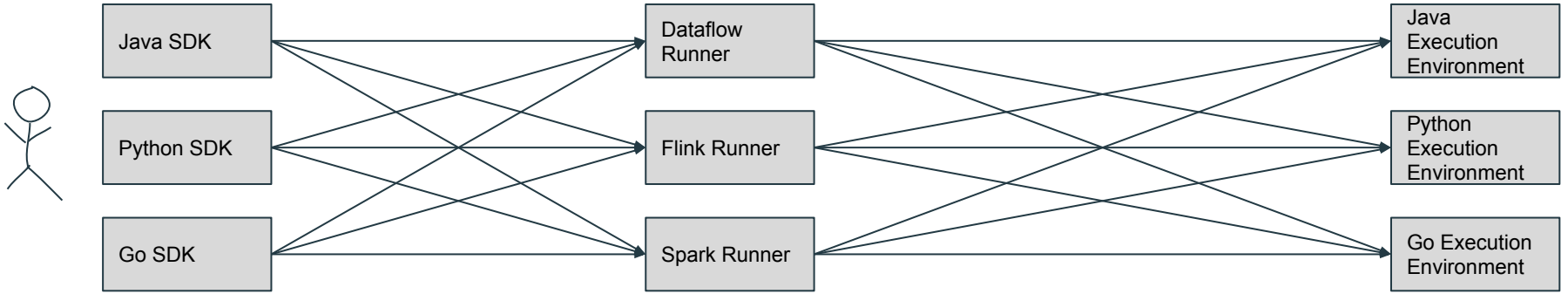
Multi-language Pipelines

Heejong Lee
(heejong@google.com)

Agenda

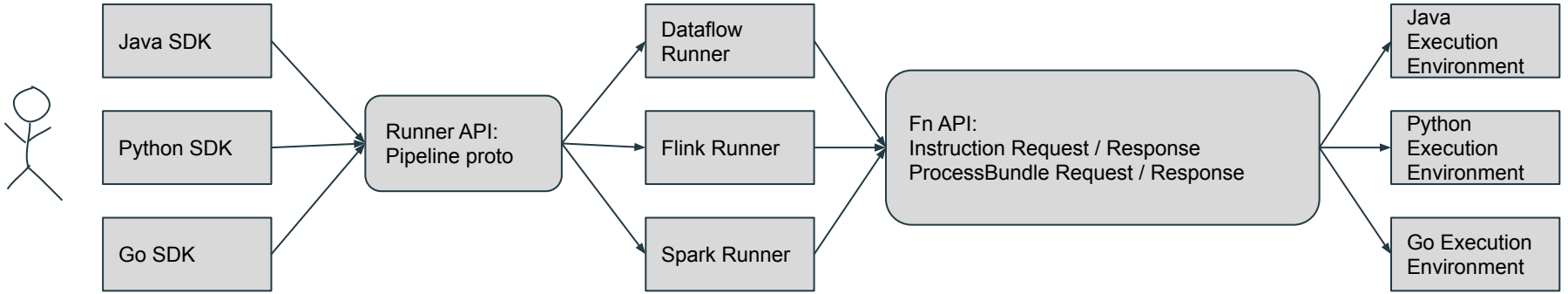
- Apache Beam Portability Model: PTransform, PCollection, Environment
- Portable Pipeline Execution
- How External Transform Works
- API: Python Pipelines using Java External Transforms
- Demo
- Tips and Current Support Status

