Testing and CI / CD

Vince Gonzalez

Data Engineer, Google Cloud

Agenda

Course Intro

Monitoring

Logging and Error Reporting

Troubleshooting and Debugging

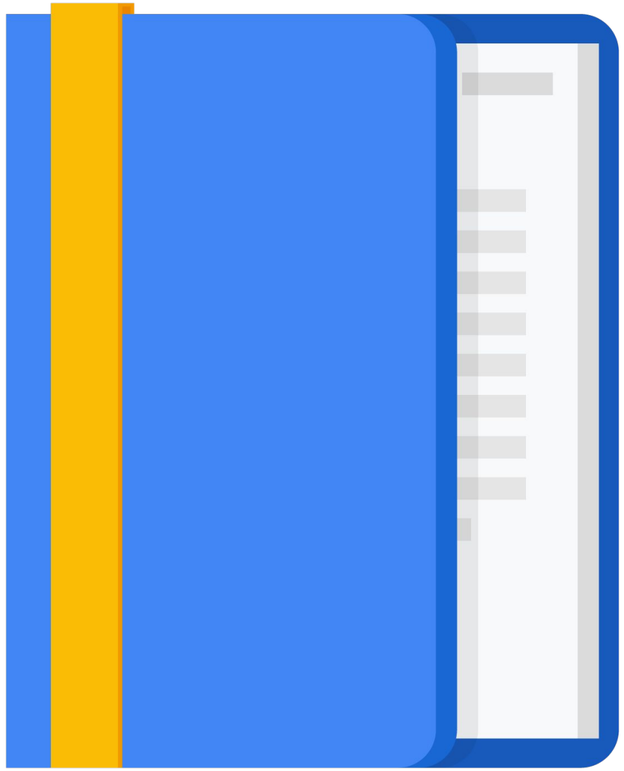
Performance

Testing and CI/CD

Reliability

Flex Templates

Course Summary



Testing and CI/CD

Agenda

Testing and CI/CD overview

Unit testing

Integration testing

Artifact building

Deployment

Testing and CI/CD

Agenda

Testing and CI/CD overview

Unit testing

