Developing interactively with Apache Beam notebooks

ature is covered by the Pre-GA Offerings Terms (/terms/service-terms#1) of the Google Cloud Platform Terms
e. Pre-GA features may have limited support, and changes to pre-GA features may not be compatible with oth
sions. For more information, see the launch stage descriptions (/products#product-launch-stages).

Using the Apache Beam interactive runner with JupyterLab notebooks lets you iteratively develop pipelines, inspect your pipeline graph, and parse individual PCollections in a read-eval-print-loop (REPL) workflow. These Apache Beam notebooks are made available through AI Platform Notebooks (/ai-platform/notebooks/docs), a managed service that hosts notebook virtual machines pre-installed with the latest data science and machine learning frameworks.

This guide focuses on the functionality introduced by Apache Beam notebooks, but does not show how to build one. For more information on Apache Beam, see the Apache Beam programming guide (https://beam.apache.org/documentation/programming-guide/).

Apache Beam notebooks currently only support Python. Apache Beam pipeline segments running in these
ooks are run in a test environment, and not against a production Apache Beam runner; however, users can exp
es created in an Apache Beam notebook and launch them on the Dataflow service. For more details, see Laun
ow jobs from your notebook (#launching_dataflow_jobs_from_your_notebook).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboar

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).
4. Enable the Compute Engine API.
   Enable the API (https://console.cloud.google.com/flows/enableapi?apiid=compute.googleapis.com)

Enable additional APIs for pipelines that use other services, such as Pub/Sub, before creating your Apache Be ook instance.

When you finish this guide, you can avoid continued billing by deleting the resources you created. For more details, see Cleaning up (#cleaning_up).

Launching an Apache Beam notebooks instance

1. In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.
2. Navigate to **Dataflow** in the side panel and click **Notebooks**.
3. In the toolbar, click + **New Instance**.
4. Select **Apache Beam**.
5. On the **New notebook instance** page, select a network for the notebook VM and click **Create**.
6. (Optional) If you want to set up a custom notebook instance, click **Customize**. For more information on customizing instance properties, see Create an AI Platform Notebooks instance with specific properties (/ai-platform/notebooks/docs/create-new#create-with-options).

7. Click **Open JupyterLab** when the link becomes active. AI Platform Notebooks creates a new Apache Beam notebook instance.

Installing dependencies (Optional)

Apache Beam notebooks already come with Apache Beam and Google Cloud connector dependencies installed. If your pipeline contains custom connectors or custom PTransforms that depend on third-party libraries, you can install them after you create a notebook instance. For more information, see Installing Dependencies (/ai-platform/notebooks/docs/dependencies) in the AI Platform Notebooks documentation.

Getting started with Apache Beam notebooks

After opening an AI Platform Notebooks instance, example notebooks are available in the **Examples** folder. The following are currently available:

- Word Count
- Streaming Word Count
- Streaming NYC Taxi Ride Data
- Dataflow Word Count

These notebooks include explanatory text and commented code blocks to help you understand Apache Beam concepts and API usage.

Example code from the Streaming Word Count notebook is used in the following sections. There might be som discrepancies between the code snippets in this guide and what is found in the Streaming Word Count notebo

Creating a notebook instance

Navigate to **File > New > Notebook** and select a kernel that is Apache Beam 2.22 or later.

Apache Beam notebooks are built against the master branch of the Apache Beam SDK. This means that the la n of the kernel shown in the notebooks UI might be ahead of the most recently released version of the SDK.

Apache Beam is installed on your notebook instance, so include the `interactive_runner` and `interactive_beam` modules in your notebook.

```
t apache_beam as beam
apache_beam.runners.interactive.interactive_runner import InteractiveRunner
t apache_beam.runners.interactive.interactive_beam as ib
```

If your notebook uses other Google APIs, add the following import statements:

```
apache_beam.options import pipeline_options
apache_beam.options.pipeline_options import GoogleCloudOptions
t google.auth
```

Setting interactivity options

The following sets the data capture duration to 60 seconds.

```
tions.capture_duration = timedelta(seconds=60)
```

For additional interactive options, see the interactive_beam.options class (https://github.com/apache/beam/blob/master/sdks/python/apache_beam/runners/interactive/interactive_beam.py#L48)
.