Without Portability Model



N SDKs X M Runners

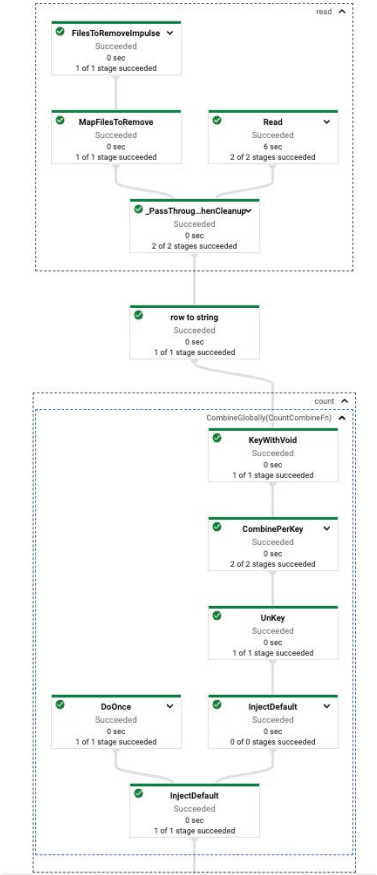
With Portability Model



N SDKs + M Runners

Beam Model: Pipeline Components

- Beam pipeline is a directed acyclic graph which mainly consists of PTransforms and PCollections
- PCollection represents data, PTransform is a transformer that accepts inputs from multiple PCollections and emits outputs to multiple PCollections
- Coder defines how to encode and decode data during transmission between runners, workers and execution environments
- Environment specifies the type and capabilities of the execution environments
- PCollection and standard coders are SDK language-agnostic



Beam Model: Pipeline Components

```
message Components {

  map<string, PTransform> transforms = 1;

  map<string, PCollection> pcollections = 2;

  map<string, WindowingStrategy> windowing_strategies = 3;

  map<string, Coder> coders = 4;

  map<string, Environment> environments = 5;

}
```

Beam Model: PTransform

```
message PTransform {

  string unique_name = 5;

  FunctionSpec spec = 1;

  repeated string subtransforms = 2;

  map<string, string> inputs = 3;

  map<string, string> outputs = 4;

  string environment_id = 7;

}
```

```
message FunctionSpec {

  string urn = 1;

  bytes payload = 3;

}
```

PTransform defines its execution environment

Beam Model: PCollection

```
message PCollection {
  string unique_name = 1;
  string coder_id = 2;
  IsBounded.Enum is_bounded = 3;
  string windowing_strategy_id = 4;
  repeated DisplayData display_data = 5;
}
```

```
message Coder {
  FunctionSpec spec = 1;
  repeated string component_coder_ids = 2;
}
```

PCollection defines its coder.
SDKs can read from and write to PCollection if they know the coder specified.

Beam Model: Environment

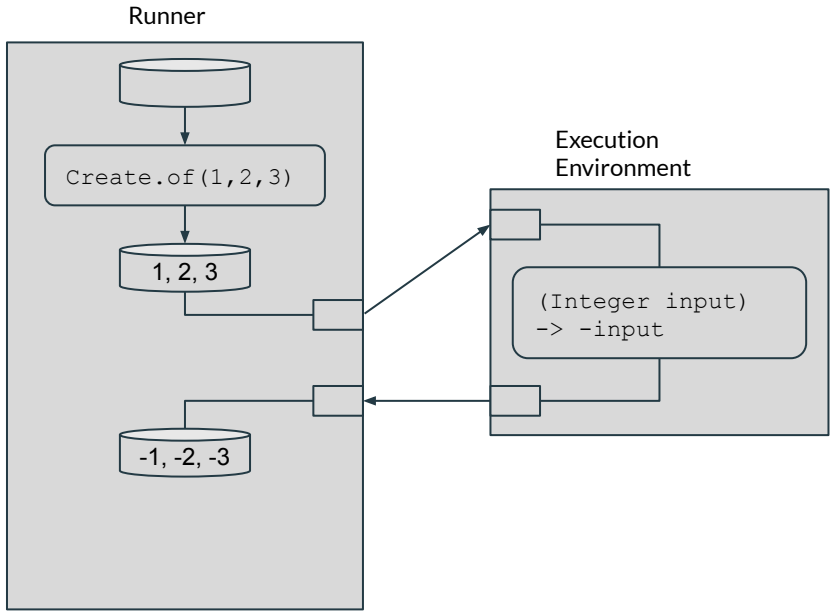
```
message Environment {
  string urn = 2;
  bytes payload = 3;
  repeated DisplayData display_data = 4;
  repeated string capabilities = 5;
  repeated ArtifactInformation dependencies = 6;
  map<string, bytes> resource_hints = 7;
}
```

```
message StandardEnvironments {
  enum Environments {
    DOCKER = 0 [(beam_urn) = "beam:env:docker:v1"];
    PROCESS = 1 [(beam_urn) = "beam:env:process:v1"];
    EXTERNAL = 2 [(beam_urn) = "beam:env:external:v1"];
    DEFAULT = 3 [(beam_urn) = "beam:env:default:v1"];
  }
}
```

