

Multi-language Pipelines Tips

This page includes tips and troubleshooting information regarding Apache Beam's multi-language pipelines framework. For full documentation on multi-language pipelines, please see [here](#).

- ["java: command not found" when starting the pipeline](#)
- [grpc error "failed to connect to all addresses" when submitting the job](#)
- ["KeyError: 'beam:coders:javask:0.1'" when expanding the transform using expansion service](#)
- ["java.lang.IllegalArgumentException: Unknown Coder URN beam:coder:pickled_python:v1" when runner a Python pipeline](#)
- ["Unknown Coder URN beam:coder:pickled_python:v1" when running a Java pipeline that uses Python cross-language transforms](#)
- [How to set Java log level from a Python pipeline that uses Java transforms](#)
- [Debugging a Python Test that calls a Java transform](#)
 - [Set up the input arguments to the Python test.](#)
 - [Run the Java expansion service](#)
 - [Run the Python test](#)
 - [Notes](#)
- [Running a cross-language transform that uses a different version of an external transform](#)

"java: command not found" when starting the pipeline

This usually occurs due to `java` command not being available when submitting a multi-language pipeline that uses a Java transform to the Beam runner. Multi-language wrappers implemented in the pipeline SDK may try to automatically start up a Java expansion service, hence `java` command being available in the system is a pre-requisite. This can be resolved by installing JDK in the machine where the job is submitted from and adding the JDK directory with the `java` binary to the environment variable `PATH`.

grpc error "failed to connect to all addresses" when submitting the job

This usually occurs when an expansion service that is used by the pipeline is not available. For example, it could be that you simply forgot to start the expansion service before running the job. Or it could be that a Java expansion service that is automatically started up by a wrapper implemented in the pipeline SDK failed for some reason.

"KeyError: 'beam:coders:javask:0.1'" when expanding the transform using expansion service

This occurs due to Java expansion service returning an expanded transform that uses a Java specific coder as one of its outputs. Cross-language transforms require coders used at the SDK boundary to be [Beam Standard Coders](#) that can be interpreted by all SDKs. Note that internal sub-transforms of the expanded transforms may choose Java specific coders. What matters are final outputs produced by the expanded Java transform.

The solution will be to update the user's transform to produce output *PCollection* types that use standard coders at the SDK boundaries.

"java.lang.IllegalArgumentException: Unknown Coder URN beam:coder:pickled_python:v1" when runner a Python pipeline

This usually means that an expansion request that is sent from Python SDK to a Java expansion service contained Python specific *PickleCoder* that Java SDK cannot interpret. For example, this could be due to following.

- Input *PCollection* that is fed into the cross-language transform in Python side uses a Python specific type. In this case, the *PTransform* that produced the input *PCollection* has to be updated to produce outputs that use [Beam's Standard Coders](#) to make sure Java side can interpret such coders.
- Input *PCollection* uses standard coders but Python type inferencing results in picking up the *PickleCoder*. This can usually be resolved by annotating the predecessor Python transform with the correct type annotation using the `with_output_types` tag. See [this Jira](#) for an example.

"Unknown Coder URN beam:coder:pickled_python:v1" when running a Java pipeline that uses Python cross-language transforms

This usually means that Python SDK was not able to properly determine a portable output *PCollection* type when expanding the cross-language transform. So it ended up picking the default *PickleCoder*.

But Java SDK is unable to interpret this, so it will fail when trying to parse the expansion response from the Python SDK.

The solution is to provide a hint to the Python SDK regarding the element type(s) of the output *PCollection*(s) of the cross-language transform. This can be provided using the `withOutputCoder` or `withOutputCoders` methods of the [PythonExternalTransform](#) API.

How to set Java log level from a Python pipeline that uses Java transforms

For supported runners (e.g. portable runners and Dataflow runner), you can set the log level of Java transforms in the same way of setting python module log level overrides, specifically, using the `--sdk_harness_log_level_overrides` pipeline option. The `python_underline_style` option names will be automatically translated to Java smallCamel style and recognized by the Java SDK harness.

If the runner does not support the automatic mapping of options, One can try adding the corresponding pipeline option as a local pipeline option explicitly in Python side. For example, to suppress all logs from Java *org.apache.kafka* package you can do following.