Creating your pipeline

Initialize the pipeline using an `InteractiveRunner` object.

```
ns = pipeline_options.PipelineOptions()

 the pipeline mode to stream the data from Pub/Sub.
ns.view_as(pipeline_options.StandardOptions).streaming = True

eam.Pipeline(InteractiveRunner(), options=options)
```

Reading and visualizing the data

The following example shows a Apache Beam pipeline that creates a subscription to the given Pub/Sub topic and reads from the subscription.

```
 = p | "read" >> beam.io.ReadFromPubSub(topic=topic)
```

The pipeline counts the words by windows from the source. It creates fixed windowing with each window being 10 seconds in duration.

```
wed_words = (words
"window" >> beam.WindowInto(beam.window.FixedWindows(10)))
```

After the data is windowed, the words are counted by window.

```
wed_words_counts = (windowed_words
"count" >> beam.combiners.Count.PerElement())
```

The `show()` method visualizes the resulting PCollection in the notebook.

```
ow(windowed_word_counts, include_window_info=True)
```

```
ib.show(windowed_word_counts, include_window_info=True)
```

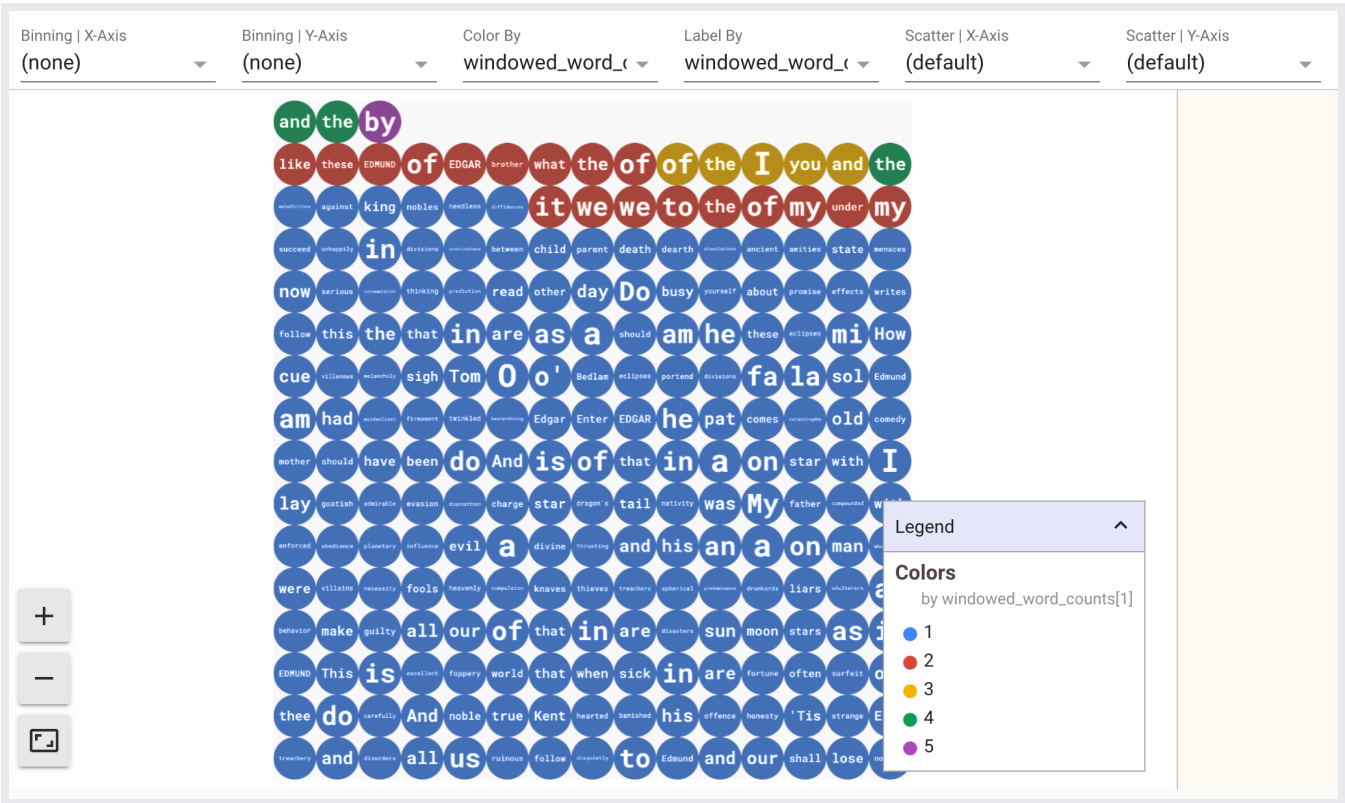
Interactive Beam has detected unbounded sources in your pipeline. In order to have a deterministic replay, a segment of data will be recorded from all sources for 60.0 seconds or until a total of 1.0GB have been written to disk.