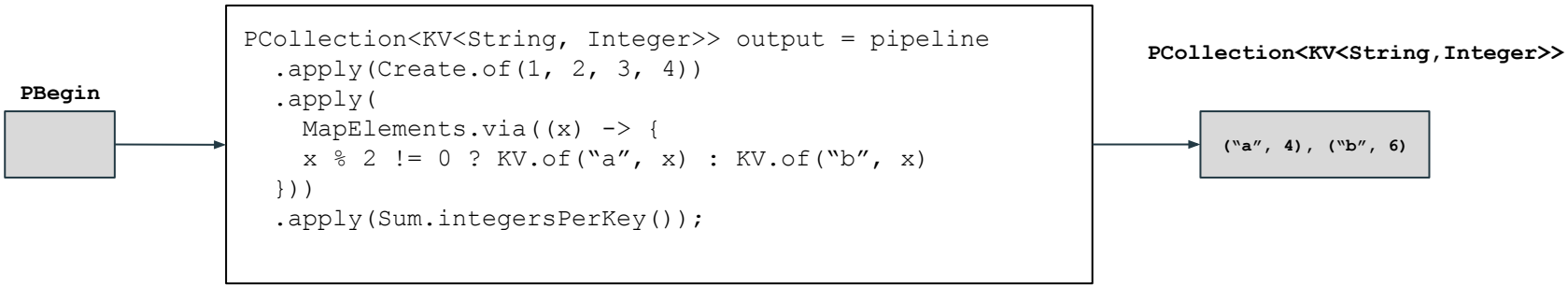
Environment defines the type of execution environment.
DOCKER environment is preferred by default.

Example of Portable Pipeline Execution

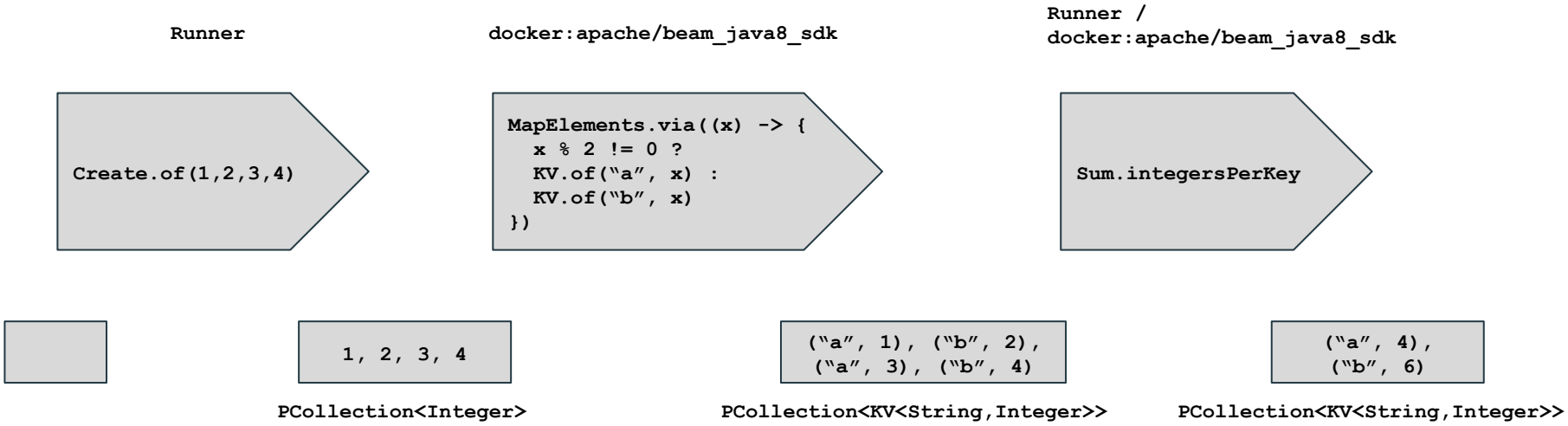
```
PCollection<Integer> output = pipeline
  .apply(Create.of(1, 2, 3))
  .apply(
    MapElements.via(
      new SimpleFunction<Integer, Integer>() {
        @Override
        public Integer apply(Integer input) {
          return -input;
        }
      }));
```