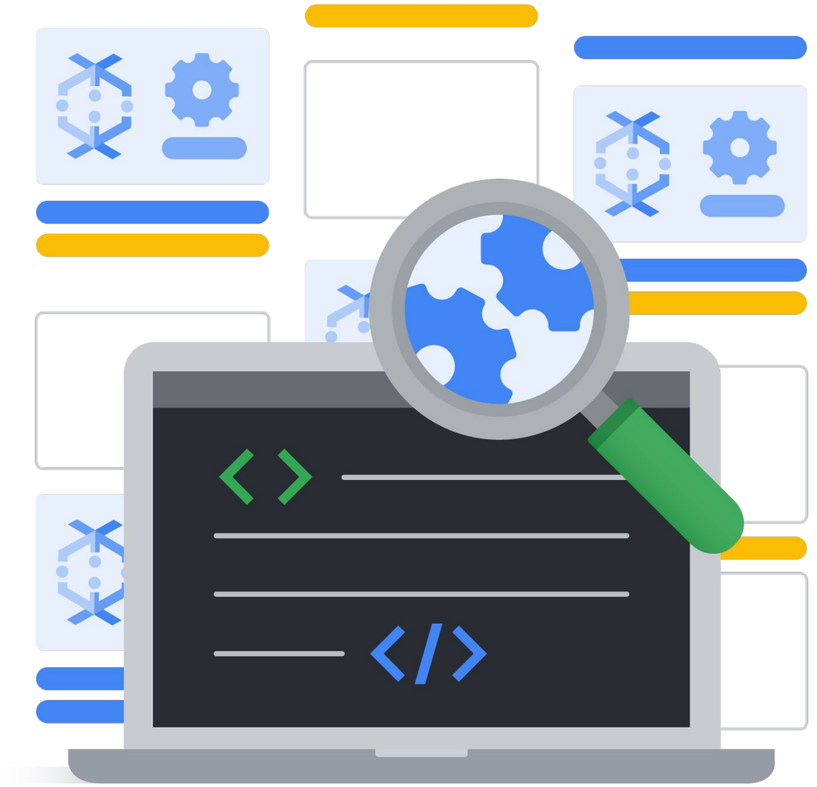
Integration testing

Artifact building

Deployment

Introduction to testing and CI/CD

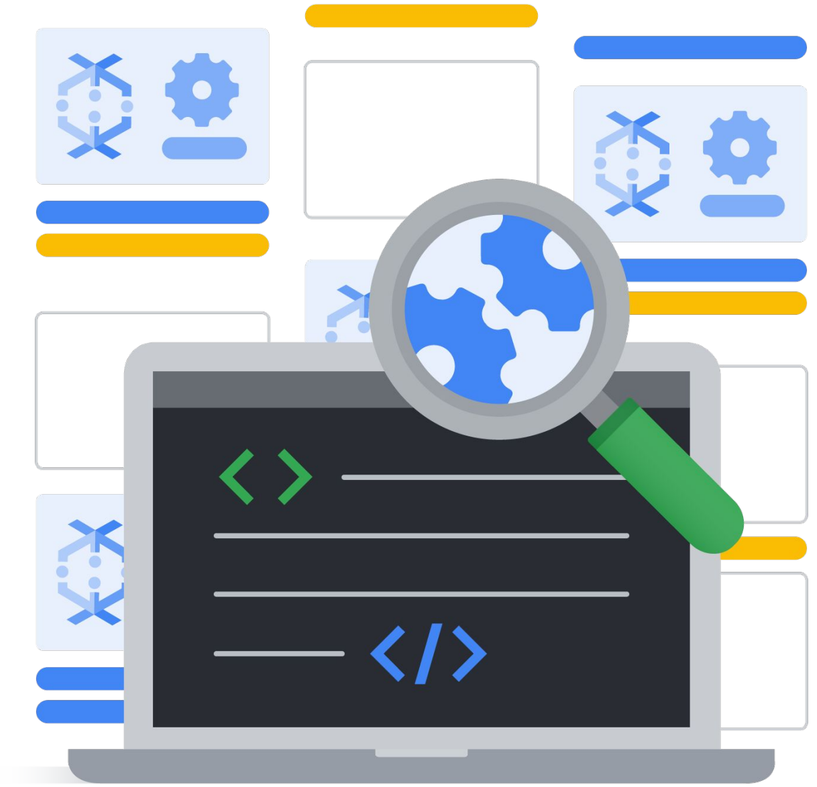
Dataflow pipelines are long-lived applications that require changes. Developers should:



Introduction to testing and CI/CD

Dataflow pipelines are long-lived applications that require changes. Developers should:

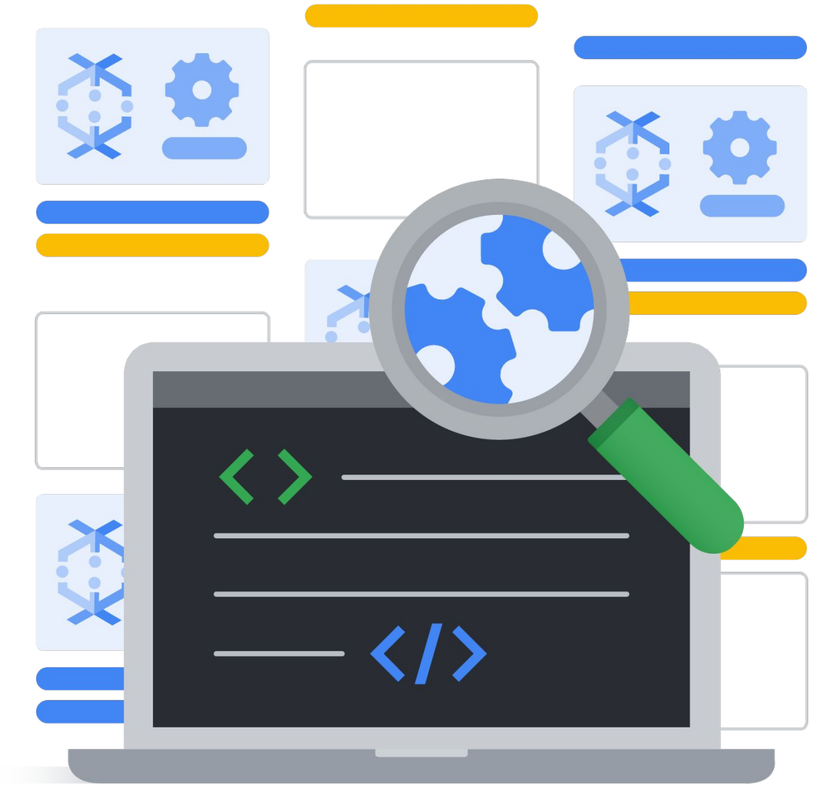
- Implement testing at multiple layers.