Show 10 entries Search:

	windowed_word_counts[0]	windowed_word_counts[1]	event_time	windows	pane_info
0	b'treachery'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
1	b'and'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
2	b'disorders'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
3	b'all'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
4	b'us'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
5	b'ruinous'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
6	b'follow'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
7	b'disquietly'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
8	b'to'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
9	b'Edmund'	1	2020-03-11 06:49:59.999999+0000	2020-03-11 06:49:50.000000+0000 (10s)	Pane 0

Showing 1 to 10 of 228 entries Previous 1 2 3 4 5 ... 23 Next

To display visualizations of your data, pass `visualize_data=True` into the `show()` method. You can apply multiple filters to your visualizations. The following visualization allows you to filter by label and axis:



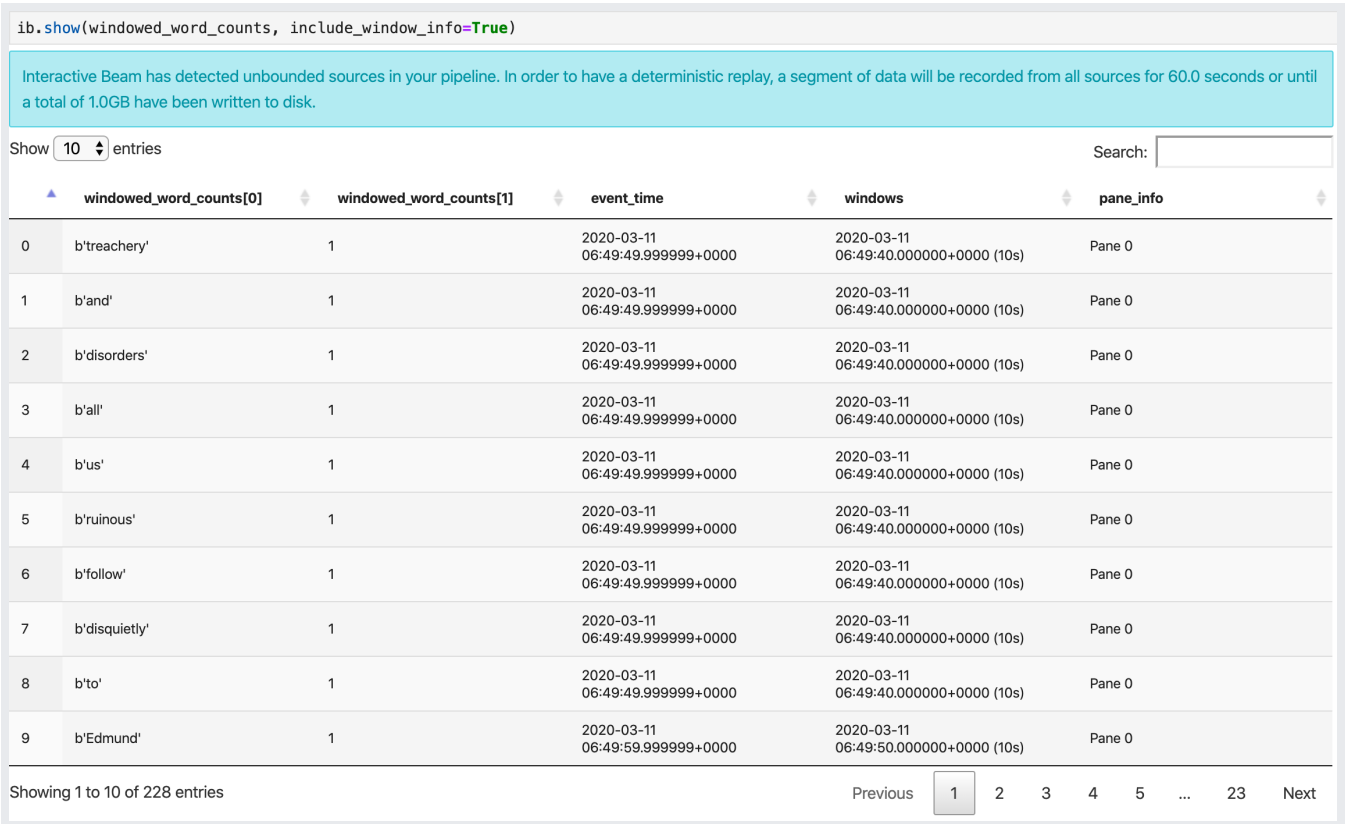
To ensure replayability while prototyping streaming pipelines, the **`show()`** method calls reuse the captured dat t. By setting **`interactive_beam.options.enable_capture_replay = False`**, you can change this beha ave the **`show()`** method always fetch new data. Also, if you add a second unbounded source to your notebook om the previous unbounded source is discarded.