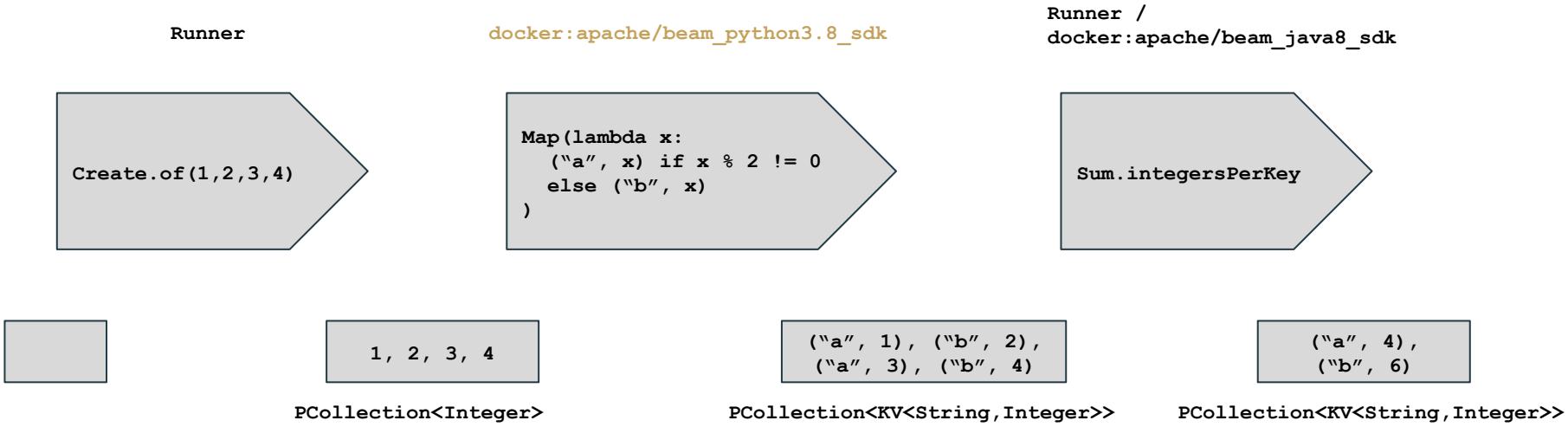
Example of Portable Pipeline Execution



Example of Portable Pipeline Execution



Example of Multi-Language Pipeline Execution



Constructing Python Multi-language Pipelines

- **`ExternalTransform`**, `JavaExternalTransform` ask the expansion service for PTransform protos that can be expanded into currently constructing pipeline

```
with pipeline as p:
      res = (
          p
          | beam.Create(['a', 'b']).with_output_types(str)
          | beam.ExternalTransform(
              TEST_PREFIX_URN,
              ImplicitSchemaPayloadBuilder({'data': u'0'}),
              self.expansion_service))
      assert_that(res, equal_to(['0a', '0b']))
```