Introduction to testing and CI/CD

Dataflow pipelines are long-lived applications that require changes. Developers should:

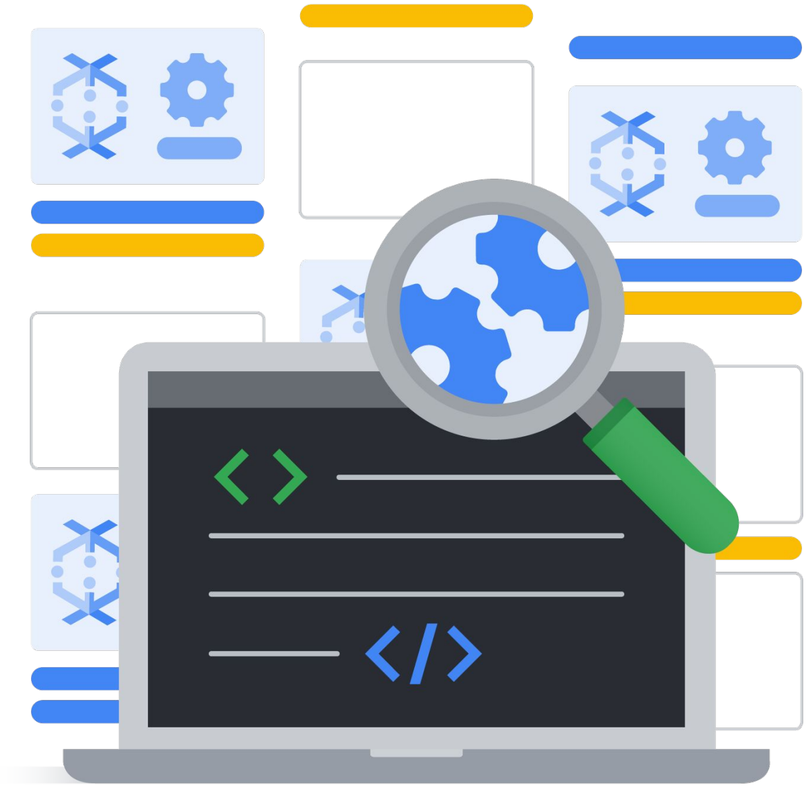
- Implement testing at multiple layers
- Approach production deployments thoughtfully.



Introduction to testing and CI/CD

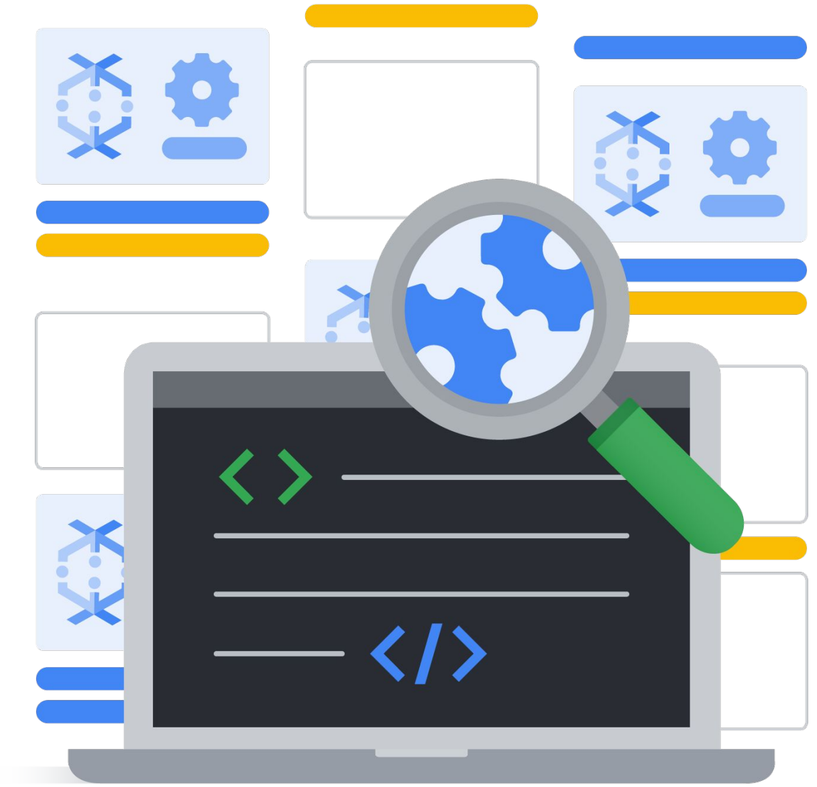
Dataflow pipelines are long-lived applications that require changes. Developers should:

- Implement testing at multiple layers
- Approach production deployments thoughtfully
- Validate changes and rollback, if necessary.



Evolving data processing applications

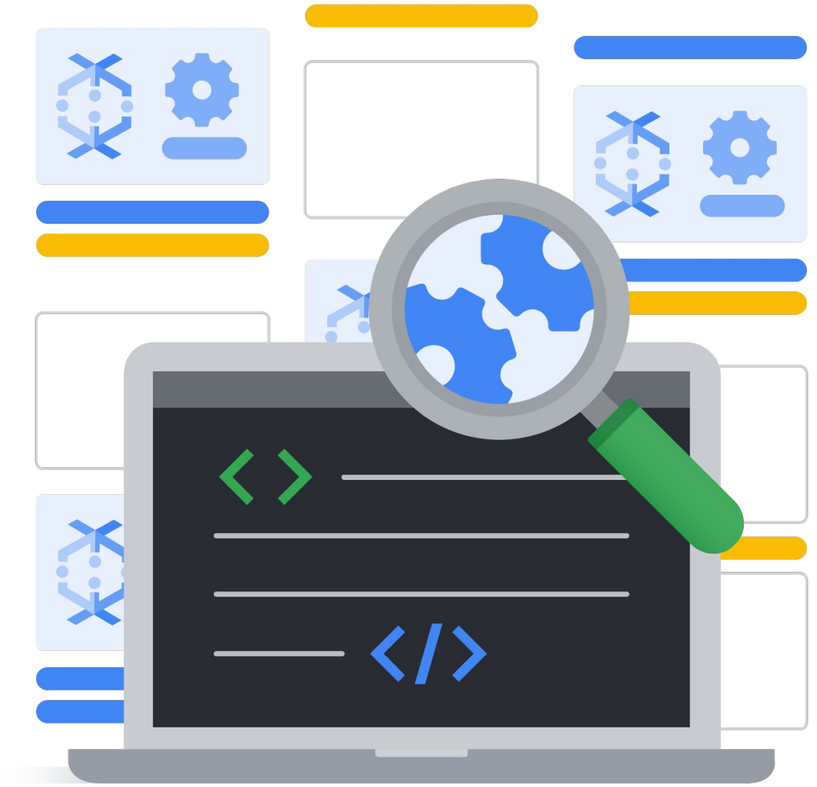
Data processing applications can differ from generic cloud-based applications.



Evolving data processing applications

Data processing applications can differ from generic cloud-based applications.

