
BEAM
SUMMIT

Multi-language Pipelines

A unique Beam feature that will make your team more efficient

Chamikara Jayalath
Google LLC

BEAM SUMMIT
NYC 2023

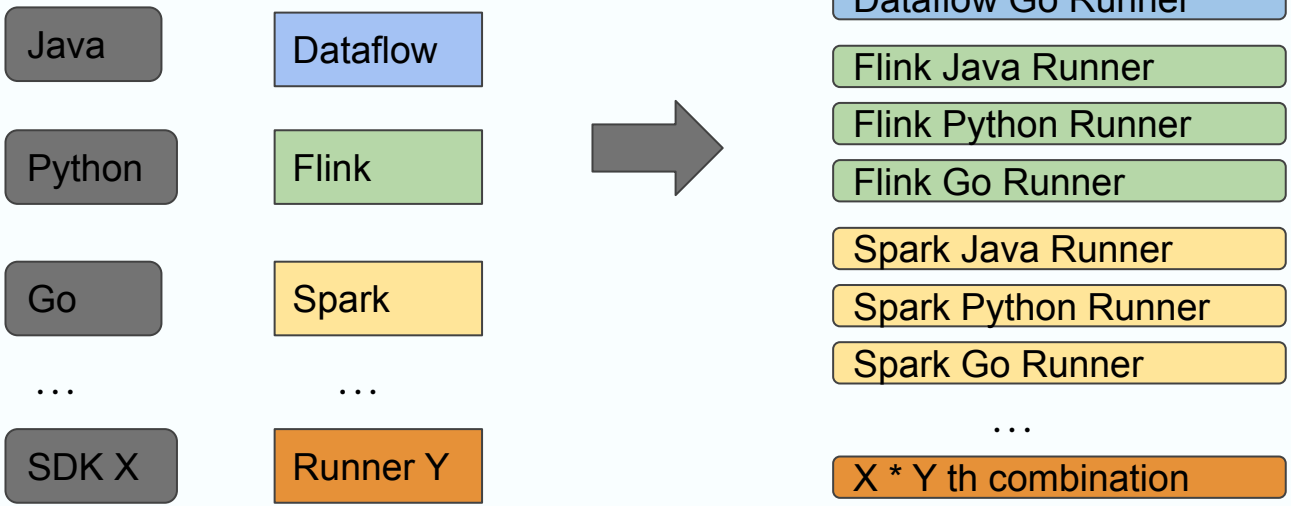
Beam SDKs

- APIs for developing data processing pipelines
- Transforms, for example, sources and sinks.
- Converts the pipeline to a format understood by runners
- Main SDKs - Java, Python, Go
- DSLs - Scio, YAML, DBT

Beam Runners

- Execute Beam pipelines
- Optimize Beam pipelines
- Manage Beam pipelines
- Different execution modes
  - Local - Direct runner
  - Distributed - Flink, Spark, Samza
  - Distributed and managed - Dataflow