Constructing Python Multi-language Pipelines

- `ExternalTransform`, **`JavaExternalTransform`** ask the expansion service for PTransform protos that can be expanded into currently constructing pipeline

```
with pipeline as p:
      res = (
          p
          | beam.Create(['a', 'b']).with_output_types(str)
          | beam.JavaExternalTransform(
              "org.apache.beam.sdk.io.TextIO").write().to("gs://test/test.txt"))
```

Java API

```
PCollection<String> lines = ...;
 lines.apply(TextIO.write().to("/path/to/file.txt")
      .withSuffix(".txt")
      .withCompression(Compression.GZIP));
```

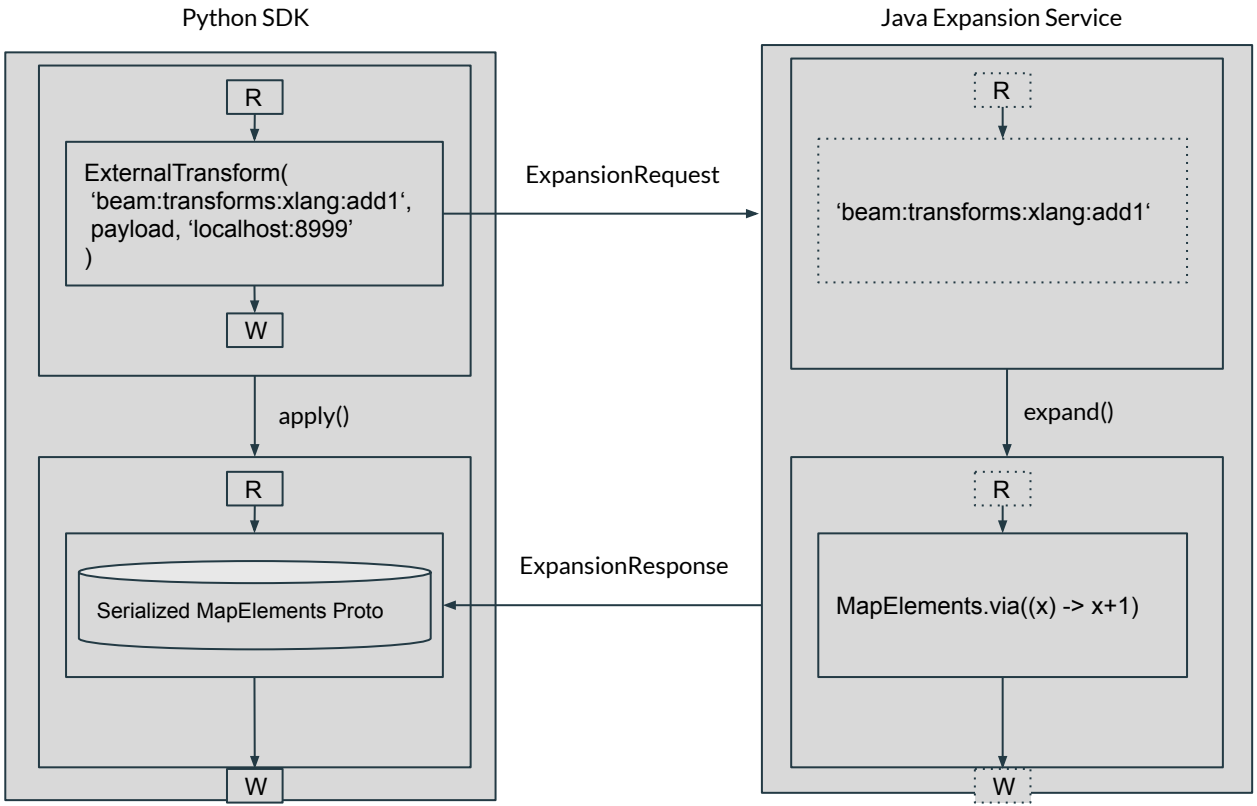
Registering External Java PTransform

- The expansion service provides the expandable PTransform proto

```
@AutoService(ExternalTransformRegistrar.class)
public static class TestTransformRegistrar implements ExternalTransformRegistrar {
  @Override
  public Map<String, Class<? extends ExternalTransformBuilder<?, ?, ?>>> knownBuilders() {
      return ImmutableMap.of(TEST_PREFIX_URN, PrefixBuilder.class);
  }
  public static class PrefixBuilder
          implements ExternalTransformBuilder<
              StringConfiguration, PCollection<? extends String>, PCollection<String>> {
    @Override
    public PTransform<PCollection<? extends String>, PCollection<String>> buildExternal(
        StringConfiguration configuration) {
      return MapElements.into(TypeDescriptors.strings())
        .via((String x) -> configuration.data + x);
    }
  }
}
```

How External Transform Expansion Works

- A native SDK sends the expansion request to the expansion service
- The expansion service looks up the requested PTransform and locally expands it in the context received along with the expansion request
- The expansion service converts the expanded PTransform object back to the PTransform proto and sends it back to the native SDK
- The native SDK temporarily stores the expanded external PTransform proto in a wrapper object for external transforms