1. Add a Python PipelineOption that represents the corresponding Java PipelineOption available [here](#). This can be simply added to your Python program that starts up the Beam job.

```
class JavaLoggingOptions(PipelineOptions):
    @classmethod
    def _add_argparse_args(cls, parser):
        parser.add_argument(
            '--sdkHarnessLogLevelOverrides',
            default={},
            type=json.loads,
            help=(
                'Java log level overrides'))
```

2. Specify the additional PipelineOption as a parameter when running the Beam pipeline.

```
--sdkHarnessLogLevelOverrides '{"org.apache.kafka":"ERROR"}'
```

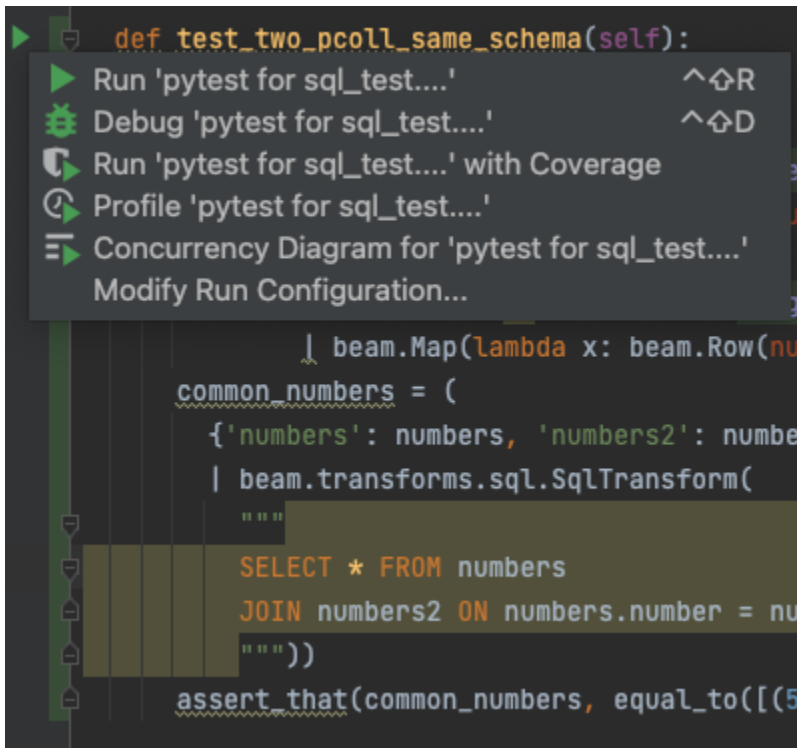
Debugging a Python Test that calls a Java transform

The public [documentation](#) is great on creating and using cross-language transforms. For developers, however, running tests and debugging them is a bit different.

We will showcase how to debug the following test `sql_test.SqlTransformTest.test_two_pcoll_same_schema`.

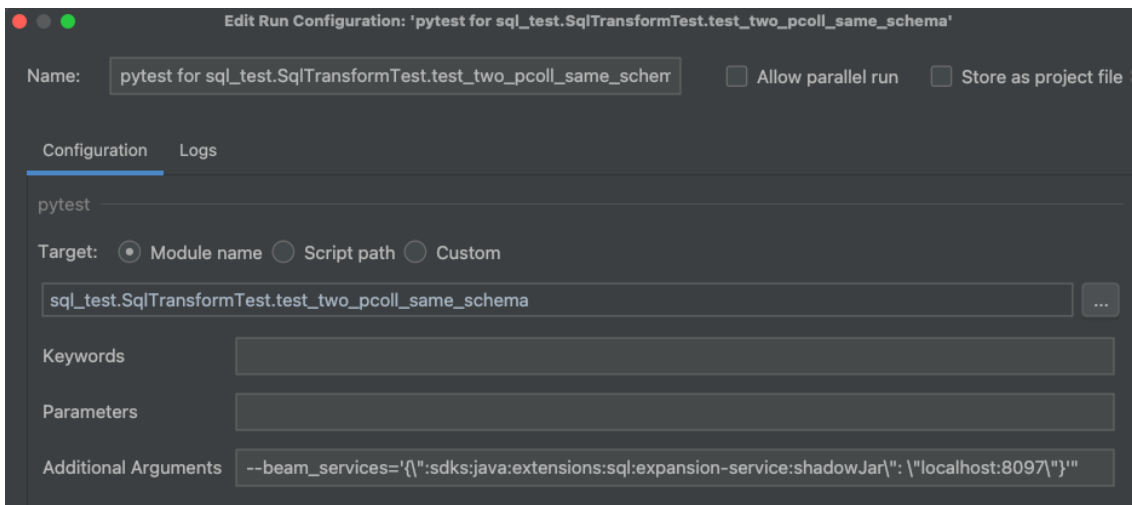
Set up the input arguments to the Python test.