SDKs and Runners

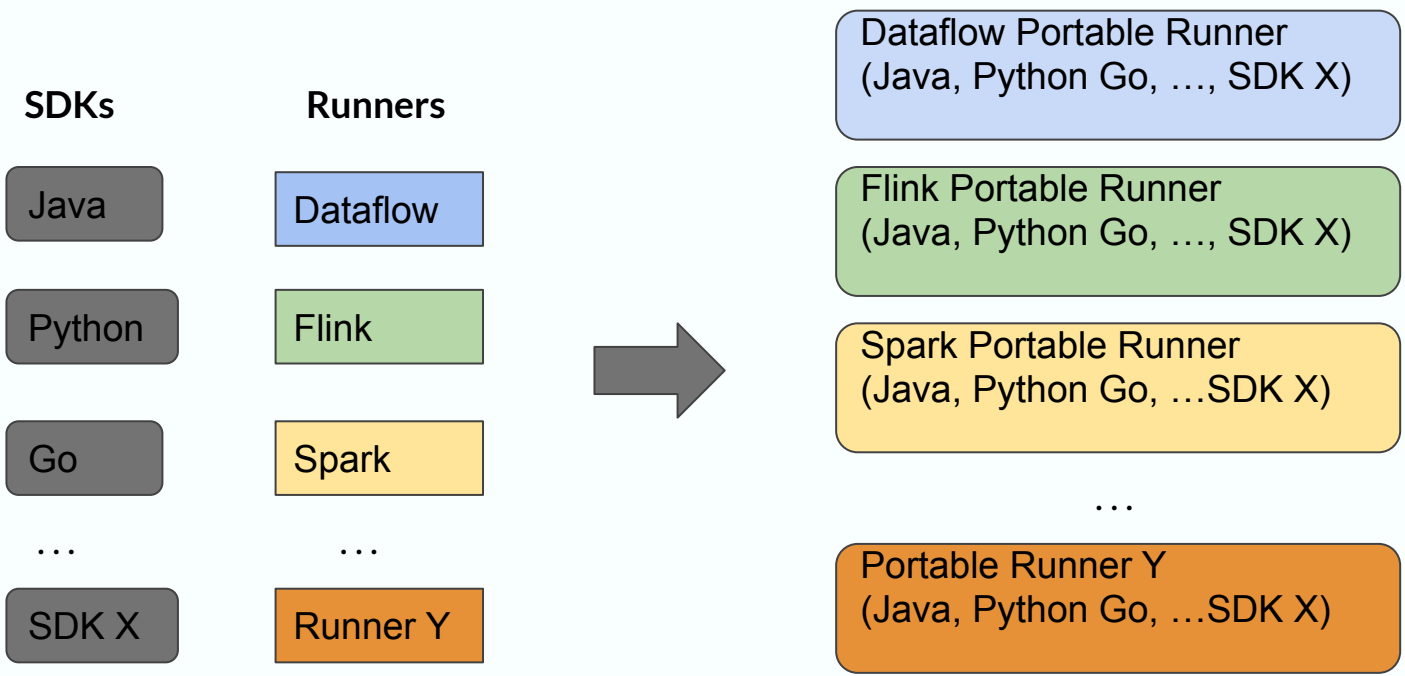


Beam Portability Framework

Quadratic Beam ecosystem

Linear Beam ecosystem

Beam Portability Framework

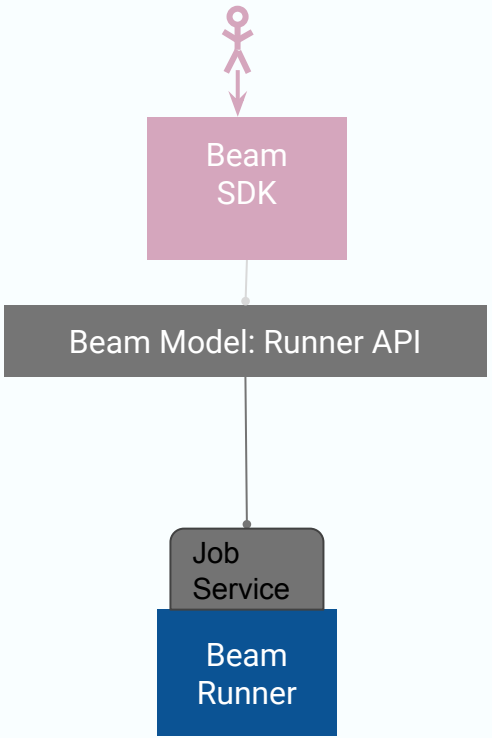


Beam Portability Framework - Two aspects

- Job submission
- Job execution

Beam Portability framework - Job Submission

- Pipeline defined using an SDK
- Pipeline is converted to a SDK and runner agnostic definition using the Beam Runner API
- Pipeline is submitted to a runner (for example, Dataflow Runner V2, Beam Portable Spark, Beam portable Flink)