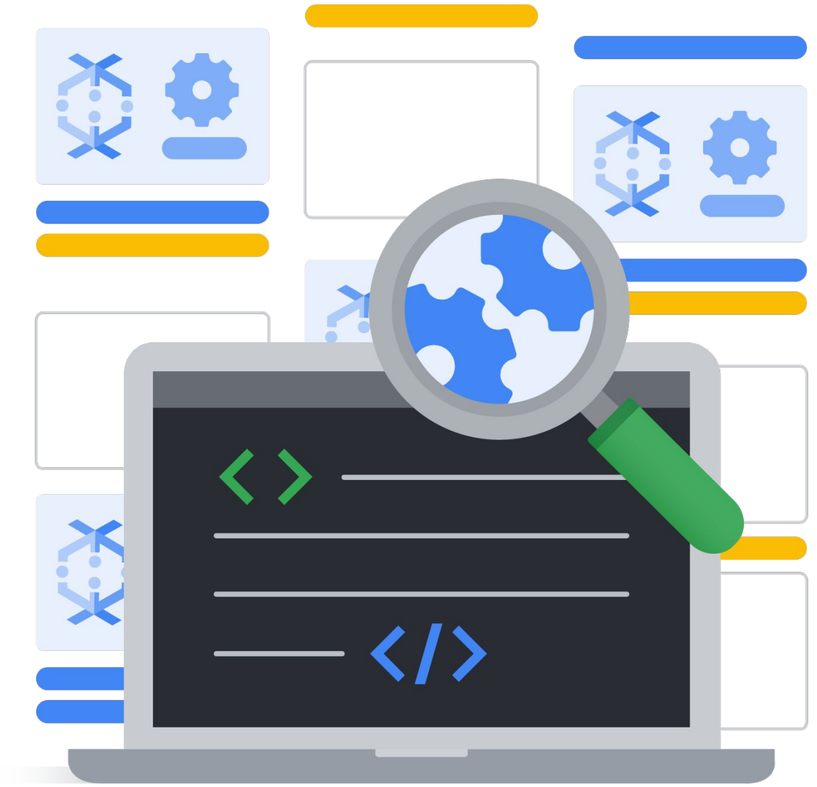
- Pipelines that aggregate data are implicitly stateful.



Evolving data processing applications

Data processing applications can differ from generic cloud-based applications.

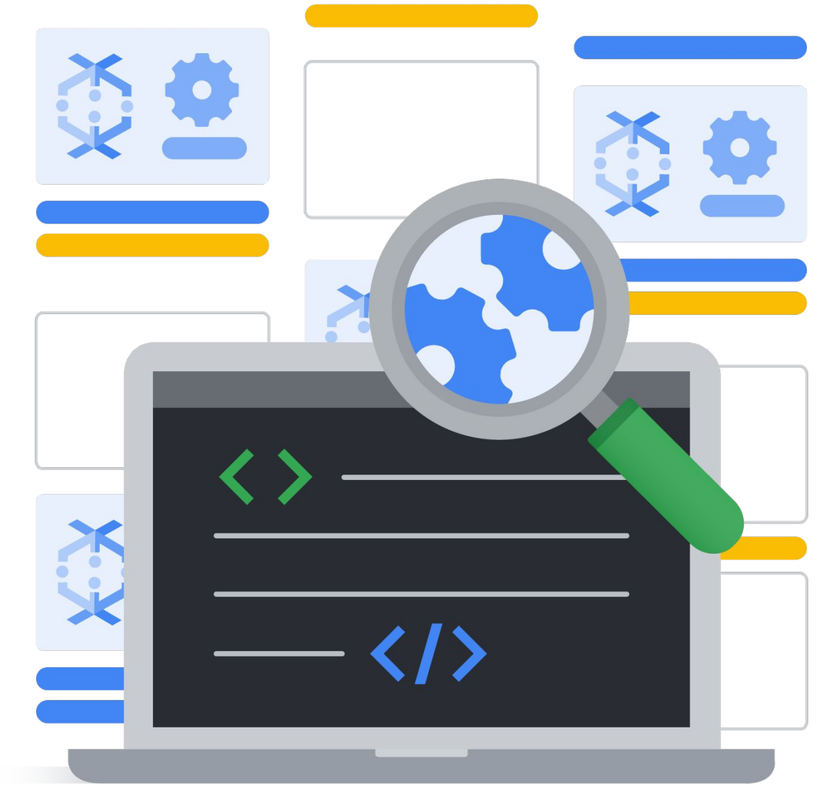
- Pipelines that aggregate data are implicitly stateful.
- Changes to pipeline logic and topology must be able to account for existing state in the original pipeline.



