGO_APP_MON_SERVICE_CONFIG="
{"host\":"<IP address>", "port\":"<port>"}"
./app-binary
```

Result

Information about the state of all executed flows of a Flogo app is displayed on the [Executions Page](#).

Viewing Statistics by Using Flogo Enterprise Monitoring app

Before you begin

- The Flogo Enterprise Monitoring app must be running. See [Running the Flogo Enterprise Monitoring App](#) or [Running the Flogo Enterprise Monitoring App on Docker](#).
- The Flogo app to be monitored must be registered with the Flogo Enterprise Monitoring app. See [Registering a Flogo App with the Flogo Enterprise Monitoring App](#).

Procedure

1. In the UI of the Flogo Enterprise Monitoring app, go to the following URL to monitor the app:

`http://<URL of Flogo Enterprise Monitoring app>:<port of Flogo Enterprise Monitoring app>`

For example:

`http://localhost:7337`

The **Apps** page is displayed as shown below:

Name	Version	Flogo Version	Instances	Tags
HTTPService1	1.1.0	2.9.0	2	FE,RESTApp
HTTPService1	1.1.1	2.9.0	1	FE,RESTApp


The **Apps** page shows all the Flogo apps registered with the Flogo Enterprise Monitoring app. For details, see [Apps Page](#).

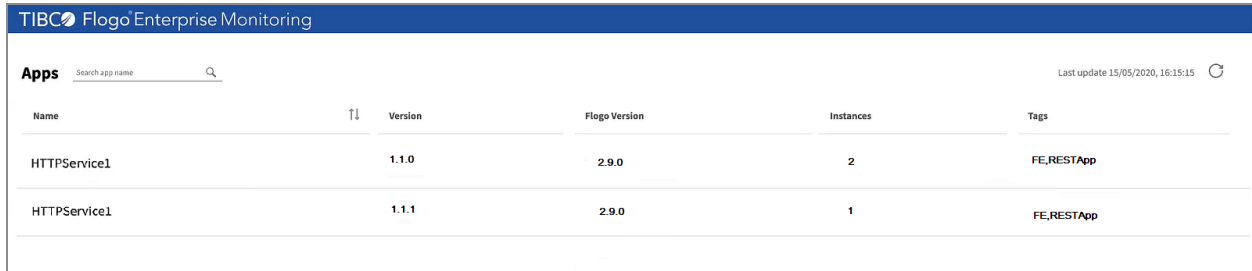
2. Click an app name.

The **Metrics** page is displayed. The instrumentation statistics are displayed in two tabs - Flow and Triggers. For details, see [Metrics Page](#).

Apps Page

The **Apps** page shows all the Flogo apps registered with the Flogo Enterprise Monitoring app.

Note: The apps list on this page is not refreshed automatically. Click  to refresh the list manually.



Name	Version	Flogo Version	Instances	Tags
HTTPService1	1.1.0	2.9.0	2	FE,RESTApp
HTTPService1	1.1.1	2.9.0	1	FE,RESTApp

For each running app, you can view the following details:

Item	Description
Name	Name of the app. Click the name of an app to get more details of the app. For example, in the above screenshot, you can click HTTPService1 to get more details about the service. The details of HTTPService1 are displayed on the Metrics page.
Version	Version of the app.
Flogo Version	The version in which the app was created.
Instances	Number of instances registered per app.
Tags	Tags of the app. These tags help you provide additional information about the app. For example, you can specify whether it is a REST app or whether it is running in Kubernetes, and so on.

From the **Apps** page, you can also:

- Click the heading in the list to sort the apps. For example, to sort the list by name,

click the heading **Name**. Click the same heading again to toggle between the ascending or descending order of listing the apps.

- The Search control above the list enables you to find apps by name.

Metrics Page

The Metrics page displays the instrumentation statistics of flows and triggers.

Select an instance ID from the instance ID list in the upper-left corner of the page. The instrumentation statistics of flows and triggers are displayed on the **Triggers** tab and **Flow** tab.

Triggers Tab

The screenshot displays the TIBCO Flogo Enterprise Monitoring interface for the Triggers tab. At the top, it shows the application name 'Rest-app 1.1.0' and the instance ID 'rgoyal'. The 'Triggers' tab is selected, and a search bar contains 'f1'. Below the search bar, there is a summary of 'Total Trigger Executions (count)' showing 1 Started, 1 Completed, and 0 Faulted. A table titled 'Handler Execution (count)' provides a detailed view of the trigger execution. The table has columns for Handler, Config, Started, Completed, and Faulted. The row for 'f1' shows a method of GET and path of /path, with 1 Started, 1 Completed, and 0 Faulted.

Handler	Config	Started	Completed	Faulted
f1	method: GET path: /path	1	1	0

Select a trigger from the list on the left to see its details. You can also search for a trigger in the list.

The following information is displayed on the right:

Name	Description
------	-------------

Total Trigger Executions (count)	
---	--

Name	Description
Started	Total number of trigger instances started
Completed	Total number of trigger instances completed
Faulted	Total number of trigger instances failed

Handler Execution (count)	
Handler	Name of the trigger handler
Config	Configuration of the trigger handler. For example:
	<pre>method: POST path: /arrayfilter</pre>
Started	Total number of trigger handlers started
Completed	Total number of trigger handlers completed
Faulted	Total number of trigger handlers failed

Flow Tab

The screenshot displays the TIBCO Flogo Enterprise Monitoring interface. The top navigation bar shows 'TIBCO Flogo Enterprise Monitoring' and 'Rest-app 1.1.0'. The main content area is titled 'Flow' and shows the following data:

- Flow Instances (count):** 1 Started, 1 Completed, 0 Faulted.
- Flow Execution Time (ms):** 0.927 Average, 0.927 Maximum, 0.927 Minimum.
- Activity Execution (count):**

Activity Name	Started	Completed	Faulted
Return	1	1	0
log1	1	1	0
log2	1	1	0
log3	1	1	0
- Activity Execution Time (ms):**

Activity Name	Average	Maximum	Minimum
Return	0.102	0.102	0.102
log1	0.182	0.182	0.182
log2	0.281	0.281	0.281
log3	0.125	0.125	0.125

Select a flow from the list on the left to see its details. You can also search for a flow in the list.

The following information is displayed in the work area on the right:

Name	Description
Flow Instances (count)	
Started	Total number of flow instances started
Completed	Total number of flow instances completed
Faulted	Total number of flow instances failed
Flow Execution Time (in milliseconds)	
<p>Note: The Flow Execution Time (ms) for a faulted flow is always displayed as 0, even if the activities within the flow took time to execute.</p>	
Average	Average execution time of the flow for successful executions
Maximum	Maximum execution time for the flow
Minimum	Minimum execution time for the flow
Activity Execution (count)	
<p>Note: If an Activity is rerun, Activity Execution (count) also includes the rerun counts. You can find out whether an Activity has been rerun through the difference in the trigger and flow metric counts.</p>	
Activity Name	Name of Activity
Started	Total number of times a given Activity has started
Completed	Total number of times a given Activity has

Name	Description
	been completed
Faulted	Total number of times a given Activity has failed
Activity Execution Time (in milliseconds)	
Activity Name	Name of Activity
Average	Average execution time of a given Activity for successful executions
Maximum	Maximum execution time for a given Activity
Minimum	Minimum execution time for a given Activity

Executions Page

The **Executions** page displays information about the state of all run flows of a Flogo app. Details of a trigger are not captured.


The screenshot displays the 'Executions' page in the TIBCO Flogo Enterprise Monitoring interface. The page title is 'Rest-app 1.1.0' and it shows 'Metrics' and 'Executions' tabs. A 'Refresh' button is located in the top right corner. Below the tabs, there is a 'Persist Execution Data' toggle and a search bar. The main content is a table with the following columns: Status, Flow Name, Execution ID, App Instance ID, Duration (ms), Start Time (UTC), and End Time (UTC). The table contains 13 rows of execution records, all with a 'Completed' status. A red box highlights the 'View Details' link next to the first row. The page footer shows 'Page 1 of 3'.

Status	Flow Name	Execution ID	App Instance ID	Duration (ms)	Start Time (UTC)	End Time (UTC)
Completed	f1	9d864b7bc010edc2b72697b0c81b4416	rgoyal	13.289	2021-11-12T07:46:05.581503Z	2021-11-12T07:46:05.594792Z
Completed	f1	44320f5484864781184a35dab167a6b9	rgoyal	8.760	2021-11-10T08:46:57.147801Z	2021-11-10T08:46:57.156562Z
Completed	f1	77116833db4fcccfc2ace21f54d8213	rgoyal	19.711	2021-11-10T08:26:14.868206Z	2021-11-10T08:26:14.887912Z
Completed	f1	ffe707680e682aceda96e889c7b4780f	rgoyal	6.603	2021-11-09T16:37:37.459004Z	2021-11-09T16:37:37.465607Z
Completed	f1	217c9d0ae6ee0fcb96a59dc30b429f4c	rgoyal	10.348	2021-11-09T16:22:16.55385Z	2021-11-09T16:22:16.564198Z
Completed	f1	227c9d0ae6ee0fcb96a59dc30b429f4c	rgoyal	9.994	2021-11-09T15:56:39.891159Z	2021-11-09T15:56:39.901153Z
Completed	f1	9b3e9e96b8503dacf58523746dc2f60	b8bede2b5010	2.904	2021-11-02T09:53:56.435609Z	2021-11-02T09:53:56.438513Z
Completed	f1	fa583eda63b75e6d472d75379fe525	rgoyal	4.330	2021-11-02T06:33:05.330154Z	2021-11-02T06:33:05.334484Z
Completed	f1	7d62e1f6d4b1334f190786ecd7a7e64f	rgoyal	3.798	2021-11-02T05:25:43.645723Z	2021-11-02T05:25:43.649521Z
Completed	f1	7a62e1f6d4b1334f190786ecd7a7e64f	rgoyal	3.986	2021-11-02T05:25:25.711363Z	2021-11-02T05:25:25.715349Z
Completed	f1	7b62e1f6d4b1334f190786ecd7a7e64f	rgoyal	3.969	2021-11-02T05:24:28.579201Z	2021-11-02T05:24:28.58317Z
Completed	f1	7862e1f6d4b1334f190786ecd7a7e64f	rgoyal	4.635	2021-11-02T05:20:28.403274Z	2021-11-02T05:20:28.407909Z

From this page, you can:

- **Persist execution data:** Select **Persist Execution Data** to persist execution data to the supported database (currently, PostgreSQL).

i Note: If **Persist execution data** is disabled, any new execution data is not saved to the database. The **Rerun flow from this Activity** feature is also disabled for all flow executions.

- **Filter based on app version:** You can use the filter to choose the app version for which the data must be displayed.
- **Filter based on time frame:** Use the **All** drop-down to filter based on time frame. For example: in the last 1 hour, last week, last 30 days, and so on.
- **Filter based on flow:** If you have multiple flows, use the **All Flows** drop-down to filter based on flows.
- **Filter based on status:** Use the **All Statuses** drop-down to filter based on the status of the flow.
- **Refresh data:** Use  to refresh the data in the table.
- **View execution data:** The following data is displayed in a tabular format.

Name	Description
Status	Status
Flow Name	Name of the flow.
Execution ID	Instance ID of the flow.
App Instance ID	Instance ID of the app.
Duration (ms)	Duration for which the flow was running.
Start Time (UTC)	Time when the flow was started, based on Coordinated Universal Time (UTC).
End Time (UTC)	Time when the flow ended, based on Coordinated Universal Time (UTC).

- **View details of a flow:** For each flow, you can view its details by clicking **View Details**. A list of activities executed is displayed:

Activity Name	Flow Name	Status	Created On
log1	f1	Completed	2021-11-12T07:46:05.584653Z
log2	f1	Completed	2021-11-12T07:46:05.587024Z
log3	f1	Completed	2021-11-12T07:46:05.589192Z
Return	f1	Completed	2021-11-12T07:46:05.591695Z

- **Rerun the flow from a specific Activity:** You can rerun the flow from a specific Activity. You cannot modify the input data; you can only rerun the Activity.

Note: If you rerun an Activity, the previous execution record for the Activity is overwritten in the database. Past execution records of the Activity that was rerun and all subsequent activities in the flow are deleted.

Important: Exercise caution while re-running a flow attached to the App Startup Trigger and App Shutdown Trigger. These triggers, typically, include logic for creating data or cleaning up data. Such flows might impact the running instances of the app.

To rerun the flow from a specific Activity:

Procedure

1. On the **View Details** page, click the expand icon and then click **Input & Output Data**.

Activities List -bd864b7be010ede2bf2607b0e81b44f6 -f1

Last update 12/11/2021, 01:41:06

Activity Name	Flow Name	Status	Created On
log1	f1	Completed	2021-11-12T07:46:05.584653Z
+ Input & Output Data			
log2	f1	Completed	2021-11-12T07:46:05.587024Z
log3	f1	Completed	2021-11-12T07:46:05.589192Z
Return	f1	Completed	2021-11-12T07:46:05.591695Z

Close

The input and output for the selected Activity are displayed.

Activities List -bd864b7be010ede2bf2607b0e81b44f6 -f1

Last update 12/11/2021, 01:41:06

Activity Name	Flow Name	Status	Created On
log1	f1	Completed	2021-11-12T07:46:05.584653Z
+ Input & Output Data			
Inputs		Outputs	
<pre>{ "Log Level": "INFO", "flowInfo": false, "message": "hello world" }</pre>			
Rerun flow from this activity			
log2	f1	Completed	2021-11-12T07:46:05.587024Z
log3	f1	Completed	2021-11-12T07:46:05.589192Z
Return	f1	Completed	2021-11-12T07:46:05.591695Z



Close

2. Click **Rerun flow from this Activity**.



Note:

- **Rerun flow from this Activity** disappears for activities that are a part of subflows. **Rerun flow from this Activity** also disappears if **Persist Execution Data** is disabled.
- If the version of the app running instance is not same as that of the selected version, you cannot rerun the activities.

After the rerun of the activity, the rerun is indicated by . The **Executions** page is also updated with the latest data. Click  to refresh the changes on the **Executions** page.

App Metrics

For REST APIs, the following methods can be used to enable and disable app metrics at runtime.

Method	Description	Status Code
POST /app/metrics	Enable instrumentation metrics collection	200 - If successfully enabled 409 - If the metrics collection is already enabled
DELETE /app/metrics	Disable metrics collection	200 - If successfully disabled 404 - If metrics collection is not enabled
GET /app/metrics/flows	Retrieve metrics for all flows	200 - Successfully returned metrics data 404 - If the metrics collection is not enabled 500 - If there is an issue when returning metrics data
GET /app/metrics/flow/<flowname>	Retrieve metrics for a given flow	200 - Successfully returned metrics data 400 - If the flow name is incorrect 404 - If the metrics collection is not enabled 500 - If there is an issue

Method	Description	Status Code
		returning metrics data
GET /app/metrics/flow/ <flowname>/activities	Retrieve metrics for all activities in a given flow	200 - Successfully returned metrics data 400 - If the flow name is incorrect 404 - If the metrics collection is not enabled 500 - If there is an issue returning the metrics data

Enabling App Metrics

Set the `FLOGO_HTTP_SERVICE_PORT` environment variable to point to the port number of the HTTP service that provides APIs for collecting app metrics. This enables the runtime HTTP service.

Procedure

1. Run the following:

```
FLOGO_HTTP_SERVICE_PORT=<port> ./<app-binary>
```

2. Run the `curl` command for the appropriate REST method. Refer to [App Statistics](#) for details on each method. Some examples are:

```
curl -X POST http://localhost:7777/app/metrics
curl -X GET http://localhost:7777/app/metrics/flows
curl -X DELETE http://localhost:7777/app/metrics
```

Enabling statistics collection using environment variables

To enable metrics collection through an environment variable:

Procedure

1. Run the following:

```
FLOGO_HTTP_SERVICE_PORT=<port> FLOGO_APP_METRICS=true ./<appname>
```

2. Run the `curl` command for the appropriate REST method. Refer to [App Statistics](#) for details on each method. Some examples are:

```
curl -X GET http://localhost:7777/app/metrics/flows
curl -X DELETE http://localhost:7777/app/metrics/flows
```

Example: retrieve specific metrics for an app

The following is an example of how you would run the above steps for a fictitious app named `REST_Echo`.