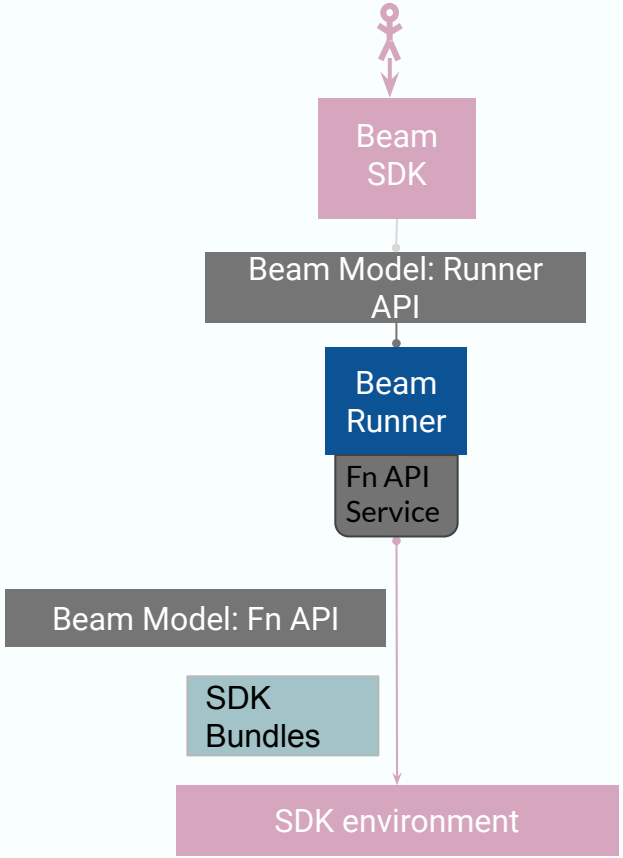


Beam Environments

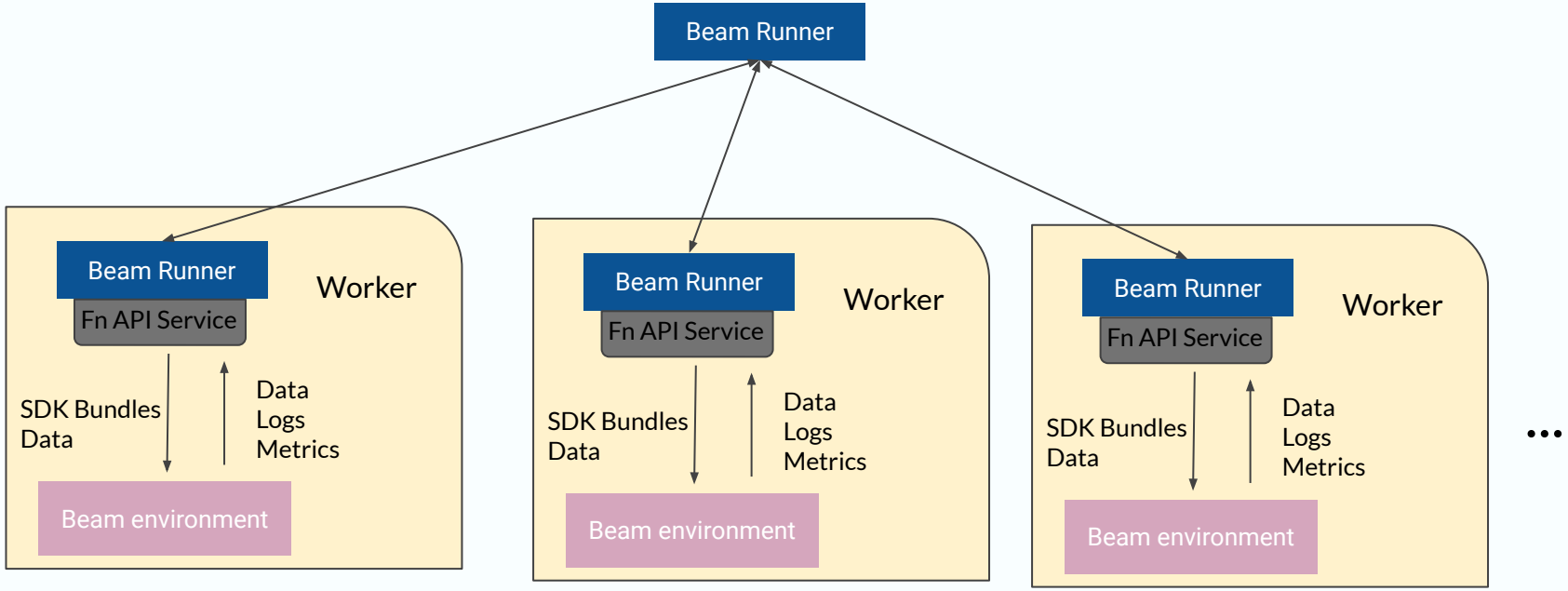
- Environments for executing Beam UDFs (for example, DoFn, CombineFn)
- Chosen by the Beam runners
- Well defined in Beam Runner API proto
- For example,
  - A docker container that contains the SDK
  - A native process that can execute user code