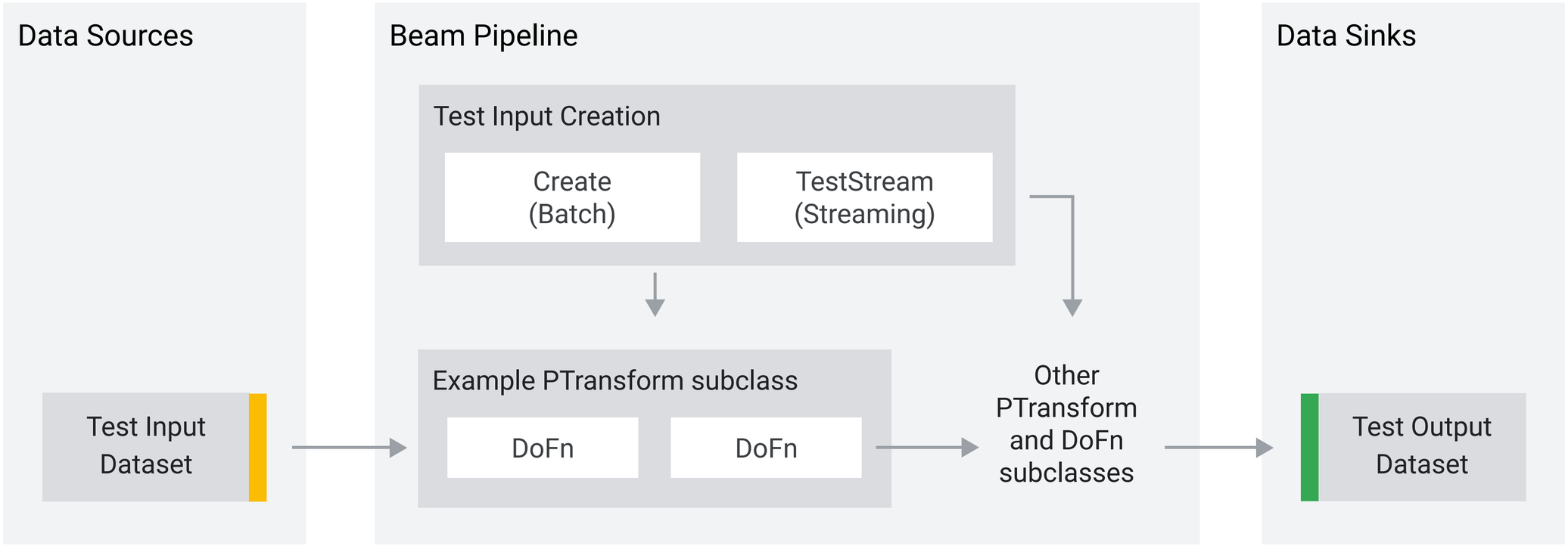
Evolving data processing applications

Data processing applications can differ from generic cloud-based applications.

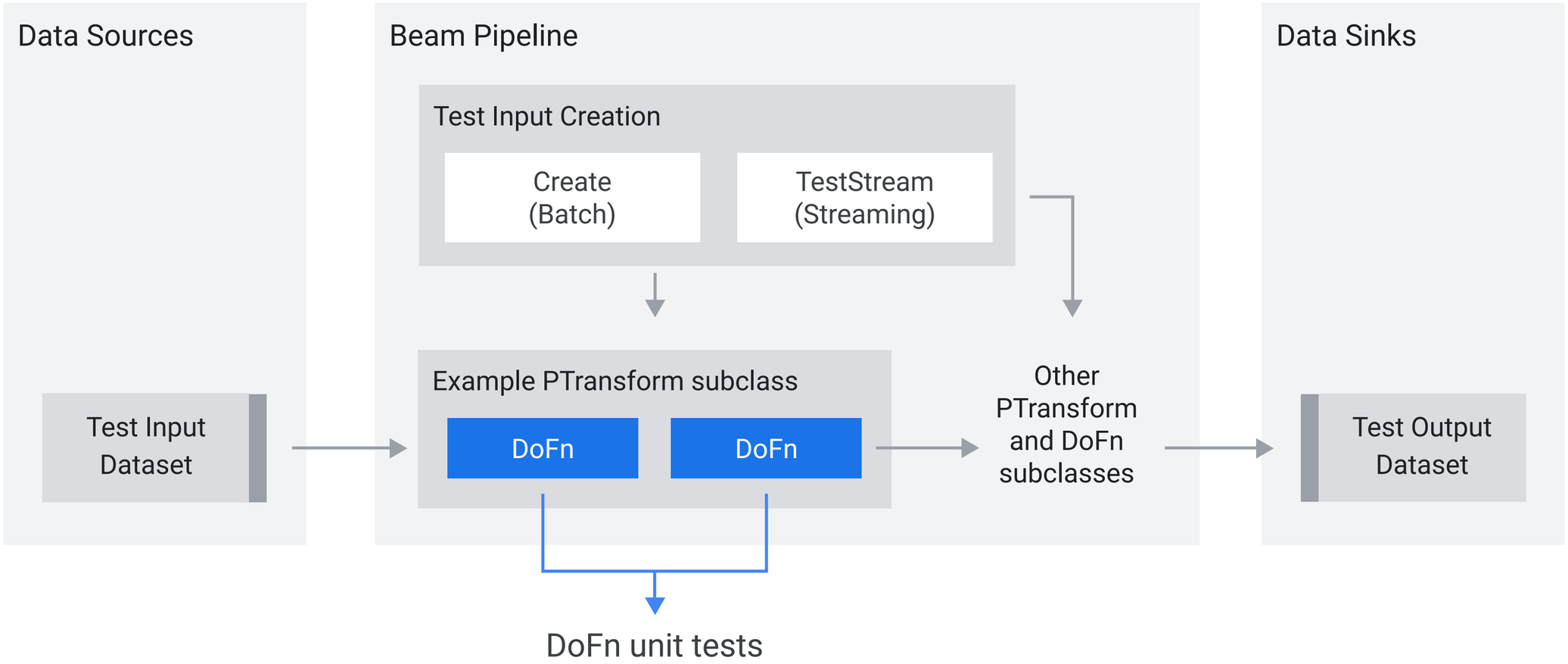
- Pipelines that aggregate data are implicitly stateful.
- Changes to pipeline logic and topology must be able to account for existing state in the original pipeline.
- Data corruption and duplication can happen with pipelines with non-idempotent side effects to external systems.