- When the native SDK translates its pipeline object to pipeline proto, the previously stored external PTransform proto is plugged into the other protos converted from the native SDK


Python SDK
Java Expansion Service
R
ExternalTransform(
'beam:transforms:xlang:add1',
payload, 'localhost:8999'
)
W
ExpansionRequest
'beam:transforms:xlang:add1'
apply()
expand()
R
Serialized MapElements Proto
ExpansionResponse
MapElements.via((x) -> x+1)
W

Expansion Request

```
message ExpansionRequest {

  org.apache.beam.model.pipeline.v1.Components components = 1;

  org.apache.beam.model.pipeline.v1.PTransform transform = 2;

  string namespace = 3;

  map<string, string> output_coder_requests = 4;

}
```

Beam Model: PTransform (Revisited)

```
message PTransform {

  string unique_name = 5;

  FunctionSpec spec = 1;

  repeated string subtransforms = 2;

  map<string, string> inputs = 3;

  map<string, string> outputs = 4;

  string environment_id = 7;

}
```

```
message FunctionSpec {

  string urn = 1;

  bytes payload = 3;

}
```

Expansion Request Payload

```
message ExternalConfigurationPayload {

  // A schema for use in beam:coder:row:v1

  Schema schema = 1;

  // A payload which can be decoded using beam:coder:row:v1 and the given

  // schema.

  bytes payload = 2;

}
```

Example Expansion Request

```
components {
  environments {
    key: "beam:env:docker:v1"
    value {
      urn: "beam:env:docker:v1", payload: "\n apache/beam_java8_sdk:2.40.0.dev"
      capabilities: "beam:version:sdk_base:apache/beam_java8_sdk:2.40.0.dev"
    }
  }
}
transform {
  spec {
    urn: "beam:transforms:python:fully_qualified_named"
    payload:
"\n\233\001\n\021\n\013constructor\032\002\020\007\n`\n\006kwargs\032R2P\nN\n\022\n\014file_pattern\032\002\020\007\n\0
22\n\010validate\032\002\020\010 \001(\001\022$01150aee-886c-44a4-a3ba-1f1e4921fec5
\001(\001\022$2d2a9289-c5df-421a-b670-f6b1a0a611a4\022S\002\000\033apache_beam.io.ReadFromText\002\0001gs://apache-beam
-samples/shakespeare/kinglear.txt\000"
  }
  unique_name: "external0"
}
namespace: "External_0"
```