```
FLOGO_HTTP_SERVICE_PORT=7777 FLOGO_APP_METRICS=true ./REST_Echo-darwin-
amd64
```

```
curl -X GET http://localhost:7777/app/metrics/flows
```

```
{"app_name":"REST_Echo","app_version":"1.0.0","flows":
[{"started":127639,"completed":126784,"failed":0,"avg_exec_time":0,
"min_exec_time":0,"max_exec_time":4,"flow_name":"PostBooks"}]}
```

```
curl -X GET http://localhost:7777/app/metrics/flow/PostBooks/activities
```

```
{"app_name":"REST_Echo","app_version":"1.0.0","tasks":
[{"started":127389,"completed":126908,"failed":0,"avg_exec_time":0,
"min_exec_time":0,"max_exec_time":4,"flow_name":"PostBooks","task_
name":"Return"}]}
```

Logging App Metrics

You can record app metrics of flows and activities to the console logs. To enable the logging of app metrics, use the following environment variables:

Environment Variable Name	Default Values	Description
FLOGO_APP_METRICS_LOG_EMITTER_ENABLE	False	<p>This property can be set to either True or False:</p> <ul style="list-style-type: none"> • True: App metrics are displayed in the logs with the values set in FLOGO_APP_METRICS_LOG_EMITTER_CONFIG. • False: App metrics are not displayed in the logs. <p>If this variable is not provided, the default values are used.</p>
FLOGO_APP_METRICS_LOG_EMITTER_CONFIG	Both flow and Activity	<p>This property can be set to either flow level or Activity level. The format for setting the property is:</p> <pre> {"interval": "<interval_in_seconds>", "type": ["flow", "Activity"]} </pre> <p>where:</p> <ul style="list-style-type: none"> • interval is the time interval (in seconds) after which the app metrics are displayed in the console. • type is the level at which the app metrics are to be displayed - flow or Activity. Depending on which level you set, the app metrics are displayed only for that level. <p>For example:</p> <pre> {"interval": "1s", "type": ["flow", "Activity"]} </pre>

For a list of fields or app metrics returned in the response, refer to [Fields returned in the response](#).

Fields returned in the response

The following table describes the fields that can be returned in the response.

Flow

Name	Description
app_name	Name of the app
app_version	Version of the app
flow_name	Name of the flow
started	Total number of times a given flow is started
completed	Total number of times a given flow is completed
failed	Total number of times a given flow has failed
avg_exec_time	Average execution time of a given flow for successful executions
min_exec_time	Minimum execution time for a given flow
max_exec_time	Maximum execution time for a given flow

Activity

Name	Description
app_name	Name of the app
app_version	Version of the app
flow_name	Name of the flow
Activity_name	Name of the Activity
started	Total number of times a given Activity is started
completed	Total number of times a given Activity is completed

Name	Description
failed	Total number of times a given Activity has failed
avg_exec_time	Average execution time of a given Activity for successful executions
min_exec_time	Minimum execution time for a given Activity
max_exec_time	Maximum execution time for a given Activity

Prometheus

Flogo apps support integration with Prometheus for app metrics monitoring. Prometheus is a monitoring tool that helps in analyzing the app metrics for flows and activities.

Prometheus servers scrape data from the HTTP `/metrics` endpoint of the apps.

Prometheus integrates with Grafana, which provides better visual analytics.

Flogo apps expose the following flow and Activity metrics to Prometheus. These metrics are measured in milliseconds:

Labels	Description
flogo_flow_execution_count:	Total number of times the flow is started, completed, or failed
ApplicationName	Name of app
ApplicationVersion	Version of app
FlowName	Name of flow
State	State of the flow. One of the following states: <ul style="list-style-type: none"> Started Completed Failed
flogo_flow_execution_duration_msec:	Total time (in ms) taken by the flow for successful

Labels	Description
completion or failure	
ApplicationName	Name of app
ApplicationVersion	Version of app
FlowName	Name of flow
State	State of the flow. One of the following states: <ul style="list-style-type: none"> • Completed • Failed
flogo_Activity_execution_count: Total number of times the Activity is started, completed, or failed	
ApplicationName	Name of app
ApplicationVersion	Version of app
FlowName	Name of flow
ActivityName	Name of Activity
State	State of the Activity. One of the following states: <ul style="list-style-type: none"> • Started • Completed • Failed
flogo_Activity_execution_duration_msec: Total time (in ms) taken by the Activity for successful completion or failure	
ApplicationName	Name of app
ApplicationVersion	Version of app

Labels	Description
FlowName	Name of flow
ActivityName	Name of Activity
State	State of the Activity. One of the following states: <ul style="list-style-type: none"> Completed Failed

Note: Deprecated in Flogo Enterprise 2.10.0.

flogo_flow_metrics: Used for flow-level queries

ApplicationName	Name of app
ApplicationVersion	Version of app
FlowName	Name of flow
Started	Total number of times flow is started
Completed	Total number of times flow is completed
Failed	Total number of times flow is failed

Note: Deprecated in Flogo Enterprise 2.10.0.

flogo_Activity_metrics: Used for Activity-level queries

ApplicationName	Name of app
ApplicationVersion	Version of app
FlowName	Name of flow
ActivityName	Name of Activity

Labels	Description
Started	Total number of times Activity is started in given flow
Completed	Total number of times Activity is completed in given flow
Failed	Total number of times Activity is failed in given flow

For a list of some often-used flow-level queries, refer to the section, [Often-Used Queries](#).

Using Prometheus to Analyze Flogo App Metrics

To enable Prometheus monitoring of Flogo apps, run the following:

```
FLOGO_HTTP_SERVICE_PORT=7779 FLOGO_APP_METRICS_PROMETHEUS=true ./<app-binary>
```

Setting `FLOGO_APP_METRICS_PROMETHEUS` variable to `true` enables Prometheus monitoring of Flogo apps. The variable, `FLOGO_HTTP_SERVICE_PORT` is used to set the port number on which the Prometheus endpoint is available.

Use the following endpoint URL in Prometheus server configuration:

```
http://<APP_HOST_IP>:<FLOGO_HTTP_SERVICE_PORT>/metrics
```

For example:

```
http:// 192.0.2.0:7779/metrics
```

Often-Used Queries

Prometheus uses the PromQL query language. This section lists some of the most often-used queries at the flow level.

Flow-level Queries

To Get this Metric	Use this Query
Total number of flows that got successfully executed per app	count(flogo_flow_execution_count {State="Completed"}) by (AppName, FlowName)
Total number of flows that failed per app	count(flogo_flow_execution_count {State="Failed"}) by (AppName, FlowName)
Total number of flows that executed successfully across all apps (when you are collecting metrics for multiple apps)	count(flogo_flow_execution_count {State="Completed"})
Total number of flows that failed across all apps (when you are collecting metrics for multiple apps)	count(flogo_flow_execution_count {State="Failed"})
Total time taken by flows which got executed successfully	sum(flogo_flow_execution_duration_msec {State="Completed"}) by (AppName, FlowName)
Total time taken by flows which failed	sum(flogo_flow_execution_duration_msec {State="Failed"}) by (AppName, FlowName)
Minimum time taken by the flows that got executed successfully (what was the minimum time taken by a flow from amongst the flows that executed successfully)	min(flogo_flow_execution_duration_msec {State="Completed"}) by (AppName)
Minimum time taken by flows which failed	min(flogo_flow_execution_duration_msec {State="Failed"}) by (AppName)
Maximum time taken by flows which executed successfully	max(flogo_flow_execution_duration_msec {State="Completed"}) by (AppName)

To Get this Metric	Use this Query
Maximum time taken by flows which failed	<code>max(flogo_flow_execution_duration_msec {State="Failed"}) by (AppName)</code>
Average time taken by flows which executed successfully	<code>avg(flogo_flow_execution_duration_msec {State="Completed"}) by (AppName, FlowName)</code>
Average time taken by flows which failed	<code>avg(flogo_flow_execution_duration_msec {State="Failed"}) by (AppName, FlowName)</code>

Activity-level Queries

To Get this Metric	Use this Query
Total number of activities that got successfully executed per flow and app	<code>count(flogo_Activity_execution_count {State="Completed"}) by (AppName, FlowName, ActivityName)</code>
Total number of activities that failed per flow and app	<code>count(flogo_Activity_execution_count {State="Failed"}) by (AppName, FlowName, ActivityName)</code>
Total number of activities that executed successfully across all apps (when you are collecting metrics for multiple apps)	<code>count(flogo_Activity_execution_count {State="Completed"})</code>
Total number of activities that failed across all apps (when you are collecting metrics for multiple apps)	<code>count(flogo_Activity_execution_count {State="Failed"})</code>
Individual time taken by activities which got executed successfully per app and flow	<code>sum(flogo_Activity_execution_duration_msec {State="Failed"}) by (AppName, FlowName, ActivityName)</code>
Individual time taken by activities which failed per app and flow	<code>sum(flogo_Activity_execution_duration_msec {State="Failed"}) by (AppName,</code>

To Get this Metric	Use this Query
	FlowName,ActivityName)
Minimum time taken by the Activity that got executed successfully within a given flow and app	min(flogo_Activity_execution_duration_msec {State="Completed"}) by (AppName, FlowName,ActivityName)
Minimum time taken by a failed Activity within a given flow and app	min(flogo_Activity_execution_duration_msec {State="Failed"}) by (AppName, FlowName,ActivityName)
Maximum time taken by an Activity which executed successfully within a given flow and app	max(flogo_Activity_execution_duration_msec {State="Completed"}) by (AppName, FlowName,ActivityName)
Maximum time taken by an Activity which failed within a given flow and app	max(flogo_Activity_execution_duration_msec {State="Failed"}) by (AppName, FlowName,ActivityName)
Average time taken by an Activity which executed successfully within a given flow and app	avg(flogo_Activity_execution_duration_msec {State="Completed"}) by (AppName, FlowName,ActivityName)
Average time taken by an Activity which failed within a given flow and app	avg(flogo_Activity_execution_duration_msec {State="Failed"}) by (AppName, FlowName,ActivityName)

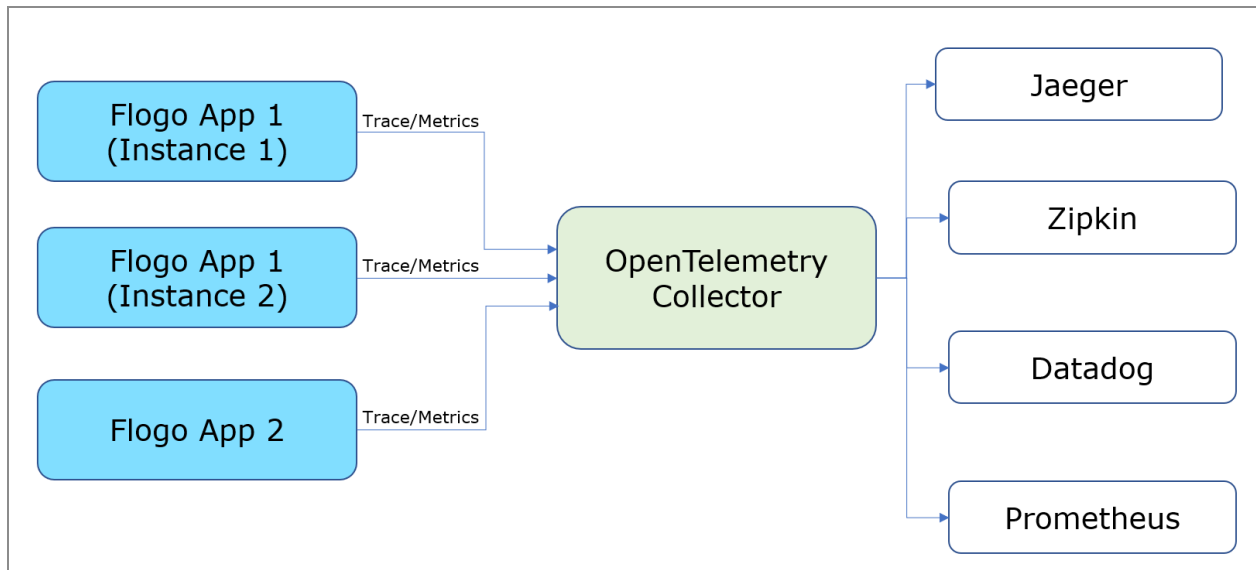
OpenTelemetry Collector

Flogo supports integration with OpenTelemetry (OT) Collector. The goal of this collector is to create standard software development kit for tracing, metrics and logging which different vendors like Jaeger, Zipkin, Datadog, Prometheus adopt. You have the flexibility to switch vendors without changing the application logic with OpenTelemetry.

i Note: To use this feature for TIBCO Cloud Integration deployments, ensure that the OpenTelemetry Collector is reachable from app containers.

Architecture

This is the schematic view of how the OT collector works:



Note: You can use the same architecture for Distributed tracing as well. For more information, see [Tracing Apps by Using OpenTelemetry Collector](#).

Configuration

The parameters listed below are required for the configuration of the OT collector:

Name	Required	Default	Description
FLOGO_ OTEL_ METRICS	Yes	False	Enable OpenTelemetry metrics for Flogo app
FLOGO_ OTEL_ METRICS_ ATTRIBUTES	No	None	Add one or more custom attributes to the metrics. For example, FLOGO_OTEL_METRICS_ATTRIBUTES="deployment_type=flogo"
FLOGO_ OTEL_ ENDPOINT	Yes	None	OpenTelemetry protocol (OTLP) receiver endpoint

Name	Required	Default	Description
OTEL_OTLP_ENDPOINT			configured for OpenTelemetry Collector. For gRPC protocol, set <host>:<otlp_grpc_port> For http protocol, set https://<host>:<otlp_http_port>
FLOGO_OTEL_OTLP_HEADERS	No	None	Set one or more custom gRPC/HTTP headers in the request to the collector. For example, FLOGO_OTEL_OTLP_HEADERS="Authorization=Bearer <token>,API_KEY=<api_key_value>".
FLOGO_OTEL_TLS_SERVER_CERT	No	None	Set PEM encoded Server/CA certificate when TLS is enabled for OTLP receiver. You can configure a path to the certificate or use base64 encoded certificate value. A file path must be prefixed with "file://". e.g. FLOGO_OTEL_TLS_SERVER_CERT="file:///Users/opentelemetry/certs/cert.pem" or FLOGO_OTEL_TLS_SERVER_CERT=<base64_encoded_server_certificate>. When this certificate is not set, unsecure connection is established with the collector.

Monitor Flogo apps metrics using OpenTelemetry

You can see the number of flows and activities executed in the app as per the below metrics:

Metrics	Label	Description
flogo_activity_executions_total	-	Total number of times the activity is started, completed, or failed.
	app_name	Name of application
	app_version	Version of application
	flow_name	Name of flow

Metrics	Label	Description
	activity_name	Name of activity
	state	State of activity - Started, Completed or Failed
	host_name	Name of the host or app instance ID
flogo_flow_executions_total	-	Total number of times the flow is started, completed or failed
	app_name	Name of application
	app_version	Version of application
	flow_name	Name of flow
	state	State of flow - Started, Completed or Failed
	host_name	Name of the host or app instance ID

Example

Prometheus

