Xcom

What is it

Xcom (short for “cross-communication”) is a way for us to pass small bits of information between our Airflow tasks. It is a great way to inform your downstream tasks about what has happened or what needs to happen.

For more information about Xcom, please read the official documentation.

How do I use it

Xcoms have two distinct mechanisms: pushing and pulling.

The information that can be passed through these mechanisms should be small and serializable python data types and data structures (string, integer, list, dict).

Trying to pass large amounts of data will lead to errors and is highly discouraged. It is recommended to stick to small strings or JSON data.

Pushing

By default, the xcom “push” of a task will be the return value of your operator (e.g. python_callable return value, last bash output, etc.) and will be identified by the `return_value` key.

Ex: The code below will push an xcom with key `return_value` and value `Hello, World!`.

```
def my_code():
  return 'Hello, World!'

```

```
xcom_push = PythonOperator(
  task_id='xcom_push',
  python_callable=my_code,
  ...
)
```

A task can also output multiple xcoms if it has access to the task_instance object.

Ex: The code below will output an xcom with the key `direct_push` and value `Bye!`, as well as the default xcom where key `return_value` and value `Hello, World!`

```
# The task instance object is passed in directly
def my_code(ti):
  ti.xcom_push(key='direct_push', value='Bye!')
  return 'Hello, World!'

xcom_push = PythonOperator(
  task_id='xcom_push',
  python_callable=my_code,
  ...
)
```

⚠ The code above only works for Python based Operators. You will not have access to the task instance object from Bash & Notebook operators.

Pulling

Xcom pulling is a bit more involved as we need to use Airflow's Jinja templating to fetch the xcom value and pass it to our operator.

In the code below, we pull the `return_value` xcom from the `xcom_push` task and pass it as a parameter to our python code :

```
def my_pull_code(xcom):
  print(f'Xcom: {xcom}')

xcom_push = PythonOperator(
  task_id='xcom_pull',
  python_callable=my_pull_code,
  op_kwargs={
    "xcom": "{{ ti.xcom_pull(task_ids='xcom_push', key='return_value') }}"
  },
  ...
)
```

The `xcom_push` task above will print `Xcom: Hello, World!`

If we want to pull both xcom values, we can do the following:

```
def my_pull_code(xcom_return, xcom_direct):
  print(f'Xcom_return: {xcom_return}')
  print(f'Xcom_direct: {xcom_direct}')

xcom_push = PythonOperator(
  task_id='xcom_pull',
  python_callable=my_pull_code,
  op_kwargs={
    "xcom_return": "{{ ti.xcom_pull(task_ids='xcom_push', key='return_value') }}",
    "xcom_direct": "{{ ti.xcom_pull(task_ids='xcom_push', key='direct_push') }}"
  },
  ...
)
```

Operators

Unfortunately, not all operators support xcom pushing by default. Below, you will find a list of the most common operators used in Hogwarts and their Xcom support:

Family	Operators	Supported
Python	• PythonOperator • HogwartsPythonOperator • GitPythonOperator	✓
Bash	• BashOperator • HogwartsBashOperator • GitBashOperator	✓
Notebook	• GitNotebookOperator • NbgalleryOperator	✓ (supported as of STRATUS 1.16.2)
Container	• ContainerOperator	✗

If you wish to use xcom with an operator that doesn’t support it or isn’t in the list above, let us know in the Hogwarts support chat.

Python family

The most straight forward operator to use with xcom.

Pushing

Python callable return value

- key: `return_value`
- value `<value_of_your_return>`

```
def code():
  return '<value_of_your_return>'

xcom_push = PythonOperator(
  task_id='xcom_push',
  python_callable=code,
  ...
)
```

Task instance object’s `xcom_push` method

- key: `<user_specified>`

- value: `<value_passed_in_method>`

```
def code(ti):
  ti.xcom_push(key='<user_specified>', value='<value_passed_in_method>')

xcom_push = PythonOperator(
  task_id='xcom_push',
  python_callable=code,
  ...
)
```

Pulling

Jinja templating through kwargs

```
def code(xcom):
  print(xcom)

xcom_pull= PythonOperator(
  task_id='xcom_pull',
  python_callable=code,
  op_kwargs={'xcom': '{{ ti.xcom_pull(task_ids="xcom_push", key="return_value") }}'},
  ...
)
```

Task instance object's `xcom_pull` method

```
def code(ti):
  print(ti.xcom_pull(task_ids="xcom_push", key="return_value"))

xcom_pull = PythonOperator(
  task_id='xcom_pull',
  python_callable=code,
  ...
)
```

Bash family

Pushing

Bash return value

- key: `return_value`
- value `<value_of_your_return>`

```
xcom_push = BashOperator(
  task_id='xcom_push',
  bash_command='echo "<value_of_your_return>"',
  ...
)
```

Pulling

```
```

```
xcom_pull = BashOperator(
  task_id='xcom_pull',
  bash_command='echo "{{ ti.xcom_pull(task_ids=\"xcom_push\", key=\"return_value\") }}"',
  ...
)
```

Notebook family

The following examples apply to all operators in the Notebook family (i.e. GitNotebookOperator, NbgalleryOperator)

Pushing

Last Notebook cell's output

Supported since `daggers==2.9.3.7553`

- key: `return_value`
- value `<value_of_your_return>`

```
xcom_push = GitNotebookOperator(
  task_id='xcom_push',
  repo_url='...',
  nb_in='notebooks/my_notebook.ipynb',
  do_xcom_push=True,
  ...
)
```

**Make sure to add `do_xcom_push=True` as it is disable by default for this operator family!

In your notebook's last cell, there can be two different output which are supported in Hogwarts :

- stream (stdout/print)
- execute_result (value returned from the cell)

To determine how your Notebook's output will look like, follow these simple steps:

- Execute your notebook on Jupyterhub and save it.
- In another notebook or python code, run the following code

```
from daggers.utils.notebook import extract_notebook_output

return_value = extract_notebook_output('<your_notebook_path_here.ipynb')
print(return_value)
```

The code above will read the notebook, parse it, and retrieve the last cell's last output. For more information on the output format of notebooks, take a look at the nbformat documentation.

Pulling

Use the same approach as the Python family operators and pass the templated xcom value to your `nb_params`.

```
xcom_pull = GitNotebookOperator(
  task_id='xcom_pull ',
  repo_url='...',
  nb_in='notebooks/my_notebook.ipynb',
  nb_params={'xcom': '{{ ti.xcom_pull(task_ids="xcom_push", key="return_value") }}'},
  ...
)
```

Make sure to add a cell with a `parameters` tag in the metadata of your notebook, otherwise these parameters won’t be injected at runtime.