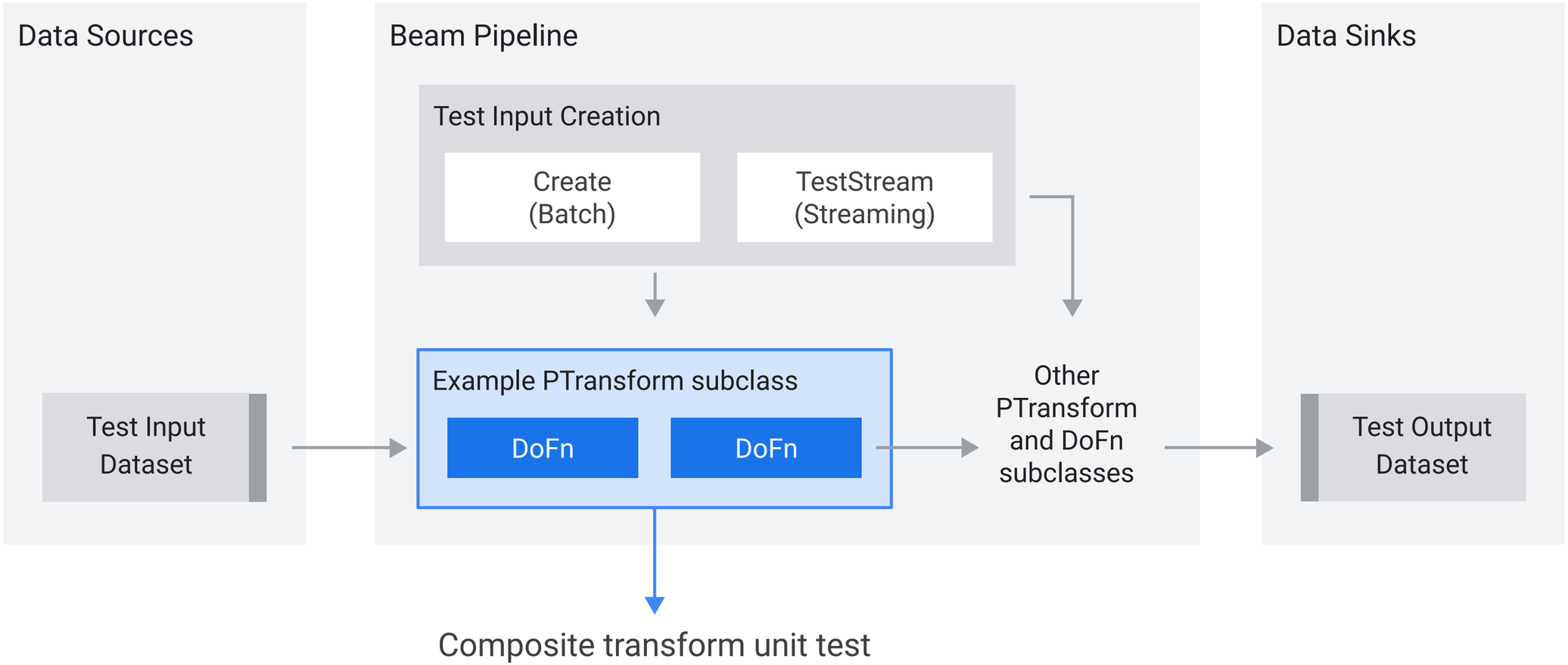
Testing in Beam: End-to-end tests