Another useful visualization in Apache Beam notebooks is a Pandas DataFrame (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html). The following example first converts the words to lowercase and then computes the frequency of each word.

```
wed_lower_word_counts = (windowed_words
beam.Map(lambda word: word.lower())
"count" >> beam.combiners.Count.PerElement())
```

The **`collect()`** method provides the output in a Pandas DataFrame.

```
llect(windowed_lower_word_counts, include_window_info=True)
```



```
ib.show(windowed_word_counts, include_window_info=True)
```

Interactive Beam has detected unbounded sources in your pipeline. In order to have a deterministic replay, a segment of data will be recorded from all sources for 60.0 seconds or until a total of 1.0GB have been written to disk.

Show 10 entries Search:

	windowed_word_counts[0]	windowed_word_counts[1]	event_time	windows	pane_info
0	b'treachery'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
1	b'and'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
2	b'disorders'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
3	b'all'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
4	b'us'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
5	b'ruinous'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
6	b'follow'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
7	b'disquietly'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
8	b'to'	1	2020-03-11 06:49:49.999999+0000	2020-03-11 06:49:40.000000+0000 (10s)	Pane 0
9	b'Edmund'	1	2020-03-11 06:49:59.999999+0000	2020-03-11 06:49:50.000000+0000 (10s)	Pane 0

Showing 1 to 10 of 228 entries Previous 1 2 3 4 5 ... 23 Next

Editing and re-executing a cell is a common practice in notebook development. When you edit and re-execute a
che Beam notebook, the cell does not undo the intended action of the code in the original cell. For example, if a
PTransform to a pipeline, re-executing that cell would add an additional PTransform to the pipeline. If you wa
he state, restart the kernel and rerun the cells.

Launching Dataflow jobs from a pipeline created in your notebook

1. (Optional) Before using your notebook to run Dataflow jobs, restart the kernel, rerun all cells, and verify the output. If you skip this step, hidden states in the notebook might affect the job graph in the pipeline object.
2. Enable the Dataflow API (https://console.cloud.google.com/apis/library/dataflow.googleapis.com).
3. Add the following import statement:

```
from apache_beam.runners import DataflowRunner
```

4. Pass in your pipeline options (/dataflow/docs/guides/specifying-exec-params).

```
# Set up Apache Beam pipeline options.
options = pipeline_options.PipelineOptions()

# Set the project to the default project in your current Google Cloud
# environment.
_, options.view_as(GoogleCloudOptions).project = google.auth.default()

# Set the Google Cloud region to run Dataflow.
options.view_as(GoogleCloudOptions).region = 'us-central1'

# Choose a Cloud Storage location.
dataflow_gcs_location = 'gs://<change me>/dataflow'

# Set the staging location. This location is used to stage the
# Dataflow pipeline and SDK binary.
options.view_as(GoogleCloudOptions).staging_location = '%s/staging' % dataflow_

# Set the temporary location. This location is used to store temporary files
# or intermediate results before outputting to the sink.
options.view_as(GoogleCloudOptions).temp_location = '%s/temp' % dataflow_gcs_lo

# Set the SDK location. This is used by Dataflow to locate the
# SDK needed to run the pipeline.
options.view_as(pipeline_options.SetupOptions).sdk_location = (
    '/root/apache-beam-custom/packages/beam/sdks/python/dist/apache-beam-%s0.ta
    beam.version.__version__)
```

You can adjust the parameter values. For example, you can change the `region` value from `us-central1`.

5. Run the pipeline with `DataflowRunner`. This runs your job on the Dataflow service.

```
runner = DataflowRunner()
runner.run_pipeline(p, options=options)
```

`p` is a pipeline object from Creating your pipeline (#creating_your_pipeline).

For an example on how to perform this conversion on an interactive notebook, see the Dataflow Word Count notebook in your notebook instance.

Alternatively, you can export your notebook as an executable script, modify the generated `.py` file using the previous steps, and then deploy your pipeline (/dataflow/docs/guides/deploying-a-pipeline) to the Dataflow service.

Saving your notebook

Notebooks you create are saved locally in your running notebook instance. If you reset (/compute/docs/instances/stop-start-instance#resetting_an_instance) or shut down the notebook instance during development, those new notebooks are deleted. To keep your notebooks for future use, download them locally to your workstation, save them to GitHub (/ai-platform/notebooks/docs/save-to-github), or export them to a different file format.

Cleaning up

After you've finished using your Apache Beam notebook instance, clean up the resources you created on Google Cloud by shutting down the notebook instance (/ai-platform/notebooks/docs/shut-down).

Last updated 2020-08-18 UTC.