Beside the test function, right click the play button, and click "Modify Run Configuration..."



In `Additional Arguments` , add the arguments. For this example, we add


















```
--test-pipeline-options="--runner=FlinkRunner --beam_services='{\"sdks:java:extensions:sql:expansion-service:shadowJar\": \"localhost:8097\"}'"
```






Run the Java expansion service

The Gradle build [file](#) defines how to run the expansion service. To do this manually, go to the Gradle tab in IntelliJ, go to this path `beam/sdks/java/extensions/sql/expansion-service/tasks/other/` , and find the `runExpansionService` entry.

- To run the expansion service without debugging, right click `runExpansionService` , and simply click "run".
- If you want to debug the Java code, set breakpoints in the IDE in Java files that are part of the expansion service. (i.e. `ExpansionService.Java`). Then click "Debug"

  help>   idev   other audit checkstyleMain checkstyleTest cleanEclipseClasspath cleanEclipseJdt cleanEclipseProject cleanUp compileJava compileTestJava components dependencyReport dependentComponents eclipseClasspath eclipseJdt eclipseProject htmlDependencyReport model packageTests processResources processTestResources propertyReport runExpansionService Run 'beam:sdks:java:exten...'   ^  R Debug 'beam:sdks:java:exten...'   ^  D Run 'beam:sdks:java:exten...' with Coverage Run 'beam:sdks:java:exten...' with 'Async Profiler' Run 'beam:sdks:java:exten...' with 'Java Flight Recorder'

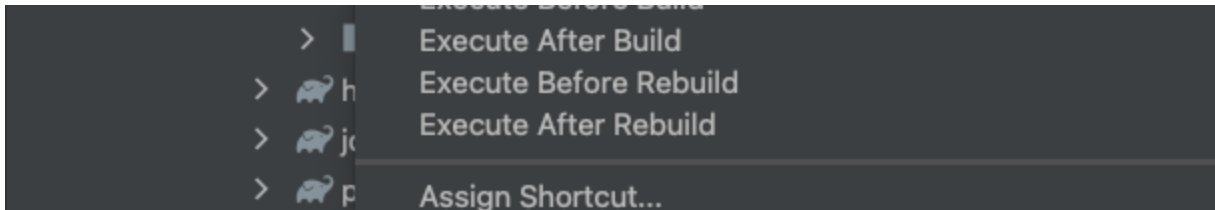
Modify Run Configuration...

 Jump to Source   ⌘ ↓

> Execute Before Sync

> Execute After Sync

> Execute Before Build



Run the Python test

- If you don't want to debug the Python test, right click the button next to the test, and click Run 'pytest for sql_test...'
- If you do want to debug the Python test, first set the breakpoints in your code. Then right click the button next to the test, and click 'Debug 'pytest for sql_test...'

Notes

You can choose to run only Python in debug mode and the expansion service not in debug mode; Java expansion service in debug mode and the Python code not in debug mode; both in debug mode; or neither in debug mode.

Running a cross-language transform that uses a different version of an external transform

By default, cross-language transforms released with Beam will automatically startup an expansion service that includes external transforms. Usually these transforms will be from the same Beam release as the pipeline SDK. For example, when using cross-language Kafka transforms from Python SDK, underlying Java KafkaIO transforms will be from the same released SDK version. If you need to use a different external SDK version you can do the following.

- Startup an expansion service that includes external transforms from a compatible SDK version. See [here](#) for more details.
- Specify the expansion service when defining the cross-language transform in your pipeline. For example, expansion service used by Python ReadFromKafka transform can be overridden [here](#).