```
FLOGO_OTEL_METRICS=true FLOGO_OTEL_OTLP_ENDPOINT="localhost:4317" FLOGO_OTEL_METRICS_ATTRIBUTES="deployment=local,product=flogo" ./<app-executable>
```

Distributed Tracing

Distributed tracing allows you to log information about an app's behavior during its execution. It shows the path an app takes from start to finish. You can then use the information to troubleshoot performance bottlenecks, errors, and debugging failures in the app execution.

As the app travels through different services, each segment is recorded as a *span*. A span is a building block of a trace and represents work done with time intervals and associated metadata. All the spans of an app are combined into a single *trace* to give you a picture of

an entire request. A trace represents an end-to-end execution; made up of single or multiple spans. A *Tracer* is the actual implementation that records the spans and publishes them.

Distributed tracing is used to help you identify issues with your app (performance of the app or simply debugging an issue) instead of going through stack traces. The use of distributed tracing is particularly useful in a distributed microservice architecture environment where each app is instrumented by a tracing framework and while the tracing framework runs in the background, you can monitor each trace in the UI. You can use that to track any abnormalities or issues to identify the location of the problem.

Some Considerations

Keep the following in mind when using the distributed tracing capability in Flogo Enterprise:

- At any given point in time, only one tracer can be registered - if you try to register multiple tracers, only the first one that you register is accepted and used at run time to trace all the activities of the flow.
- All the traces start at the flow level. There are two relations between spans - a span is either the child of a parent span or the span is a span that follows (comes after) another span. You should be able to see all the operations and the traces for the flows and activities that are part of an app. Traces of the triggers used in the app are not shown.
- Tracing can be done across apps bypassing the tracing context from one app to another. To trace across multiple apps, you must make sure that all apps are instrumented with similar tracing frameworks, such as Jaeger semantics so that they understand the framework language. Otherwise, you can't get a holistic following of the entire trace through multiple services.
- When looping is enabled for an Activity, each loop is considered one span, since each loop calls the server which triggers a server flow.
- If a span is passed on to the trigger, that span becomes the parent span. You should be able to see how much time is taken between the time the event is received by the trigger and the time the trigger replies. This only works for triggers that support the extraction of the context from the underlying technology, for instance, triggers those support HTTP headers.

The **ReceiveHTTPMessage** REST trigger and **InvokeRESTService** Activity are supported for this release where the REST trigger can extract the context from the

request and **InvokeRESTService** Activity can inject the context into the request. If two Flogo apps are both Jaeger-enabled, when one app calls the other, you can see the chain of events (invocation and how much time is taken by each invocation) in the Jaeger UI. If app A is calling app B, the total request time taken by app A is the cumulative of the time taken by all activities in app A plus the time taken by the service that it calls. If you open up each invocation separately, you can see the details of how much time was taken by each Activity in that invocation.

- Triggers that support span (for instance the REST trigger) are always the parent, so any flows that are attached to that trigger are always the children of the trigger span. Trigger span is completed only after the request goes to the flow and the flow returns.
- A subflow becomes a child of the Activity from which it is called.

Tracing Apps Using Jaeger

Flogo apps provide an implementation of the OpenTracing framework using the Jaeger backend. The Flogo app binary is built with Jaeger implementation and can be enabled by setting the `FLOGO_APP_MONITORING_OT_JAEGER` environment variable to `true`. You can track how the flow went through, the execution time for each Activity, or in case of failure, the cause of the failure.

Each app is displayed as a service in the Jaeger UI. In a Flogo app, each flow is one operation (trace) and each Activity in the flow is a span of the trace. A trace is the complete lifecycle of a group of spans. The flow is the root span and its activities are its child spans.

Prerequisites: The following prerequisites must be met before using the tracing capability in Flogo Enterprise:

- By default, Jaeger is not enabled in Flogo, hence tracing is not enabled. To enable Jaeger, set the `FLOGO_APP_MONITORING_OT_JAEGER` environment variable to `true`.
- Ensure that the Jaeger server is installed, running, and accessible to the Flogo app binary.
- If your Jaeger server is running on a machine other than the machine on which your app resides, be sure to set the `JAEGER_ENDPOINT=http://<JAEGER_HOST>:<HTTP_TRACE_COLLECTOR_PORT>/api/traces` environment variables. Refer to the [Environment Variables](#) page for the environment variables that you can set.

Flogo Enterprise-Related Tags in Jaeger

In OpenTracing, each trace and span have their tags. Tags are useful for filtering traces, for example, if you want to search for a specific trace or time interval.

Note: Adding your custom tags for any one span (Activity) only is currently not supported. Any custom tags that you create are added to **all** spans and traces.

Flogo Enterprise introduces the following Flogo-specific tags:

For flows

flow_name	Name of the flow
flow_id	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times)

For activities

flow_name	Name of the flow
flow_id	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times)
task_name	Name of Activity
taskInstance_id	Unique instance ID that is generated by the Flogo engine. This identity is used to identify the specific instance of an Activity when an Activity is iterated multiple times. This ID is used in looping constructs such as iterator or Repeat while true .

For subflows

parent_flow	Name of the parent flow
-------------	-------------------------

parent_flow_id	Unique ID of the parent flow
flow_name	Name of the subflow
flow_id	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times)

The tag values are automatically generated by the Flogo Enterprise runtime. You cannot override the default values. You have the option to set custom tags by setting them in the environment variable `JAEGER_TAGS` as key/value pair. Keep in mind that these tags are added to **all** spans and traces.

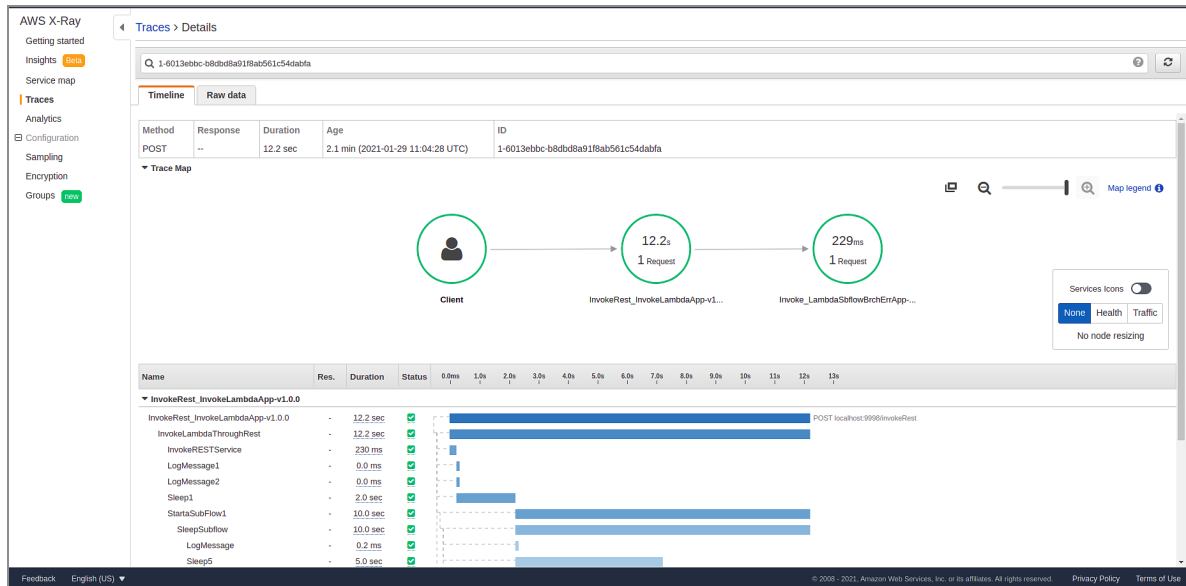
Refer to the [Environment Variables](#) page for the environment variables that you can set.

Tracing Apps by Using AWS X-Ray

If you are running your Flogo app on the cloud or in your local environment, you can track your app performance or troubleshoot issues by using AWS X-Ray. For more information about AWS X-Ray, refer to [AWS X-Ray](#).

When you use AWS X-Ray for tracing, your app sends trace data to AWS X-Ray. X-Ray processes the data to generate a service map and searchable trace summaries. For each flow, subflow, and Activity, details such as execution time are displayed on the AWS X-Ray dashboard.

The following example shows the trace details of the `InvokeRest_InvokeLambdaApp-v1.0.0` app. It includes details such as activities that were invoked, and their execution time and status.



Before you begin

Make sure that you meet the following requirements:

- **Knowledge of AWS X-Ray:** For more information, refer to [AWS XRay](#).
- **For an app containing a non-Lambda trigger:**
 - **AWS X-Ray daemon:** You must have an AWS X-Ray daemon running on your machine to send trace data to the AWS X-Ray service. Alternatively, your app must have access to another machine where the daemon is running. Download the AWS X-Ray daemon from the AWS website and run the AWS X-Ray daemon.
 - **Environment variable:** If the AWS X-Ray daemon and app are running on two different machines, set the environment variable `AWS_XRAY_DAEMON_ADDRESS` to the IP address where the AWS X-Ray daemon is running for receiving traces. You need not set this variable if the daemon and app are running on the same machine.
- **For an app containing a Lambda trigger:**
 - To trace the app end-to-end, TIBCO recommends that you enable the **Active Tracing** option in AWS along with the Flogo tracing feature. **Active Tracing** provides all the details of the app while the Flogo tracing feature provides details specific to the Flogo app. For example, details such as how long it took to initialize the container, are provided by **Active Tracing**. Details specific to the Flogo implementation (such as the flows, sub-flows, or activities executed)

are provided by the Flogo tracing feature.

- For an app containing a Lambda trigger, you need not run the AWS X-Ray daemon. This is because AWS X-Ray is integrated with AWS Lambda.
- Add the following permissions to the execution role. For more information on how to add the permissions, refer to the [AWS Documentation](#).
 - `xray:PutTraceSegments`
 - `xray:PutTelemetryRecords`

i Note: The AWS API Gateway Lambda and S3 Bucket Event Lambda triggers are not supported.

Enabling Tracing Using AWS X-Ray

To enable tracing using AWS X-Ray, set the `FLOGO_AWS_XRAY_ENABLE` environment variable to true. The default is false.

Search Using Annotations

You can search based on predefined Flogo annotations. The following annotation is available in this release:

`flogo_flow_name`: Name of the flow

Here is an example of using annotations to search:

```
annotation.flogo_flow_name="sampleFlow"
```

Metadata

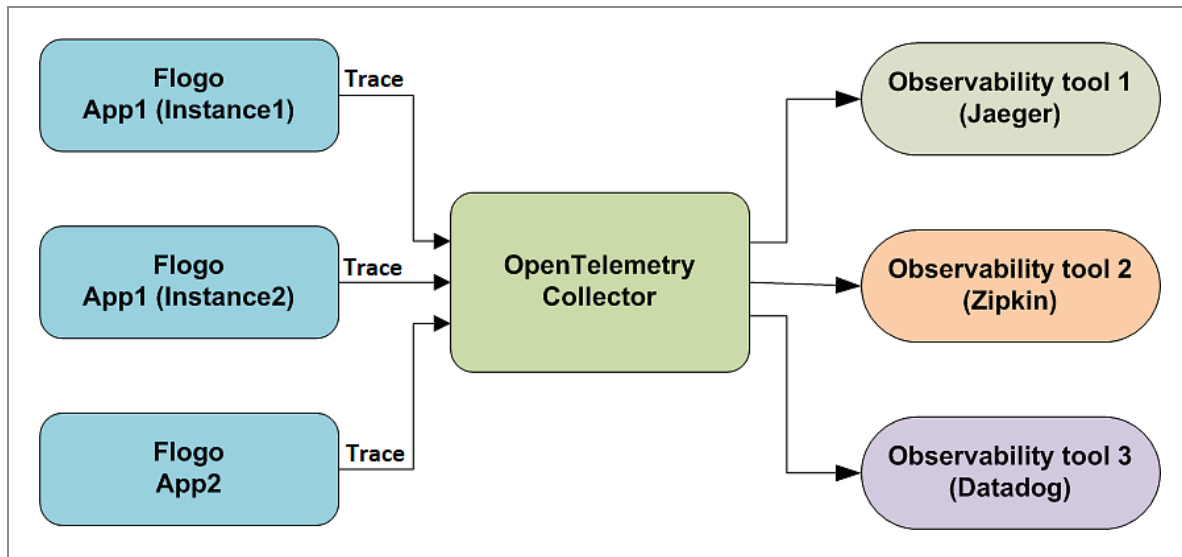
The following metadata about an app is stored in the `flogo` namespace:

- `flow_name`: Name of the flow
- `Activity_name`: Name of the Activity

This metadata can be used when debugging. You can use the metadata to identify the exact errors, stack traces, flow name, Activity name, and so on. Note that the metadata cannot be used for searching traces.

Tracing Apps by Using OpenTelemetry Collector

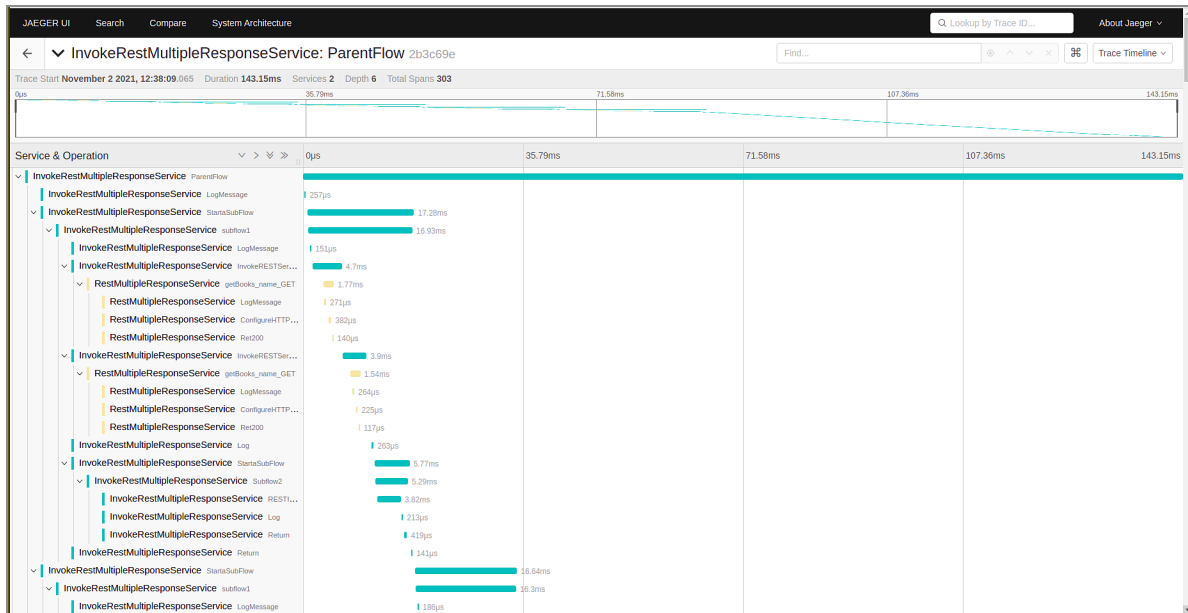
By using OpenTelemetry Collector, you can capture traces from your Flogo app and send them to observability vendor tools such as Jaeger, Zipkin, and Datadog. This gives you the flexibility to switch between observability vendor tools without changing the logic of your app. For more information about OpenTelemetry Collector, see [OpenTelemetry documentation](#).



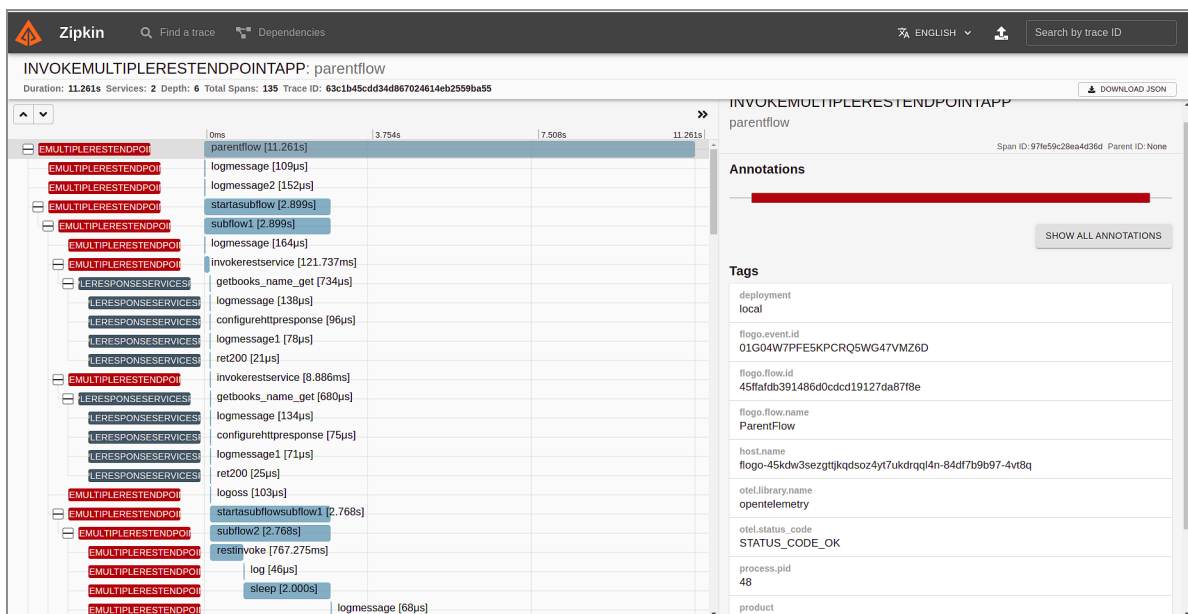
When you use this feature, traces of the Flogo app are sent to the OpenTelemetry Collector. OpenTelemetry Collector has vendor-specific configurations that allow you to send these traces to supported observability vendor tools. For example, you can specify Zipkin-specific configurations in the `otel-zipkin-collector-config.yaml` configuration file for the traces to be displayed on the Zipkin dashboard.

The following screenshots show traces from one Flogo app on two different observability vendor tools, Jaeger and Zipkin.

Jaeger output of a Flogo app



Zipkin output of a Flogo app



Enabling Tracing for OpenTelemetry Collector

Before you begin

Make sure that you meet the following requirements:

- Ensure that you can connect to the OpenTelemetry Collector.

**Note:**

- To use this feature for TIBCO Cloud Integration deployments, ensure that the OpenTelemetry Collector is reachable from app containers.
- If the connection to OpenTelemetry Collector is lost, traces during that time duration are not collected.

- Install an observability vendor tool of your choice: Jaeger, Zipkin, Datadog, and so on.

Mandatory Configuration Parameters

To enable tracing by using OpenTelemetry Collector, set the following mandatory parameters:

Name	Default	Description
FLOGO_OTEL_TRACE	False	Enables tracing by using OpenTelemetry Collector.
FLOGO_OTEL_OTLP_ENDPOINT	None	<p>Specifies the OpenTelemetry protocol (OTLP) receiver endpoint for OpenTelemetry Collector.</p> <p>Supported protocols are:</p> <ul style="list-style-type: none"> • gRPC: Set to <host>:<otlp_grpc_port>. • HTTP: Set to https://<host>:<otlp_http_port>.

Optional Configuration Parameters

You can also use some optional configuration parameters when tracing apps using OpenTelemetry Collector. Here are some commonly used parameters and their descriptions:

Name	Default	Description
FLOGO_OTEL_TRACE_ATTRIBUTES	None	<p>Add one or more custom attributes to the trace. The format is key-value pairs separated by commas. For example, to filter based on the deployment type and deployment cluster, you can use:</p> <pre>FLOGO_OTEL_TRACE_ATTRIBUTES="deployment.type=staging,deployment.cluster=staging3"</pre>
FLOGO_OTEL_OTLP_HEADERS	None	<p>Set one or more custom gRPC or HTTP headers in the request to the OpenTelemetry Collector. The format is key-value pairs separated by commas. For example:</p> <pre>FLOGO_OTEL_OTLP_HEADERS="Authorization=Bearer <token>,API_KEY=<api_key_value>"</pre>
FLOGO_OTEL_TLS_SERVER_CERT	None	<p>If TLS is enabled for OpenTelemetry protocol receiver, set PEM-encoded server or CA. You can configure a path to the certificate or use base64-encoded certificate value. A file path must be prefixed with "file://".</p> <p>For example:</p> <ul style="list-style-type: none"> FLOGO_OTEL_TLS_SERVER_CERT="file:///Users/opentelemetry/certs/cert.pem" FLOGO_OTEL_TLS_SERVER_CERT=<base64_encoded_server_certificate> <p>You can also encrypt base64 encoded certificate value by using either TIBCO Cloud Integration platform API or by using app executable and set it to the environment variable with prefix "SECRET:"</p> <p>For example:</p> <ul style="list-style-type: none"> FLOGO_OTEL_TLS_SERVER_CERT=SECRET:<encrypted_base64_encoded_cert_value> <p>For details about encryption, see Encryption using App executable or Encryption using TIBCO Cloud Platform API.</p> <p>When this certificate is not set, an unsecure connection is established with OpenTelemetry Collector.</p>

Tracing With OpenTelemetry Collector

Using OpenTelemetry Collector with Jaeger

The Jaeger Docker image includes OpenTelemetry Collector. So, you need not run OpenTelemetry Collector separately.

```
docker run --name jaeger -p 13133:13133 -p 16686:16686 -p 4317:55680 -d
--restart=unless-stopped jaegertracing/opentelemetry-all-in-one
```

For example:

```
FLOGO_OTEL_TRACE=true FLOGO_OTEL_OTLP_ENDPOINT="localhost:4317" FLOGO_
OTEL_TRACE_ATTRIBUTES="deployment=local,product=flogo" ./TimerOTel-
darwin_amd64
```

Using OpenTelemetry Collector with Zipkin

Procedure

1. Start Zipkin as follows:

```
docker run -it --rm -p 9411:9411 -d --name zipkin openzipkin/zipkin
```

2. Update the OpenTelemetry Collector configuration file for Zipkin, `otel-zipkin-collector-config.yaml`, as follows:

```
receivers:
otlp:
  protocols:
    http:

exporters:
zipkin:
  # Change IP
  endpoint: "http://xxx.xxx.x.x:xxxx/api/v2/spans"
  format: proto

processors:
batch:
```

```

service:
  pipelines:
  traces:
  receivers: [otlp]
  processors: [batch]
  exporters: [zipkin]

```

3. Start OpenTelemetry Collector with Zipkin Exporter as follows:

```

docker run -d --rm -p 4318:4318 -v "${PWD}/otel-zipkin-collector-
config.yaml":/otel-collector-config.yaml --name otelcol
otel/opentelemetry-collector:0.35.0 --config otel-collector-
config.yaml

```

For example:

```

FLOGO_OTEL_TRACE=true FLOGO_OTEL_OTLP_ENDPOINT="https://localhost:4318"
FLOGO_OTEL_TRACE_ATTRIBUTES="deployment=local,product=flogo"
./TimerOTel-darwin_amd64

```

Using OpenTelemetry Collector with Zipkin (with TLS)

Procedure

1. Update server-cert-gen.sh as follows:

```

openssl req -newkey rsa:2048 \
-new -nodes -x509 \
-days 3650 \
-out cert.pem \
-keyout key.pem \
-extensions san \
-config <(echo '[req]'; echo 'distinguished_name=req';
echo '[san]'; echo 'subjectAltName=DNS:localhost,DNS:127.0.0.1') \
-subj
"/C=US/ST=California/L=Sunnyvale/O=TIBCO/OU=Flogo/CN=localhost"

```

2. Start Zipkin.

```

docker run -it --rm -p 9411:9411 -d --name zipkin openzipkin/zipkin

```

3. Update the OpenTelemetry Collector configuration file for Zipkin, `otel-zipkin-collector-config.yaml`, as follows:

```

receivers:
  otlp:
  protocols:
  grpc:
  tls_settings:
  cert_file: /var/certs/cert.pem
  key_file: /var/certs/key.pem

exporters:
  zipkin:
  endpoint: "http://xxx.xxx.x.x:xxxx/api/v2/spans"
  format: proto

processors:
  batch:

service:
  pipelines:
  traces:
  receivers: [otlp]
  processors: [batch]
  exporters: [zipkin]

```

4. Create a `certs` directory under the current directory and copy `cert.pem` and `key.pem` in the `certs` directory.
5. Start OpenTelemetry Collector with Zipkin Exporter as follows:

```

docker run -d --rm -p 4317:4317 -v "${PWD}/otel-zipkin-collector-
config.yaml":/otel-collector-config.yaml -v
"${PWD}/certs":/var/certs --name otelcol otel/opentelemetry-
collector:0.35.0 --config otel-collector-config.yaml

```

For example:

```

FLOGO_OTEL_TRACE=true FLOGO_OTEL_OTLP_ENDPOINT="localhost:4317"
FLOGO_OTEL_TLS_SERVER_
CERT="file:///Users/dev/Installations/OpenTelemetry/zipkin/certs/cert.pe
m"
FLOGO_OTEL_TRACE_ATTRIBUTES="deployment=local,product=flogo"
./TimerOTel-darwin_amd64

```

Flogo Related Attributes in OpenTelemetry Collector

In OpenTelemetry, each trace has its own attributes. These attributes are useful for filtering traces, for example, if you want to search for a specific trace or time interval.



Note:

- Adding your custom attributes for only one span (Activity) is currently not supported. Any custom tags that you create are added to **all** traces
- The prefix 'flogo' is added to tags with product-specific attributes only.

The following attributes specific to Flogo are available:

For flows

flogo.event.id	The event ID is the unique ID of a single request, job or a action initiated by the user.
flogo.flow.id	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times).
flogo.flow.name	Name of the flow.

For activities

flogo.flow.name	Name of the flow.
flogo.flow.id	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times).
flogo.task.name	Name of Activity.
flogo.taskInstance.id	Unique instance ID that is generated by the Flogo engine. This identity is used to identify the specific instance of an Activity when an Activity is iterated multiple times. This ID

is used in looping constructs such as **iterator** or **Repeat while true**.

For subflows

<code>flogo.parent.flow</code>	Name of the parent flow
<code>flogo.flogo.parent.flow.id</code>	Unique ID of the parent flow
<code>flogo.flow.name</code>	Name of the subflow
<code>flogo.flow.id</code>	Unique instance IDs that are generated by the Flogo engine. They are used to identify specific instances of a flow (such as when the same flow is triggered multiple times)

The attribute values are automatically generated at runtime. You cannot override the default values. You have the option to set attributes by setting them in the environment variable `FLOGO_OTEL_TRACE_ATTRIBUTES` as key/value pair. Keep in mind that these tags are added to **all** traces.

Using APIs

You can obtain the runtime statistics of the Go language in Flogo Enterprise.

Healthcheck API

Flogo Enterprise runtime allows you to enable healthcheck for a Flogo app that is running.

To enable healthcheck for your running app:

Procedure

1. Set `FLOGO_HTTP_SERVICE_PORT` to enable runtime HTTP Service as follows:

```
FLOGO_HTTP_SERVICE_PORT=<port> ./<app_name>
```

2. Run the following command:

```
curl http://localhost:<port>/ping
```

i Note: Currently, healthcheck endpoint returns HTTP status 200 only when all triggers in the app are successfully started. Otherwise, it returns HTTP status 500.

Go Language Runtime Statistics and Profiling

Flogo Enterprise allows you to gather runtime system statistics for a Flogo app that is running.

⚠ Warning: Your management port must be set for the Flogo app, to call the API to gather Go language runtime statistics. To set a different management port for your Flogo app, run `FLOGO_HTTP_SERVICE_PORT=<port> ./<app-name>`. You can use curl to call this API.

To obtain the system statistics on your running app:

Procedure

1. From the folder in which your app binary resides, enable the HTTP service using the following command:

```
FLOGO_HTTP_SERVICE_PORT=<port> ./<app_name>
```

2. Run the following command:

```
curl http://localhost:<port>/debug/vars
```

The command returns the following statistics:

System Metric Name	Description
cmdline	Command-line arguments passed to the app binary
cpus	Number of logical CPUs usable by the current process
goroutines	The number of Go routines that currently exist
memstats	Memory statistics for the current process. See the Golang documentation for details.
processid	System process ID
version	Go language version used to build the app

Profiling your app runtime

You can collect and visualize runtime profiling data for Flogo apps using the pprof tool in Golang.

Endpoint	Description
/debug/pprof	List all profiles
/debug/pprof/profile	<p>Profile current CPU usage. By default, it is profiled for every 30 seconds. To change the profiling interval, set the seconds query parameter to a desired value. For example,</p> <pre>go tool pprof http://localhost: <port>/debug/pprof/profile?seconds=15</pre>
/debug/pprof/heap	<p>A sampling of memory allocations of live objects. For example,</p> <pre>go tool pprof http://localhost:<port>/debug/pprof/heap</pre>

Endpoint	Description
/debug/pprof/goroutine	Stack traces of all current Go routines. For example, <pre>go tool pprof http://localhost:<port>/debug/pprof/goroutine</pre>
/debug/pprof/trace	A trace of execution of the current program. For example, <pre>go tool pprof http://localhost:<port>/debug/pprof/trace</pre>

CPU and Memory Profiling

If you observe low throughputs or high memory usage, you can enable CPU and/or Memory profiling for your Flogo app. Enabling this profiling impacts performance. Hence, we do not recommend enabling them in a production environment.

Before you begin

- You must have GO version 1.9.0 or higher installed.
- Make sure that the pprof tool is installed on your machine. Refer to [PPOF](#) for more details on the pprof tool.

Enabling CPU Profiling

To enable CPU profiling:

Procedure

1. Open a command prompt or terminal.
2. Change the directory to the folder in which your app binary is located.
3. Run the following command:

```
./<app_binary> -cpuprofile <file>
```

where *<file>* is the profile file. For example, `./StockService -cpuprofile`

```
/home/users/StockService_cpu.prof
```

Enabling Memory Profiling

To enable memory profiling:

Procedure

1. Open a command prompt or terminal.
2. Change the directory to the folder in which your app binary is located.
3. Run the following command:

```
./<app_binary> -memprofile <file>
```

where *<file>* is the profile file. For example, `./StockService -memprofile /home/users/StockService_mem.prof`

Enabling CPU and Memory Profiling in a Single Command

To enable CPU and memory profiling in a single command:

Procedure

1. Open a command prompt or terminal.
2. Change the directory to the folder in which your app binary is located.
3. Run the following command:

```
./<app_binary> -memprofile <file> -cpuprofile <file>
```

Analyzing your profiling data

Once you capture the profiling data, analyze it using `pprof` by running the following command:

```
go tool pprof <profile file>
```

Monitoring and Managing Enterprise Apps in TIBCO Cloud Integration

With the TIBCO Cloud Integration - Hybrid Agent, you can now monitor Remote apps and perform various operations through the TIBCO Cloud Integration user interface, such as scaling the app instances, updating application and engine variables, starting or stopping an app, and monitoring app metrics. Remote apps are auto-discovered by the Hybrid Agent.

For detailed information, see [Configuring Remote Apps](#).

Environment Variables

This section lists the environment variables that are associated with the Flogo Enterprise runtime environment.

Environment Variable Name	Default Values	Description
FLOGO_RUNNER_QUEUE_SIZE	50	The maximum number of events from all triggers that can be queued by the app engine.
FLOGO_RUNNER_WORKERS	5	The maximum number of concurrent events that can be run by the app engine from the queue.
FLOGO_LOG_LEVEL	INFO	Used to set a log level for the Flogo app. Supported values are: <ul style="list-style-type: none">• INFO• DEBUG• WARN• ERROR This variable is supported for Remote

Environment Variable Name	Default Values	Description
FLOGO_ LOGACTIVITY_ LOG_LEVEL	INFO	Apps managed with the TIBCO Cloud Integration Hybrid Agent. Used to control logging in the Log activity. Values supported, in the order of precedence, are: <ul style="list-style-type: none"> • DEBUG • INFO • WARN • ERROR For example: <ul style="list-style-type: none"> • If the Log level is set to WARN, WARN and ERROR logs are filtered and displayed. • If Log Level is set to DEBUG, then DEBUG, INFO, WARN, and ERROR logs are displayed.
FLOGO_MAPPING_ SKIP_MISSING	False	When mapping objects if one or more elements are missing in either the source or target object, the mapper generates an error when FLOGO_MAPPING_SKIP_MISSING is set to false. Set this environment variable to true, if you would like to return a null instead of receiving an error.
FLOGO_APP_ METRICS_LOG_ EMITTER_ENABLE	False	If you set this property to True, the app metrics are displayed in the logs with the values set in FLOGO_APP_METRICS_LOG_EMITTER_CONFIG. App metrics are not displayed in the logs if this

Environment Variable Name	Default Values	Description
		environment variable is set to False. To set it to True, run: <code>export FLOGO_APP_METRICS_LOG_EMITTER_ENABLE=true</code>
FLOGO_APP_METRICS_LOG_EMITTER_CONFIG	Both flow and Activity	<p>This property can be set to either flow level or Activity level. Depending on which level you set, the app metrics displays only for that level. Also, you can provide an interval (in seconds) at which to display the app metrics.</p> <p>For example, to set the interval to 30 seconds and get the app metrics for the flow, run:</p> <pre>export FLOGO_APP_METRICS_LOG_EMITTER_CONFIG='{ "interval": "30s", "type": ["flow"] }'</pre> <p>To set the interval for 10 seconds and get the app metrics for both flow and activities, run:</p> <pre>export FLOGO_APP_METRICS_LOG_EMITTER_CONFIG='{ "interval": "30s", "type": ["flow", "Activity"] }'</pre>
FLOGO_APP_METRICS	70	Enables app metrics on the Monitoring tab.
FLOGO_APP_MEM_ALERT_THRESHOLD	70	The threshold for memory utilization of the app. When the memory utilization by an app running in a container exceeds the threshold that you have specified, you get a warning log
FLOGO_APP_CPU_	70	The threshold for CPU utilization of the

Environment Variable Name	Default Values	Description
ALERT_THRESHOLD		app. When the CPU utilization by an app running in a container exceeds the threshold that you have specified, you get a warning log
FLOGO_APP_DELAYED_STOP_INTERVAL	10 seconds	<p>When you scale down an instance, all inflight jobs are lost because the engine is stopped immediately. To avoid losing the jobs, delay the stopping of the engine by setting the FLOGO_APP_DELAYED_STOP_INTERVAL variable to a value less than 60 seconds. Here, when you scale down the instance, if there are no inflight jobs running, then the engine stops immediately without any delay. In case of inflight jobs:</p> <ul style="list-style-type: none"> • If there are any inflight jobs running, then the engine stops immediately after the inflight job is completed. • If the inflight job is not completed within a specified time interval, then the job gets canceled and the engine stops.
FLOGO_APP_DELAYED_STOP_INTERVAL	10 seconds	<p>When you scale down an instance, all inflight jobs are lost because the engine is stopped immediately. To avoid losing the jobs, delay the stopping of the engine by setting the FLOGO_APP_DELAYED_STOP_INTERVAL variable to a value less than 60 seconds. Here, when you scale down the instance, if there are no inflight jobs running, then the engine stops immediately without any delay. In</p>

Environment Variable Name	Default Values	Description
		<p>case of inflight jobs:</p> <ul style="list-style-type: none"> • If there are any inflight jobs running, then the engine stops immediately after the inflight job is completed. • If the inflight job is not completed within a specified time interval, then the job gets canceled and the engine stops.
GOGC	100	<p>Sets the initial garbage collection target percentage.</p> <p>Setting it to a higher value delays the start of a garbage collection cycle until the live heap has grown to the specified percentage of the previous size.</p> <p>Setting it to a lower value causes the garbage collector to be triggered more often as less new data can be allocated to the heap before triggering a collection.</p>

This section lists the user-defined environment variables that are associated with the Flogo Enterprise runtime environment.

Environment Variable Name	Default Values	Description
FLOGO_MAPPING_OMIT_NULLS	True	Used to omit all the keys in the activity input evaluating to null.
FLOGO_FLOW_CONTROL_	False	If you set FLOGO_FLOW_CONTROL_

Environment Variable Name	Default Values	Description
EVENTS		EVENTS as true, the Flow limit functionality is enabled, whenever the incoming requests to trigger reaches FLOGO_ RUNNER_QUEUE_ SIZE limit then trigger is paused. When all the requests currently under processing are finished, the trigger is resumed again. All the connectors supporting the Flow limit functionality are mentioned in their respective user guides.
FLOGO_HTTP_ SERVICE_PORT	N/A	Used to set the port number to enable runtime HTTP service, which provides APIs for healthcheck and statistics.
FLOGO_LOG_ FORMAT	TEXT	Used to switch the logging format between text and JSON. For example, to use the JSON format, set FLOGO_LOG_FORMAT=JSON <i>./<app-name></i>
FLOGO_MAX_ STEP_COUNT	N/A	The application stops processing requests after the FLOGO_MAX_ STEP_COUNT limit is reached. The default limit is set to 10 Million even when you do not add this variable.
FLOGO_ EXPOSE_ SWAGGER_EP	False	If you set this property to True, the Swagger endpoint is exposed. The Swagger of the Rest trigger app can be accessed by hitting the swagger endpoint at <code>http://<service-</code>

Environment Variable Name	Default Values	Description
		<code>url>/api/v2/swagger.json.</code>
FLOGO_SWAGGER_EP	N/A	<p>To customize the URI for the Swagger endpoint, set this environment variable to your desired endpoint.</p> <p>For example: <code>FLOGO_SWAGGER_EP=/custom/swagger/endpoint</code></p> <p>This makes the Swagger endpoint available at <code>/custom/swagger/endpoint</code> instead of the default <code>/api/v2/swagger.json.</code></p>
FLOGO_OTEL_SPAN_KIND	INTERNAL	<p>Used to specify the type of span to be used in OpenTelemetry. The supported values are INTERNAL, SERVER, CLIENT, PRODUCER, and CONSUMER.</p> <div style="border: 1px solid #ccc; background-color: #f9f9f9; padding: 5px; margin-top: 10px;"> <p>Note: If no value or an invalid value is provided, the default value will be set to INTERNAL.</p> </div>
FLOGO_LOG_CONSOLE_STREAM	stderr	Used to specify the logging output stream for Flogo engine and app logs. The supported values are <code>stdout</code> and <code>stderr</code> .

Pushing Apps to TIBCO Cloud

You can push apps that were created in Flogo Enterprise to TIBCO Cloud Integration using the TIBCO Cloud - Command Line Interface (`tibcli`).

You must download the TIBCO Cloud Integration artifacts to use TIBCO Cloud CLI to push the apps.

Before you begin

You must have the TIBCO Cloud CLI installed on your local machine before you follow this procedure. Refer to the "Downloading TIBCO Cloud Integration Tools" and "Installing the TIBCO® Cloud - Command Line Interface" sections in the TIBCO Cloud Integration documentation for details on how to download the TIBCO Cloud CLI and install it.

i Note: For REST apps, be sure to change the port to 9999 *before* downloading the artifacts.

To push the app using the TIBCO Cloud CLI, follow this procedure:

Procedure

1. On the app details page, click **Export**.
2. Select **TIBCO Cloud Integration artifacts**.


The `manifest.json` and `flogo.json` files are downloaded. The `manifest.json` contains the manifest details such as the endpoints, memory resource details, and so on. The `flogo.json` contains the app itself. These are the artifacts needed to push the app directly from Flogo Enterprise using TIBCO Cloud CLI.

3. Create a temporary directory on your machine.
4. Move the downloaded `flogo.json` and the `manifest.json` files into a temporary directory.

i Note: The `tibcli` or `tibcli.exe` executable should not be in the same directory (the temporary directory you created) as the app you are pushing.

5. Open a terminal or command prompt and navigate to the temporary directory.
6. Run the following command to push the app:

```
tibcli app push <app-name>
```

 **Important:** If there is an existing app with an identical name as the app that you are trying to push to the cloud, the existing app is overwritten with the newly pushed app. You do not get a warning about it.

Result

The app is pushed to TIBCO Cloud Integration. You can see the progress of the app push on the UI. After the app is pushed, the app implementation details on the **Flowtab** are replaced with the actual flow.

Best Practices

For efficient development of Flogo apps, follow these best practices:

Development

Flow Design

- **Re-use with subflows**

If you are executing the same set of activities within multiple flows of the Flogo app, you should put them in a subflow instead of adding the same logic in multiple flows again and again. For example, error handling and common logging logic.

Sub-flows can be called from other flows, thus enabling the logic to be reused. A subflow does not have a trigger associated with it. It always gets triggered from another flow within the same app.

- **Terminate the flow execution using a Return Activity**

Add a **Return** Activity at the end of the flow, when you want to terminate the flow execution and the flow has some output that needs to be returned to either the trigger (in the case of REST flows) or the parent flow (in the case of a branch flow). An Error Handler flow must also have a **Return** Activity at the end.

- **Copying a flow or an Activity**

In scenarios where you want to create a flow or an Activity that is very similar to an existing flow in your app, you can do so by duplicating the existing flow, then making your minimal changes to the flow duplicate. You need not create a new flow. For details on how to duplicate a flow, see [Duplicating a Flow](#). You can also copy activities. For details on how to copy an Activity, see [Duplicating an Activity](#).

- **Use of ConfigureHTTPResponse Activity**

If you define a response code in your REST trigger, **ReceiveHTTPMessage**, configure the return value for the response code in the **ConfigureHTTPResponse** Activity.

The **Return** Activity is a generic Activity to return data to a trigger. However, when developing a REST/HTTP API, you might need to use different schema for different HTTP response codes. You can configure the **ReceiveHTTPMessage** trigger to use

different schema for different response codes by either using the Swagger 2.0 or OpenAPI 3.0 specification or manually adding them to the trigger configuration.

In such a scenario, you should add the **ConfigureHTTPResponse** Activity in the flow before the **Return** Activity, to construct the response data for a specific response code. **ConfigureHTTPResponse** Activity allows you to select a response code, generate the input based on the schema defined on the trigger for that code, and map data from the upstream activities to the input.

You can then map the output of the **ConfigureHTTPResponse** Activity to the **Return** Activity to return the data and response code.

When you call a REST API from a Flow using the **InvokeRESTService** Activity, you can enable the 'Configure Response Codes' option to handle the response codes returned by the API. You can add specific codes, for example, 200, 404, and define a schema for each of them using this option. You can also define the status code range in a format such as 2xx if the same schema is being used for all codes in that range.

- **Reserved keywords**

Flogo Enterprise uses some words as keywords or reserved names. Do not use these words in your schema. For a complete list of the keywords to be avoided, see [Reserved Keywords to be Avoided in Schemas](#).

Mapper

- **Synchronizing schema**

If you make any changes to the trigger configuration after the trigger was created, you must click **Sync** for the schema changes to be propagated to the flow parameters. For more information, see [Synchronizing Schema Between Trigger and Flow](#).

- **Using Expressions and Functions**

Within any one flow, use the mapper to pass data between the activities, between the trigger and the activities, or the trigger and the flow. When mapping, you can use data from the following sources:

- **Literal values** - Literal values can be strings or numeric values. These values can either be manually typed in or mapped to a value from the output of the trigger or a preceding Activity in the same flow. To specify a string, enclose the string in double-quotes. To specify a number, type the number into the text box for the field. Constants and literal values can also be used as input to

functions and expressions.

- Direct mapping of an input element to an element of the same type in the Upstream Output.
 - Mapping using functions - The mapper provides commonly used functions that you can use in conjunction with the data to be mapped. The functions are categorized into groups. Click a function to use its output in your input data. When you use a function, placeholders are displayed for the function parameters. You click a placeholder parameter within the function, then click an element from the Upstream Output to replace the placeholder. Functions are grouped into logical categories. For more information, see [Using Functions](#).
 - Expressions - You can enter an expression whose evaluated value is mapped to the input field. For more information, see [Using Expressions](#).
- **Complex data mappings**
 - Using the `array.forEach()` mapper function, you can map complex nested arrays, filter elements of an array based on a condition and map array elements to non-array elements or elements of another array with a different structure. See the following sections for details:
 - [Mapping complex arrays - Using the array.forEach\(\) Function](#)
 - [Mapping Array Child Elements to Non-Array Elements or to an Element in a Non-Matching Array](#)
 - [Filtering Array Elements to Map Based on a Condition](#)
 - [Mapping an Identical Array of Objects](#)
 - You can extract a particular element from a complex JSON object. The `json.path()` function takes JSONPath expression as an argument. JSONPath is an XPATH like query language for querying an element from JSON data. Refer to [Using the json.path\(\) function](#) for more details.

Branches

- **Branch conditions**

You can design conditional flows by creating one or more branches from an Activity and defining the branch types as well as the conditions for executing these branches. Refer to the [Creating a Flow Execution Branch](#) section for details on how to create

branches, the type of branches you can create, and the order in which the branches get executed in a flow.

Error handling

Errors can be handled at the Activity level or at the flow level. To catch errors at the Activity level, use an error branch. In this case, the flow control transfers to the main branch when there is an error during Activity execution. Refer to the section, [Catching Errors](#) for more details on error handling. To catch errors at the flow level (when you want to catch all errors during the flow execution regardless of the activities from which the errors are thrown), use the Error Handler at the bottom left on the flow page to create an error flow. Since this flow must have a **Return** Activity at the end, the flow execution gets terminated after the Error Handler flow executes. The control never goes back to the main flow. Refer to the section, [Catching Errors](#), for more details.

To handle network faults, Flogo Enterprise provides the ability to configure the Timeout and Retry on Error settings for some specific activities such as **InvokeRESTService** and **TCCMessagePublisher**. Refer to the "General Category Triggers and Activities" section of the *TIBCO Flogo® Enterprise Activities, Triggers, and Connections Guide* for details on each **General** category Activity and trigger.

Deployment and Configuration

Memory considerations

When Flogo apps are deployed in TIBCO Cloud™ Integration, keep in mind that a maximum 1GB of memory is allocated to each app instance. If the Flogo app flow execution is memory heavy, the container is stopped due to lack of required memory and the following error message is displayed:

```
502 Bad Gateway Error
```

Using environment variables

When deploying a Flogo app, you can override the values of the app properties using environment variables. For details on using environment variables, see the section on [Environment Variables](#).

Externalize configuration using app properties

When developing Cloud-Native microservices, we recommend that you separate the configuration from the app logic. You should avoid hard-coding values for configuration parameters in the Flogo app and use the app properties instead.

The use of app properties allows you to externalize the app configuration. Externalizing the configuration in turn allows you to change the value for any property without having to update the app. This is particularly useful when testing your app with different configurations and automating deployments across multiple environments as part of the CI/CD strategy configurations and automating deployments across multiple environments as part of the CI/CD strategy. For details on using app properties, see the section, [App Properties](#).

Generating and using SSL certificates

When generating an SSL certificate, it is recommended that you use Public DNS as a Common Name. Also, when using an SSL certificate, use Public DNS instead of IP address.

Building Engine binary

For multiple apps that have a common set of functionality, you can build a generic Flogo Enterprise binary instead of building a separate binary for each app.

Performance Tuning

This section provides guidelines that can be used to understand your performance objectives and fine-tune the app environment to optimize performance.

The performance of an app affects stability, scalability, throughput, latency, and resource utilization. For optimal performance of the app, it is important to understand the various levels at which the tuning methods and best practices can be applied to the components. This section includes the different tuning parameters, steps required to configure the parameters, and design techniques for better performance.

This section must be used along with other product documentation and project-specific information to achieve the desired performance results. The goal is to assist in tuning and optimizing the runtime for the most common scenarios. At the same time, one must focus on real-life scenarios to understand the issue and the associated solution.

i Note: The performance tuning and configurations in this section are provided for reference only. They can be reproduced only in the exact environment and under workload conditions that existed when the tests were done. The numbers in the document are based on the tests conducted in the performance lab and may vary according to the components installed, the workload, the type and complexity of different scenarios, hardware, and software configuration, and so on. The performance tuning and configurations should be used only as a guideline, after validating the customer requirements and environment. TIBCO does not guarantee its accuracy.

Tuning Environment Variables

This section lists the environment variables associated with the TIBCO Flogo environment. Details such as the default value of environmental variables and how we can change them are also included.

FLOGO_RUNNER_TYPE

This variable defines how events are handled by the Flogo engine.

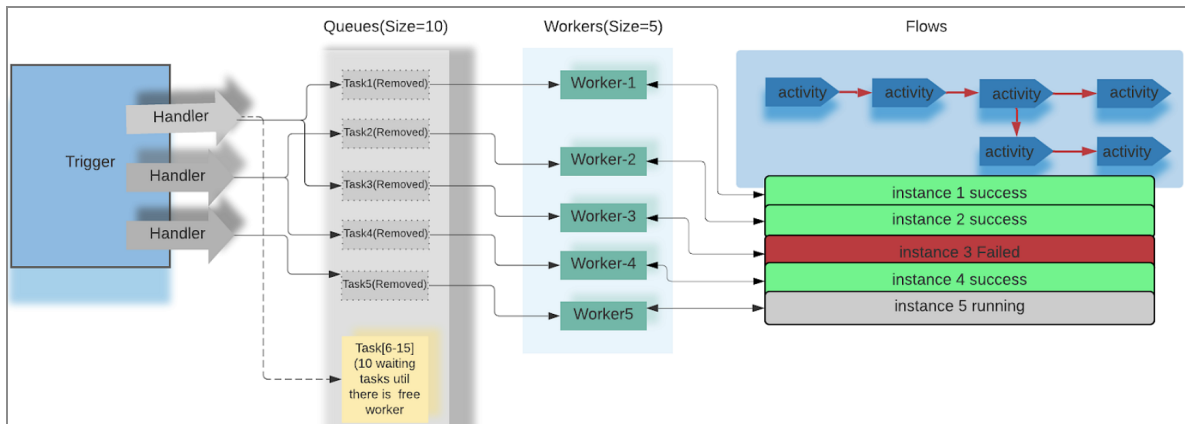
- Supported values: DIRECT and POOLED.
- Default: POOLED

POOLED Mode

In this mode, the engine handles events in a flow-controlled way.

The following pictorial diagram explains the handling of events in POOLED mode.

Events in POOLED mode



Sets of workers are created to handle events received by all the triggers in the given Flogo app. In golang terms, one worker corresponds to one go-routine. The events received are added to the worker queue before the workers can pick these events from the worker queue.

Once an event is picked from the queue, the corresponding action (for example, flow) is triggered and the worker continues to execute that action until completion (that is, until the action is successful or fails). An event that is picked up from the queue is removed to allow the next event to be added to the queue.

When the queue is full, all trigger handlers that are adding new events to the queue are blocked until workers pick up the next set of events from the queue. Once the worker starts executing the action, it never interleaves the action until its completion. So, the total number of events processed at a time is directly proportional to the time taken by the action to complete and the number of workers in the pool. Hence, for better concurrency,

gradually increase the value of queues and workers based on the available compute resources (such as CPU and memory).

Configurations in POOLED mode:

You can configure the workers and the queue size by setting FLOGO_RUNNER_WORKERS and FLOGO_RUNNER_QUEUE_SIZE respectively.

- FLOGO_RUNNER_WORKERS variable determines the maximum number of concurrent events that can be executed by the app engine from the queue. FLOGO_RUNNER_WORKERS execute a finite number of tasks or concurrent events uninterrupted and then yield to the next ready job. FLOGO_RUNNER_WORKERS can be tuned to the optimum value by starting with a default value set and increasing it as per requirement until the maximum CPU is reached.

The default value is FLOGO_RUNNER_WORKERS=5.

- FLOGO_RUNNER_QUEUE_SIZE variable specifies the maximum number of events from all triggers that can be queued by the app engine. FLOGO_RUNNER_QUEUE_SIZE can be tuned to the optimum value by starting with a default value set and increasing it as per requirement. You can change the variable value if you anticipate having more than default value events queued at the same time.

The default value is FLOGO_RUNNER_QUEUE_SIZE=50.

The CPU and memory resources must be measured under a typical processing load to determine if the default variable value is suitable for the environment. If the user load is more than the default set value, the user can change the runner worker variable as per the requirement to expedite the execution of the concurrent events. Set variable values according to your processing volumes, number of CPUs, and allocated memory.

Deploying the app to your environment

Set the variable value as follows:

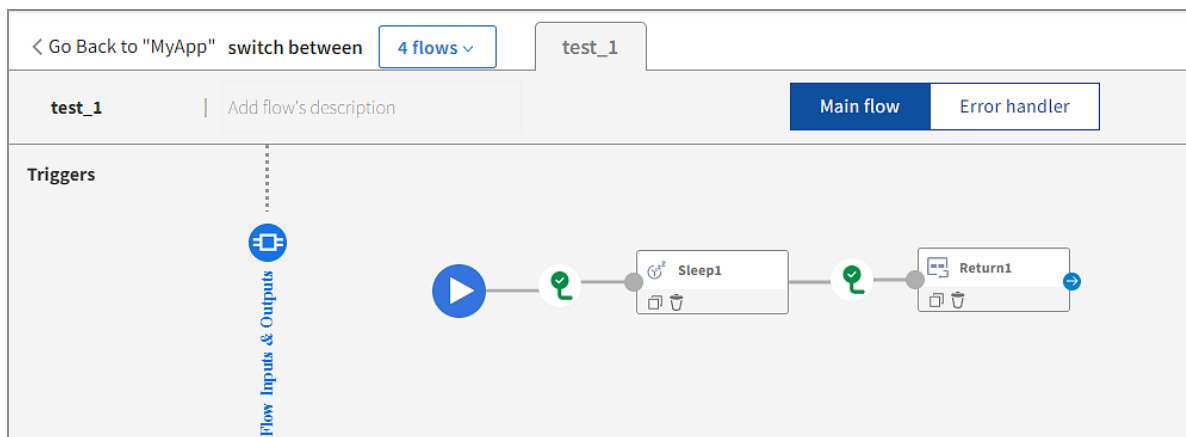
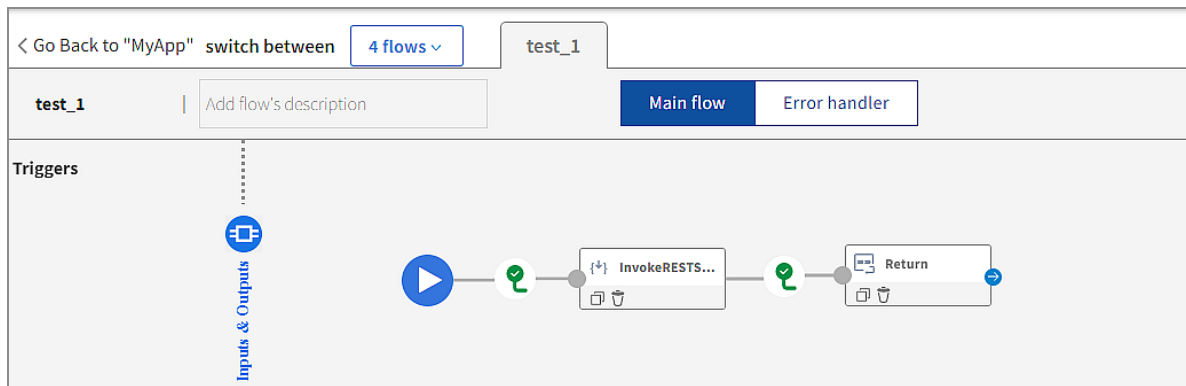
```
FLOGO_RUNNER_WORKERS=75 FLOGO_RUNNER_QUEUE_SIZE =150 ./<app_binary>
```

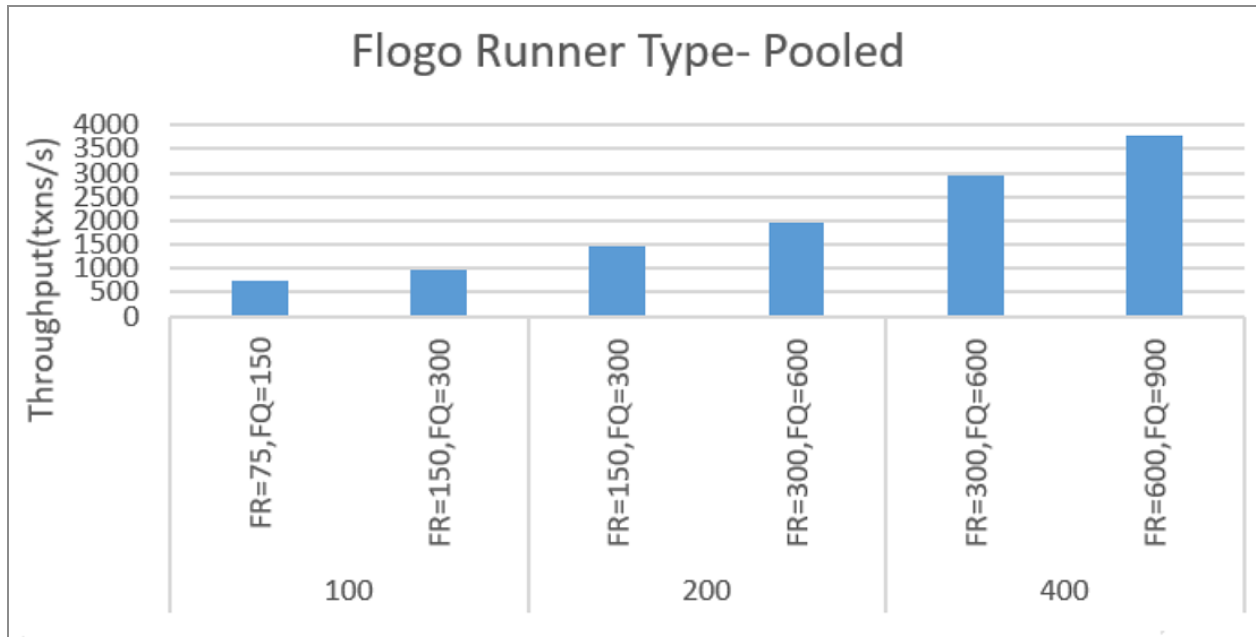
```
docker run -it -e FLOGO_RUNNER_WORKERS=75 -e FLOGO_RUNNER_QUEUE_SIZE=150  
<docker-image>
```

Case Study

While setting up the FLOGO_RUNNER_TYPE as POOLED, Flogo runner workers and Flogo runner queues are used to handling events received by the trigger. You can increase the Flogo runner worker and queue values gradually to reach the app performance. Set variable values according to your processing volumes concerning your number of CPUs and allocated memory.

It is recommended that you set the queue size greater than or equal to the number of workers.

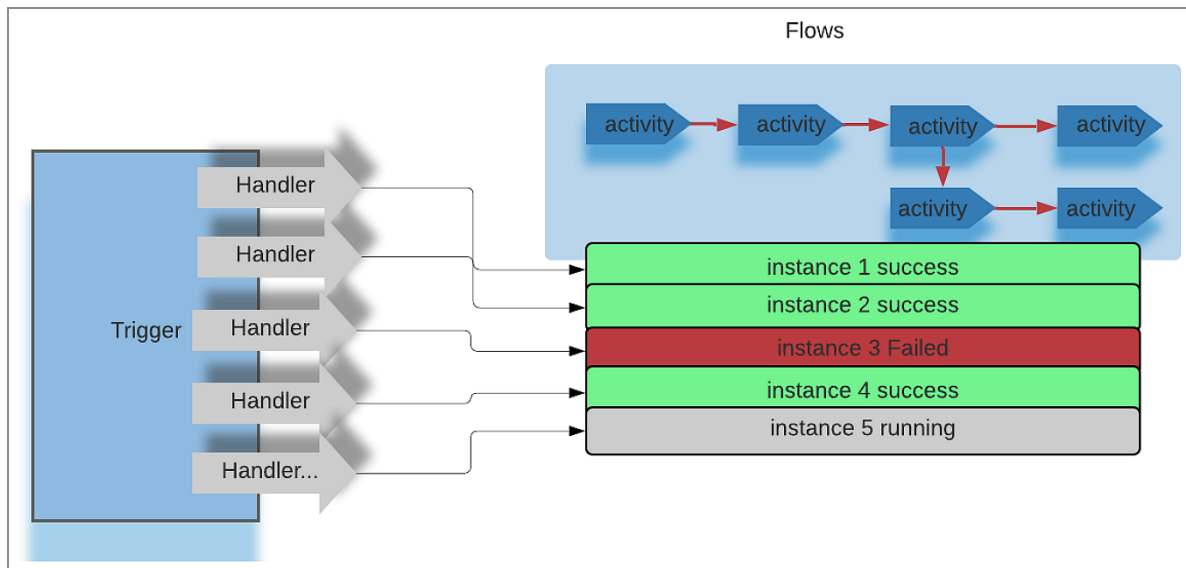




DIRECT Mode

In this mode, every event delivered by the handler triggers a corresponding action. Unlike the POOLED mode, the handling of events is unbounded. All the events are processed concurrently. This might lead to CPU saturation or out-of-memory errors.

The following pictorial diagram explains the handling of events in DIRECT mode.



Deploying the app to your environment

Set the variable value as follows:

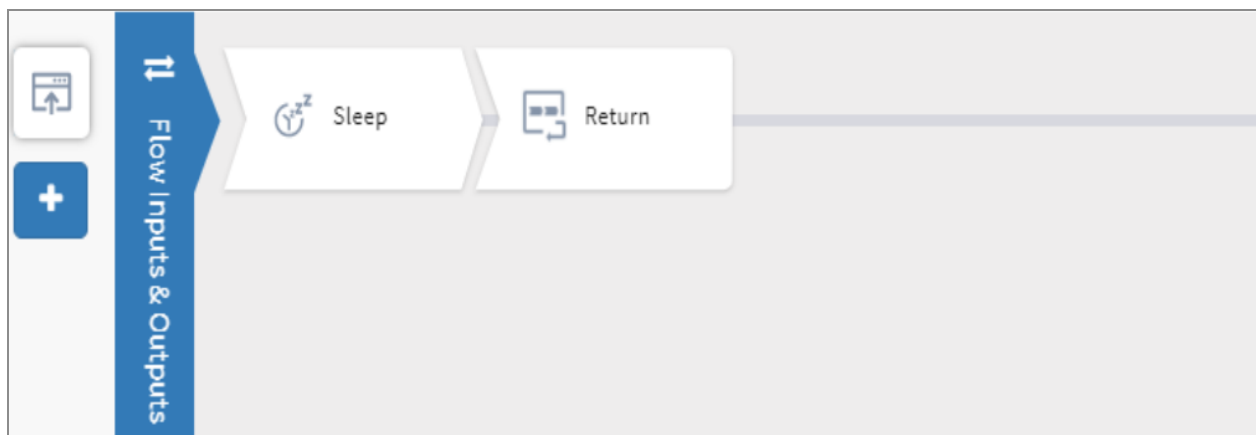
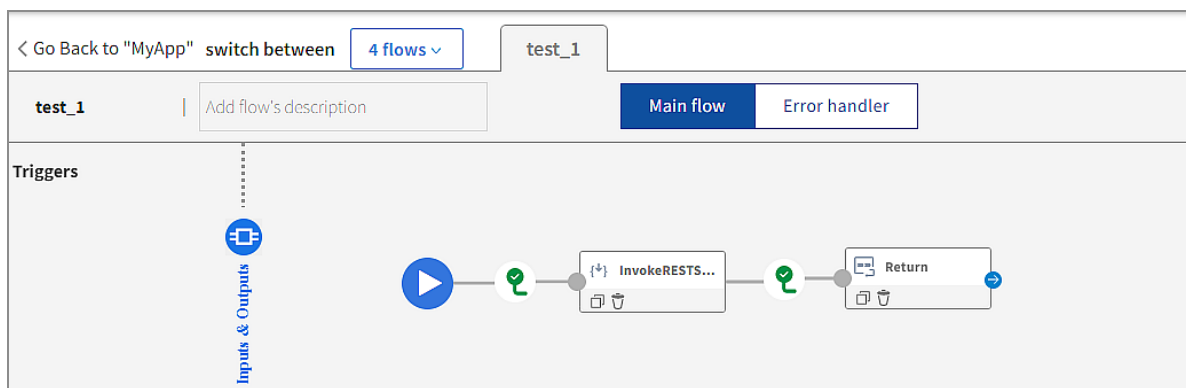
```
FLOGO_RUNNER_TYPE=DIRECT ./<app_binary>
```

```
docker run -it -e FLOGO_RUNNER_TYPE=DIRECT <docker-image>
```

Case Study

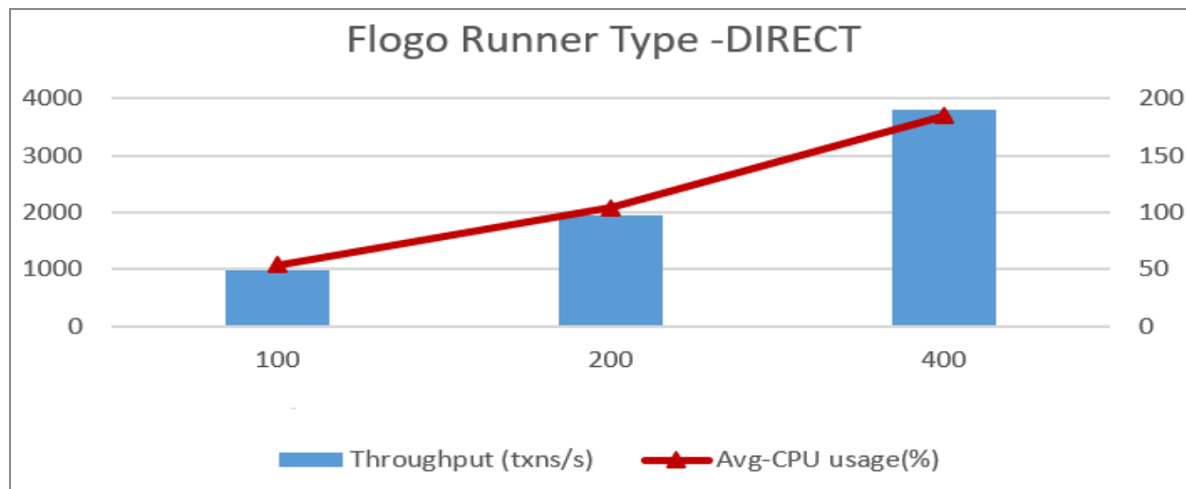
This case study illustrates the app performance when Flogo event handling mode is set to DIRECT.

App under test - FLOGO_RUNNER_TYPE



While setting up the FLOGO_RUNNER_TYPE as DIRECT, all the events sent to the trigger are processed concurrently. As you keep on increasing the concurrency, you can observe the linear increase in resources, that is, CPU and memory utilization.

Flogo Engine - Direct Mode



FLOGO_LOG_LEVEL

This environment variable is used to set a log level for an app.

- Supported values: INFO, DEBUG, WARN, and ERROR.
- Default: INFO

You can increase or decrease the logging of the app using this environment variable. To increase the logging of the app to debug, change FLOGO_LOG_LEVEL to DEBUG. To skip detailed logging and to just log an error, set FLOGO_LOG_LEVEL to ERROR. Changes to the log level are reflected after restarting the Flogo app in your environment and by pushing the Flogo app again to the cloud environment.

Deploying the app to your environment

Set the variable value as follows:

```
FLOGO_LOG_LEVEL=ERROR ./<app_binary>
```

```
docker run -it -e FLOGO_LOG_LEVEL=ERROR <docker-image>
```

Log level - ERROR

```
[ec2-user@ip-172-31-9-14 products]$ FLOGO_LOG_LEVEL=ERROR ./Rest-Invoke-server-linux_amd64
TIBCO Flogo® Runtime - 2.11.0 (Powered by Project Flogo™ - v1.2.0)
TIBCO Flogo® connector for General - 1.2.0.455
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Starting TIBCO Flogo® Runtime
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Runtime started in 1.658066ms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Log level - INFO

```
[ec2-user@ip-172-31-9-14 products]$ FLOGO_LOG_LEVEL=INFO ./Rest-Invoke-server-linux_amd64
TIBCO Flogo® Runtime - 2.11.0 (Powered by Project Flogo™ - v1.2.0)
TIBCO Flogo® connector for General - 1.2.0.455
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Starting TIBCO Flogo® Runtime
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2021-01-11T07:30:12.720Z WARN [flogo] - unable to create child logger named: ReceiveHTTPMessage - unable to create child logger
2021-01-11T07:30:12.720Z INFO [general-trigger-rest] - Name: ReceiveHTTPMessage, Port: 9999
2021-01-11T07:30:12.720Z INFO [general-trigger-rest] - ReceiveHTTPMessage: Registered handler [Method: POST, Path: /echo]
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Starting app [ Rest-Invoke-server ] with version [ 1.1.0 ]
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Engine Starting...
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Starting Services...
2021-01-11T07:30:12.720Z INFO [flogo] - ActionRunner Service: Started
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Started Services
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Starting Application...
2021-01-11T07:30:12.720Z INFO [flogo] - Starting Triggers...
2021-01-11T07:30:12.720Z INFO [general-trigger-rest] - Starting ReceiveHTTPMessage...
2021-01-11T07:30:12.720Z INFO [general-trigger-rest] - Started ReceiveHTTPMessage
2021-01-11T07:30:12.720Z INFO [flogo] - Trigger [ ReceiveHTTPMessage ]: Started
2021-01-11T07:30:12.720Z INFO [flogo] - Triggers Started
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Application Started
2021-01-11T07:30:12.720Z INFO [flogo.engine] - Engine Started
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Runtime started in 2.956203ms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

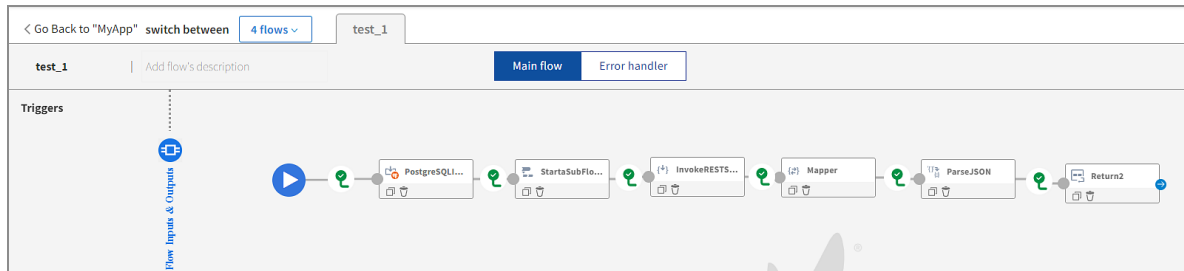
Log level - DEBUG