Beam Portability framework - Job Execution

- Runner optimizes the pipelines
- Runner identifies bundles to be executed using an SDK environment
- SDK bundle - some data + some steps
- Runner starts up one or more SDK environments
- Runner executes SDK bundles using the Beam Fn API



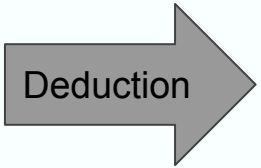
Beam Portability framework - Distributed Execution



Multi-language pipelines

Key Insights

- SDK agnostic pipeline definition may refer to two or more Beam environments
- Fn API based pipeline execution may start more than one type of SDK environment



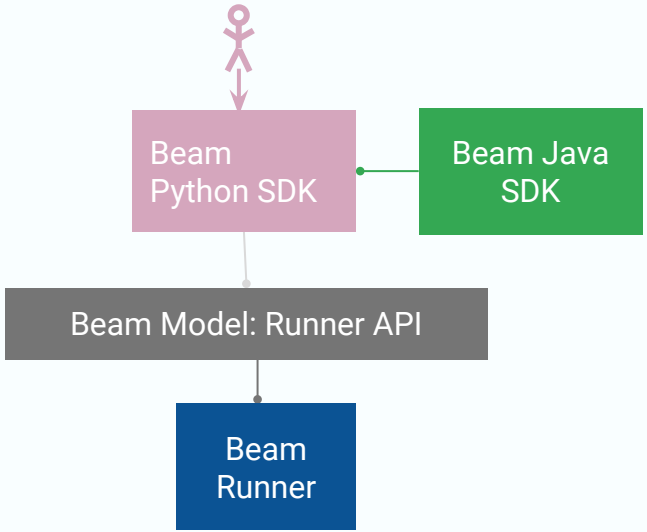
A Beam portable runner can execute pipelines with transforms from multiple language SDKs

Multi-language pipelines

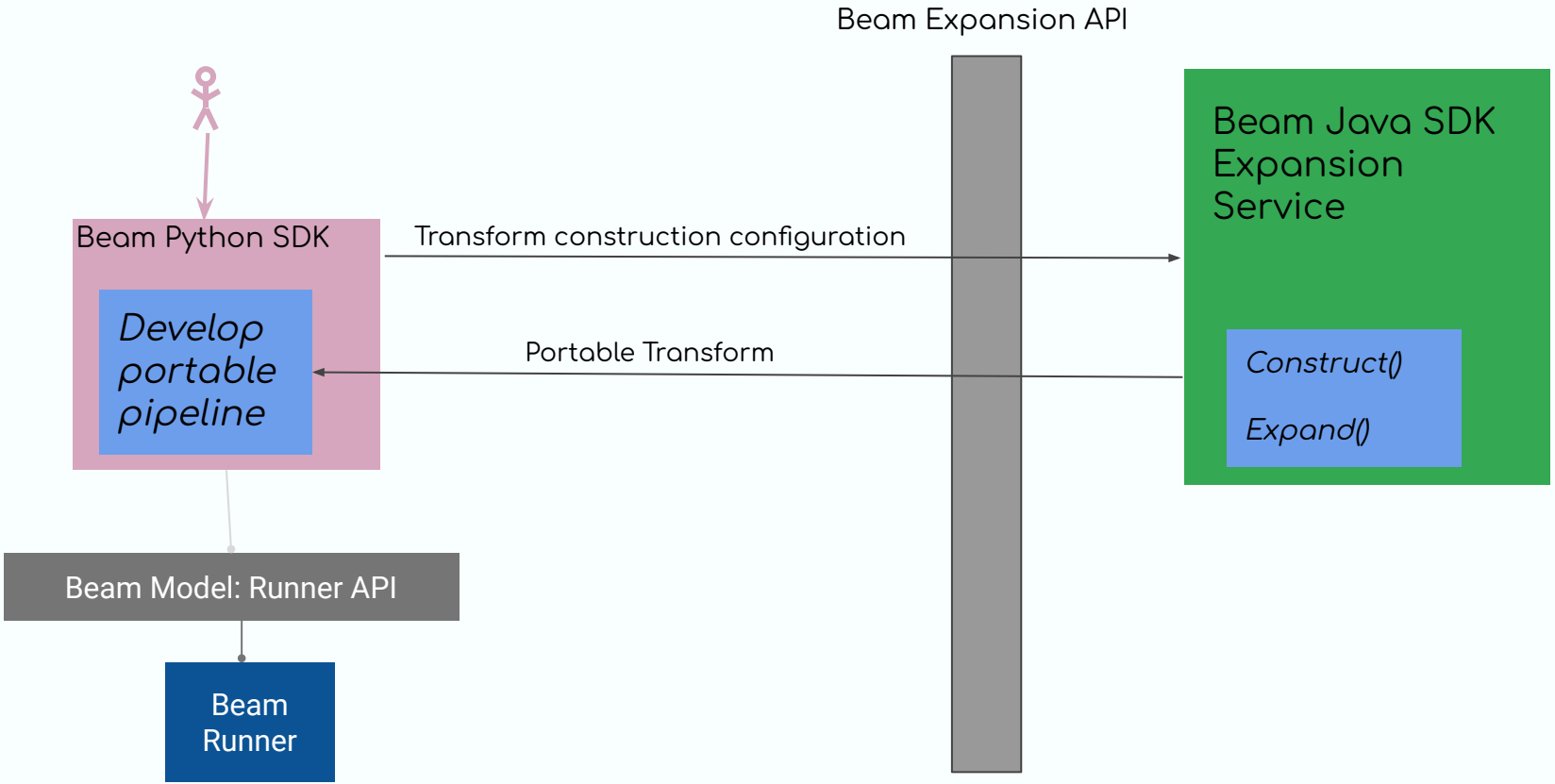
Pipelines that employ more than one Beam SDK.