Expansion Response

```
message ExpansionResponse {
  org.apache.beam.model.pipeline.v1.Components components = 1;
  org.apache.beam.model.pipeline.v1.PTransform transform = 2;
  repeated string requirements = 3;
  string error = 10;
}
```

Example Expansion Response

```
transform {
  spec {
    urn: "beam:transforms:python:fully_qualified_named"
    payload:
"\n\307\001\n\021\n\013constructor\032\002\020\007\n2\n\004args\032*2(\n&\022$da55433c-47b9-48fc-871c-3aec4c449565\nX\n
\006kwargs\032N2L\nJ\n\022\n\014file_pattern\032\002\020\007\n\016\n\010validate\032\002\020\010\022$01cd72fb-e408-4b28
-b1ce-82f7b0e0b66e\022$51a7dee1-8c37-4624-b0dc-d135411bd7eb\022U\003\000\033apache_beam.io.ReadFromText\000\000\002\000
1gs://apache-beam-samples/shakespeare/kinglear.txt\000"
  }
  subtransforms: "External_0_AppliedPTransform_external0-ReadFromText_3"
  outputs {
    key: "None"
    value: "External_0_PCollection_PCollection_3"
  }
  unique_name: "external0"
  environment_id: "External_0_Environment_default_environment_1"
}
```

Full Proto: https://gist.github.com/ihji/c4962d98cb1662f541f2ab23cba00a43

Dependencies for Java External Transforms

- By default, the Java expansion service creates the list of dependencies from jars on its own classpath
- Bundling all dependencies into a shadowjar (or onejar) is the easiest way to manage Java external transform dependencies
- Otherwise, users can manually specify the list of dependencies by `--filesToStage` command-line option

Example Environment Proto

```
environments {
    key: "beam:env:docker:v1"
    value {
      urn: "beam:env:docker:v1"
      payload: "\n apache/beam_java8_sdk:2.40.0.dev"
      dependencies {
        type_urn: "beam:artifact:type:file:v1"
        type_payload:
"\nH/Users/heejong/.sdkman/candidates/java/8.0.312-tem/jre/lib/ext/dnsns.jar\022@4df20a72eda6aebb50f90ef1fd82b21319df0c9e6440d47e9311a69
d4ef6cef0"
        role_urn: "beam:artifact:role:staging_to:v1"
        role_payload: "\n5dnsns-TfIKcu2mrrtQ-Q7x_YKyExnfDJ5kQNR-kxGmnU72zvA.jar"
      }
      dependencies {
        type_urn: "beam:artifact:type:file:v1"
        type_payload:
"\nZ/Users/heejong/Works/examples-beam/external_java_using_python/build/libs/portable-beam.jar\022@a8f7698dc5b4d1abee61d3d160ea3da997cef
29dbaf32f1c0669eea7aaa87acf"
        role_urn: "beam:artifact:role:staging_to:v1"
        role_payload: "\n=portable-beam-qPdpjcW00avuYdPRYOo9qZfO8p268y8cBmnup6qoes8.jar"
      }
    }
  }
```

Coder and Data Interoperability

- Elements need to be encoded and decoded by standard (universal) coders when they want to cross the SDK language border; `SerializableCoder` only works for Java SDK, `PickleCoder` only works for Python SDK
- Some useful standard coders (Java class names): `ByteArrayCoder`, `BooleanCoder`, `StringUtf8Coder`, `KvCoder`, `VarLongCoder`, `DoubleCoder`, `IterableCoder`, `RowCoder`
- RowCoder is especially useful since users can define any composite types with RowCoder schema