```
2021-01-11T07:32:25.034Z DEBUG [flogo] - Resolved function 'git.tibco.com/git/product/ipaas/wi-contrib.git/function/datetime:create' to
'datetime.create'
2021-01-11T07:32:25.034Z DEBUG [flogo.activity.actreturn] - Mappings: map[code:200 data:map[mapping:map[data=$flow.body.data]]]
2021-01-11T07:32:25.034Z WARN [flogo] - unable to create child logger named: ReceiveHTTPMessage - unable to create child logger
2021-01-11T07:32:25.034Z DEBUG [general-trigger-rest] - In init, id 'ReceiveHTTPMessage'
2021-01-11T07:32:25.034Z INFO [general-trigger-rest] - Name: ReceiveHTTPMessage, Port: 9999
2021-01-11T07:32:25.034Z INFO [general-trigger-rest] - ReceiveHTTPMessage: Registered handler [Method: POST, Path: /echo]
2021-01-11T07:32:25.034Z DEBUG [flogo.engine] - Creating app [ Rest-Invoke-server ] with version [ 1.1.0 ]
2021-01-11T07:32:25.034Z INFO [flogo.engine] - Starting app [ Rest-Invoke-server ] with version [ 1.1.0 ]
2021-01-11T07:32:25.034Z INFO [flogo.engine] - Engine Starting...
2021-01-11T07:32:25.034Z INFO [flogo.engine] - Starting Services...
2021-01-11T07:32:25.034Z DEBUG [flogo] - ActionRunner Service: Starting...
2021-01-11T07:32:25.034Z DEBUG [flogo] - Starting worker with id '1'
2021-01-11T07:32:25.034Z DEBUG [flogo] - Starting worker with id '2'
2021-01-11T07:32:25.035Z DEBUG [flogo] - Starting worker with id '3'
2021-01-11T07:32:25.035Z DEBUG [flogo] - Starting worker with id '4'
2021-01-11T07:32:25.035Z DEBUG [flogo] - Starting worker with id '5'
2021-01-11T07:32:25.035Z INFO [flogo] - ActionRunner Service: Started
2021-01-11T07:32:25.035Z INFO [flogo.engine] - Started Services
2021-01-11T07:32:25.035Z INFO [flogo.engine] - Starting Application...
2021-01-11T07:32:25.035Z INFO [flogo] - Starting Triggers...
2021-01-11T07:32:25.035Z DEBUG [flogo] - Trigger [ ReceiveHTTPMessage ]: Starting...
2021-01-11T07:32:25.035Z INFO [general-trigger-rest] - Starting ReceiveHTTPMessage...
2021-01-11T07:32:25.035Z INFO [general-trigger-rest] - Started ReceiveHTTPMessage
2021-01-11T07:32:25.035Z INFO [flogo] - Trigger [ ReceiveHTTPMessage ]: Started
2021-01-11T07:32:25.035Z DEBUG [flogo] - Trigger [ ReceiveHTTPMessage ] has ref [ git.tibco.com/git/product/ipaas/wi-contrib.git/contrib
utions/General/trigger/rest ] and version [ ]
2021-01-11T07:32:25.035Z INFO [flogo] - Triggers Started
2021-01-11T07:32:25.035Z INFO [flogo.engine] - Application Started
2021-01-11T07:32:25.035Z INFO [flogo.engine] - Engine Started
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Runtime started in 2.593609ms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Case Study

This use case illustrates the app logging impact on the performance of the app.

App under test for Flogo Log level



Performance lab results have shown that the performance of the app depends on the app log level that is set, request payload, and app latency. Set the log level to DEBUG for functional issues and to ERROR for performance scenarios because setting the logging to DEBUG might impact the performance of the app.

Maximum throughput was achieved with a Log Level set as ERROR.

GOGC

The GOGC variable sets the initial garbage collection target percentage. A collection is triggered when the ratio of freshly allocated data to live data remaining after the previous collection reaches this percentage.

Garbage collection refers to the process of managing heap memory allocation: free the memory allocations that are no longer in use and keep the memory allocations that are being used. Garbage collection significantly affects the performance of your app.

Deploying the app to your environment

Set the variable value as follows:

```
GOGC=150 ./<app_binary>
```

```
docker run -it -e GOGC=150 <docker-image>
```

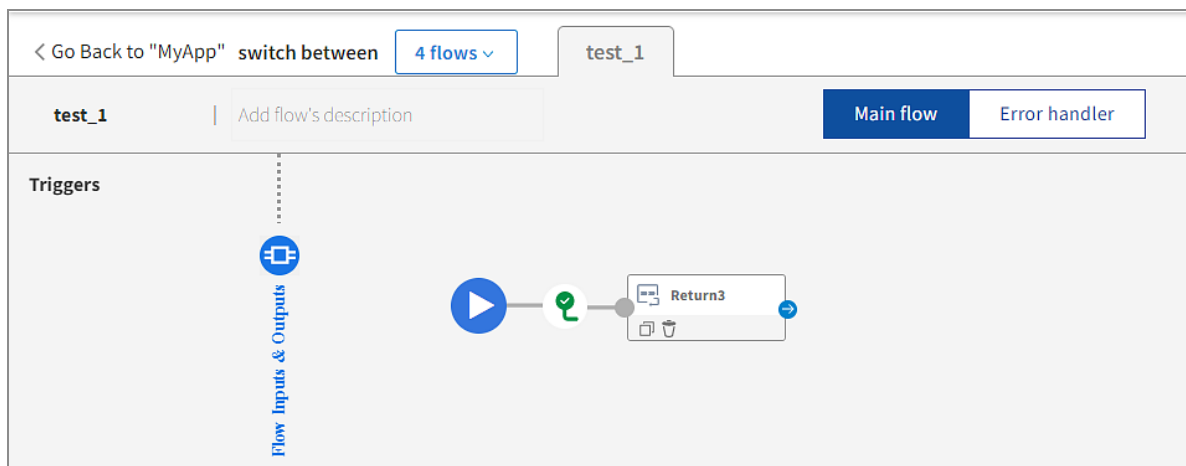
The default is 100. This means that garbage collection is not triggered until the heap has grown by 100% since the previous collection. Setting the variable to a higher value (for example, GOGC=200) delays the start of a garbage collection cycle until the live heap has

grown to 200% of the previous size. Setting the variable to a lower value (for example, GOGC=20) increases the frequency of garbage collection as less new data can be allocated on the heap before triggering a collection.

Case Study

This use case illustrates the impact of the GOGC variable on performance.

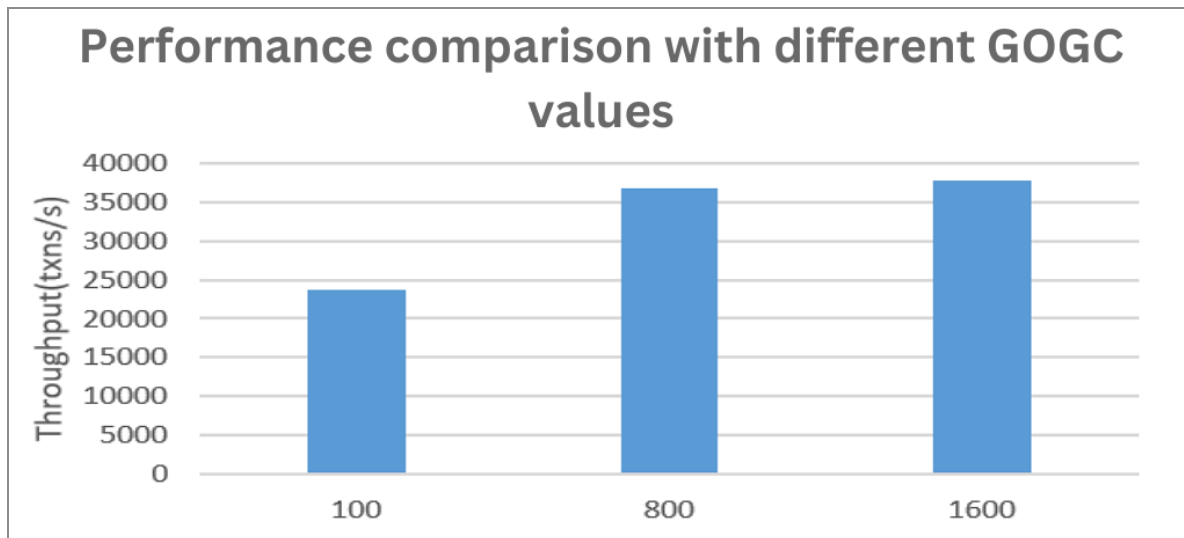
App under test - GOGC



In this low latency scenario, you can observe significant improvement in-app performance while increasing the GOGC variable value from 100 to 1600. It is advisable to test this value for the specific scenario and understand its impact before tuning. You can get the best-suited value by running the performance test in your test environment.

GOGC value can be tuned based on the workload and available resources after validating your test environment.

Performance comparison with different GOGC values



Flow Limit

Flow limit is useful when the engine needs to be throttled, as the `FLOGO_RUNNER_QUEUE_SIZE` engine variable specifies the maximum number of events that can be started before pausing the process trigger. This ensures that the incoming requests do not overwhelm the engine performance and the CPU and memory is preserved.

If the number of incoming requests on the trigger exceeds the `FLOGO_RUNNER_QUEUE_SIZE` limit, the engine pauses the trigger to get any new requests, but continues running the existing ones. The engine resumes this trigger when all the requests currently under processing are finished.

And this flow limit is not enforced by the engine unless the `FLOGO_FLOW_CONTROL_EVENTS` variable is set to `true` for an application as a user-defined **Engine Variable** on the **Environment Variable** tab of the Flogo Enterprise runtime environment.

Environment variables associated with Flow Limit

Environment Variable Name	Default Values	Description
<code>FLOGO_RUNNER_QUEUE_SIZE</code>	50	The maximum number of events from all triggers that can be

Environment Variable Name	Default Values	Description
		queued by the app engine.
FLOGO_RUNNER_WORKERS	5	The maximum number of concurrent events that can be run by the app engine from the queue.
FLOGO_FLOW_CONTROL_EVENTS	N/A	If you set FLOGO_FLOW_CONTROL_EVENTS as true, the Flow limit functionality is enabled, whenever the incoming requests to trigger reaches FLOGO_RUNNER_QUEUE_SIZE limit then trigger is paused. When all the requests currently under processing are finished, the trigger is resumed again.

i Note: All the connectors supporting the flow limit functionality are mentioned in their respective user guides.

CPU and Memory Monitoring

Top Command

i Note: The top command works on Linux platforms only.

The top command is used for memory and CPU monitoring.

The top command produces an ordered list of running processes selected by user-specified criteria. The list is updated periodically. By default, ordering is by CPU usage and it shows the processes that consume maximum CPU. The top command also shows how much

processing power and memory are being used, as well as the other information about the running processes.

The `top` command output monitors the memory as well as the CPU utilization of the TIBCO Flogo app binary.

The sample output is as follows:

```
top -p PID > top.txt
Cpu(s):  4.7%us,  1.1%sy,  0.0%ni, 94.1%id,  0.0%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:   65914304k total, 59840516k used,  6073788k free,  3637208k buffers
Swap: 15359996k total,  119216k used, 15240780k free, 43597120k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+
COMMAND
```

Docker Stats Command

The `docker stats` command returns a live data stream for running containers. To limit data to one or more specific containers, specify a list of container names or ids separated by a space.

The `docker stats` command output monitors the memory as well as the CPU utilization of the TIBCO Flogo Enterprise app container and TCI Flogo app container.

- CPU % is the percentage of the host's CPU the container is using.
- MEM USAGE / LIMIT is the total memory the container is using and the total amount of memory, it is allowed to use.

```
# docker stats a3f78cb32a8e
CONTAINER ID   NAME           CPU %  MEM USAGE / LIMIT  MEM %  NET I/O  BLOCK  I/O  PIDS
a3f78cb32a8e  hello-world   0.00%  2.137MiB / 3.605GiB  0.06%  0B / 0B  9.95MB / 0B  0
```

Runtime Statistics and Profiling

The Go language provides CPU and memory profiling capabilities. With the profiling tools provided by Go, one can identify and correct the specific bottlenecks. You can make your app run faster and with less memory.

The `pprof` package writes runtime profiling data in the format expected by the `pprof` visualization tool. There are many commands available from the `pprof` command line. Commonly used commands include `top`.

For details about profiling, see the “Go Language Runtime Statistics and Profiling” section of *TIBCO Flogo® Enterprise User Guide*.

Samples

When creating apps in TIBCO Flogo® Enterprise, you can import and customize any of the predefined samples provided in the tci-flogo GitHub repository. These samples demonstrate how to develop, test, and deploy a Flogo app using various out-of-the-box capabilities. In the GitHub repository, the samples are organized by category and each sample folder contains a readme. Follow the instructions in the readme to import the sample to your local workspace and use it. The following samples are currently available:

Flow Design Concepts

Includes Hello World, Branching, Error Handling, Loops, Subflows, and Shared Data samples

API Development

Includes REST, GraphQL, and gRPC samples

Array Mapping and Filtering

Includes `array.forEach`, `json.path`, and JavaScript Activity samples

Connectors

Includes Flogo connector samples for CRM, DB Connectors, Messaging, and more

Serverless

Includes sample for deploying a Flogo app as an Azure function

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO Flogo® Enterprise Product Documentation](#) page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, and Flogo are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2016-2025. Cloud Software Group, Inc. All Rights Reserved.