Multi-language pipelines

- External SDKs define a part of the Runner API definition
- Main SDK constructs the full Runner API definition

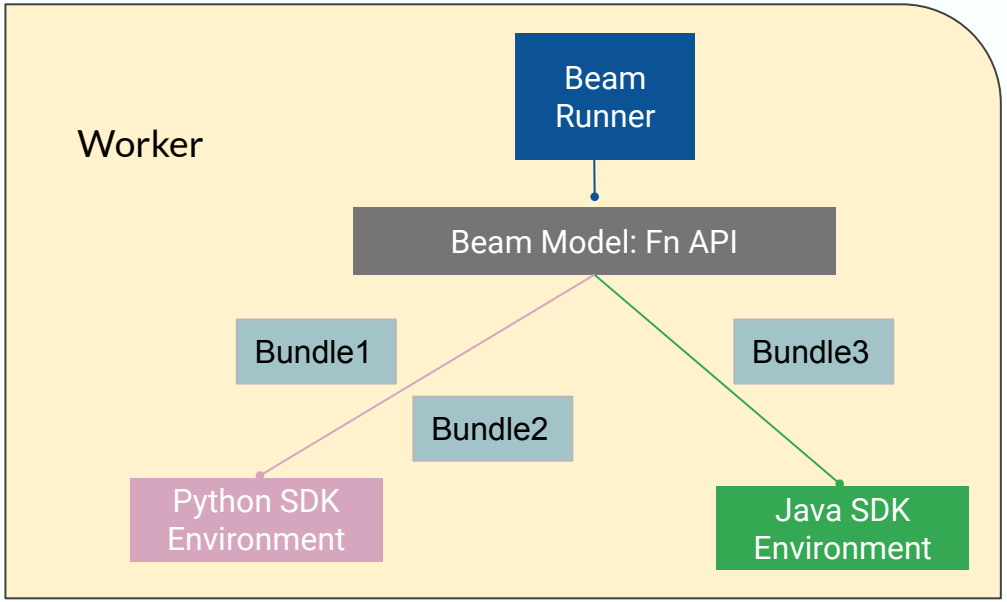


Multi-language pipelines - Expansion Services



Multi-language pipelines - job execution

Runner sends the SDK bundles to corresponding environments.