Demo

Java Wordcount using Python TextIO

```
java -cp build/libs/portable-beam.jar
org.example.MultiLanguagePythonRead
--inputFile=gs://apache-beam-samples/shakespe
are/kinglear.txt --numRecords=5525
--runner=PortableRunner
--jobEndpoint=localhost:18088
```

```
class MultiLanguagePythonRead {

  @SuppressWarnings("unused")
  public interface Options extends PipelineOptions {
    @Description("The input file to be read" )
    @Validation.Required
    String getInputFile();

    void setInputFile(String filename);

    @Description("The expected number of records" )
    @Validation.Required
    long getNumRecords();

    void setNumRecords(long numRecords);
  }

  public static void main(String[] args)  throws Exception {
    Options options =
PipelineOptionsFactory. fromArgs(args).as(Options.class);
    Pipeline p = Pipeline. create(options);
    PCollection<Long> count = p. apply("Read",
              PythonExternalTransform.<PBegin, PCollection<String>>from(
                        "apache_beam.io.ReadFromText" ,
"localhost:18089" )
                       . withKwarg("file pattern", options.getInputFile())
                       . withKwarg("validate", false))
             . apply("Count", Count.globally());
    PAssert. thatSingleton(count). isEqualTo(options. getNumRecords());

    p.run().waitUntilFinish();
  }
}
```

Demo

Python Wordcount using Java TextIO

```
python wordcount_xlang.py
--input=gs://apache-beam-samples/shakespeare/ki
nglear.txt --num_records=5525
--runner=PortableRunner
--job_endpoint=localhost:18088
```

```
def main():
  logging.getLogger().setLevel(logging.INFO)

  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--input',
      dest='input',
      default='gs://dataflow-samples/shakespeare/kinglear.txt' ,
      help='Input file to process.' )
  parser.add_argument(
      '--num_records' ,
      dest='num_records' ,
      required=True,
      type=int,
      help='Number of records' )

  known_args, pipeline_args = parser.parse_known_args()

  pipeline_options = PipelineOptions(pipeline_args)

  pipeline_options.view_as(SetupOptions).save_main_session =   True

  java_read =
JavaExternalTransform( 'org.apache.beam.sdk.io.TextIO' ).read()
  with beam.Pipeline(options=pipeline_options)  as p:
    count = (
      p | 'read' >> getattr(java_read,
'from')(filepattern=known_args.input)
        | 'count' >> combine.Count.Globally()
    )
    assert_that(count, equal_to([known_args.num_records]))
```

Example Codes

- How to write and register Java external transforms
  - TestExpansionService: https://github.com/apache/beam/blob/v2.38.0/sdks/java/testing/expansion-service/src/test/java/org/apache/beam/sdk/testing/expansion/TestExpansionService.java
- How to use Java external transform from Python SDK
  - external_test: https://github.com/apache/beam/blob/v2.38.0/sdks/python/apache_beam/transforms/external_test.py
  - multi-language/python: https://github.com/apache/beam/tree/v2.38.0/examples/multi-language/python
  - wordcount_xlang: https://github.com/apache/beam/blob/v2.38.0/sdks/python/apache_beam/examples/wordcount_xlang.py

Benefits of Multi-language Pipelines

- Save time and resources by implementing PTransform once and utilizing in all SDKs e.g. no need to add Kafka IO module to all Python, Java and Go SDKs, users can pick the target SDK which has the best third-party library support
- Pick the best existing PTransform implementation from any SDKs e.g. one can choose to use the most stable one and/or the fastest one
- Simplify development environment and no need to teach a new programming language to developers for using a specific SDK e.g. don't have to learn and develop in Python for TFX transforms

Current Support Status (May, 2022)

- “Supported” means certain combinations of host and guest SDKs are well tested and utility functions for easier use are implemented; multi-language capability itself was totally enabled by Beam model and portability framework
- Supported Runners: Dataflow Runner (runner_v2), Flink Portable Runner, Spark Portable Runner, Direct Portable Runner (a.k.a. FnAPI runner)
- Supported SDKs
  - Python pipeline using Java external transforms: GA for Dataflow, commonly used for Java SDK IO modules
  - Java pipeline using Python external transform: In development, Preview for Dataflow later this year. Expected to be commonly used for TFX and pandas dataframe operations
  - Go pipeline using Java/Python external transform: In development