Multi-language Pipelines - Benefits

- Reduced cost of development
- Reduced maintenance overheads

Reduced Cost of Development

- Develop once and offer to all SDK languages
- Share I/O connectors
- Share code between development teams
- New SDKs can be introduced with reduced effort

Reduced Maintenance Overheads

- No more multiple implementations of complex transforms
- Evolve development teams without re-implementing
- Easily use transforms developed by third parties
- Uniform user experience when using multiple SDKs

Multi-language Pipelines - Benefits

Overall, Apache Beam Multi-language pipelines framework can make your team's software development and maintenance efforts much more efficient and save a significant amount of associated costs.

Example pipelines

- Using the Java *Count.perElement()* transform from a Python pipeline
- Using the Python *RunInference* transform from a Java pipeline

Java from Python - API

- Configuration
- Builder
- Registrar

Java Count.perElement() from Python

Configuration

```
public class JavaCountConfiguration {}
```

Java Count.perElement() from Python

Builder

```
public class JavaCountBuilder
    implements ExternalTransformBuilder<
        JavaCountConfiguration, PCollection<String>, PCollection<KV<String, Long>>> {

  @Override
  public PTransform<PCollection<String>, PCollection<KV<String, Long>>> buildExternal(
      JavaCountConfiguration configuration) {
    return new JavaCount();
  }
}
```

Registrar

```java
@AutoService(ExternalTransformRegistrar.class)
public class JavaCountRegistrar implements ExternalTransformRegistrar {

  static final String URN = "beam:transform:org.apache.beam:javacount:v1";

  @Override
  public Map<String, ExternalTransformBuilder<?, ?, ?>> knownBuilderInstances() {
    return ImmutableMap.of(URN, new JavaCountBuilder());
  }
}
```

Python API

```
beam.ExternalTransform(URN, payload)
```

```
beam.ExternalTransform(URN, payload, expansion_service)
```

Java Count.perElement() from Python

Demo - Java Count.perElement() from Python using DirectRunner.

Using Java from Python - simplified

- Wrappers
- Java class lookup
- Schema-aware transforms (separate talk)

Python from Java - API

- `PythonExternalTransform('<name>')`

- `'<name>'`
  - Fully qualified name of the Python transform
  - A callable that returns the Python transform

- ```
  PythonExternalTransform.from('<name>')
      .withArgs(...)
      .withKwargs(...)
      .withExtraPackages();
  ```
  ```

Python RunInference from Java

```
PythonExternalTransform.<PCollection<?>, PCollection<Row>>from(
  "apache_beam.ml.inference.base.RunInference.from_callable")
  .withExtraPackages(ImmutableList.of("scikit-learn", "pandas"))
  .withOutputCoder((coder))
  .withKwarg(
    "model_handler_provider",
    PythonCallableSource.of(
      "apache_beam.ml.inference.sklearn_inference.SklearnModelHandlerNumpy"))
  .withKwarg("model_uri", modelFile));
```

Python RunInference from Java

Demo - Python RunInference from Java using DirectRunner

Python RunInference from Java - simplified

- For key transforms, instead of using the base API, you could use a Beam provided wrapper.
- Usually more concise and convenient than that base API

```
RunInference.of(
   "apache_beam.ml.inference.sklearn_inference.SklearnModelHandlerNumpy",
   schema)
 .withKwarg(
   "model_uri", modelFile)
```

Python RunInference from Java

Demo - SklearnMnistClassification using DataflowRunner.

Go SDK Support

- Supports using Java transforms from Go. For example,
  - Java BigQuery Storage Write API from Go
  - Java KafkaIO from Go
- Supports using Python transforms from Go. For example,
  - Python RunInference from Go
- Future
  - Go Prism runner - will be able to support multi-language locally for all SDKs (separate talk)
  - Implement a Go expansion service to make Go transforms available to other SDKs

References

- Python multi-language quickstart
  - beam.apache.org/documentation/sdks/python-multi-language-pipelines/
- Java multi-language quickstart
  - beam.apache.org/documentation/sdks/java-multi-language-pipelines/
- Multi-language programming guide
  - beam.apache.org/documentation/programming-guide/#multi-language-pipelines
- Multi-language examples
  - github.com/apache/beam/tree/master/examples/multi-language

Chamikara Jayalath

QUESTIONS?

www.linkedin.com/in/chamikaramj
https://github.com/chamikaramj