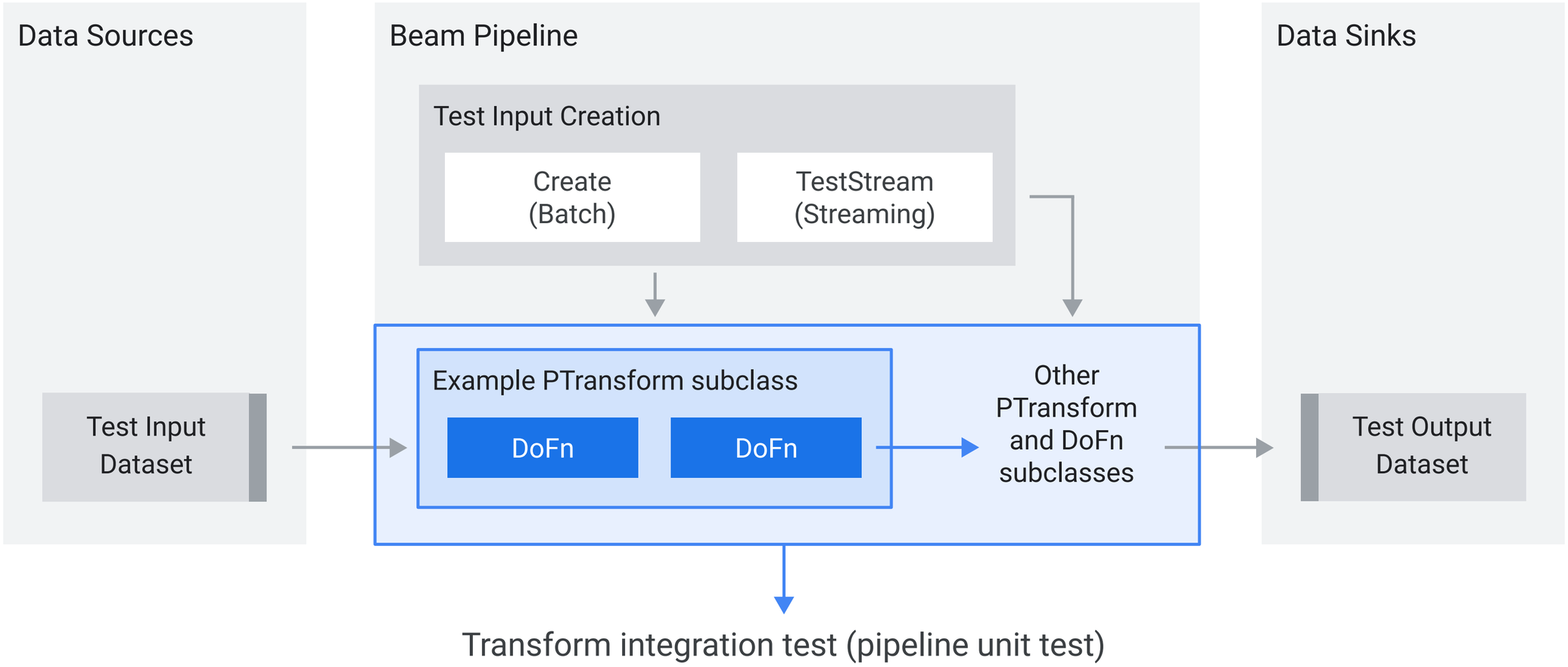
Testing in Beam: End-to-end tests



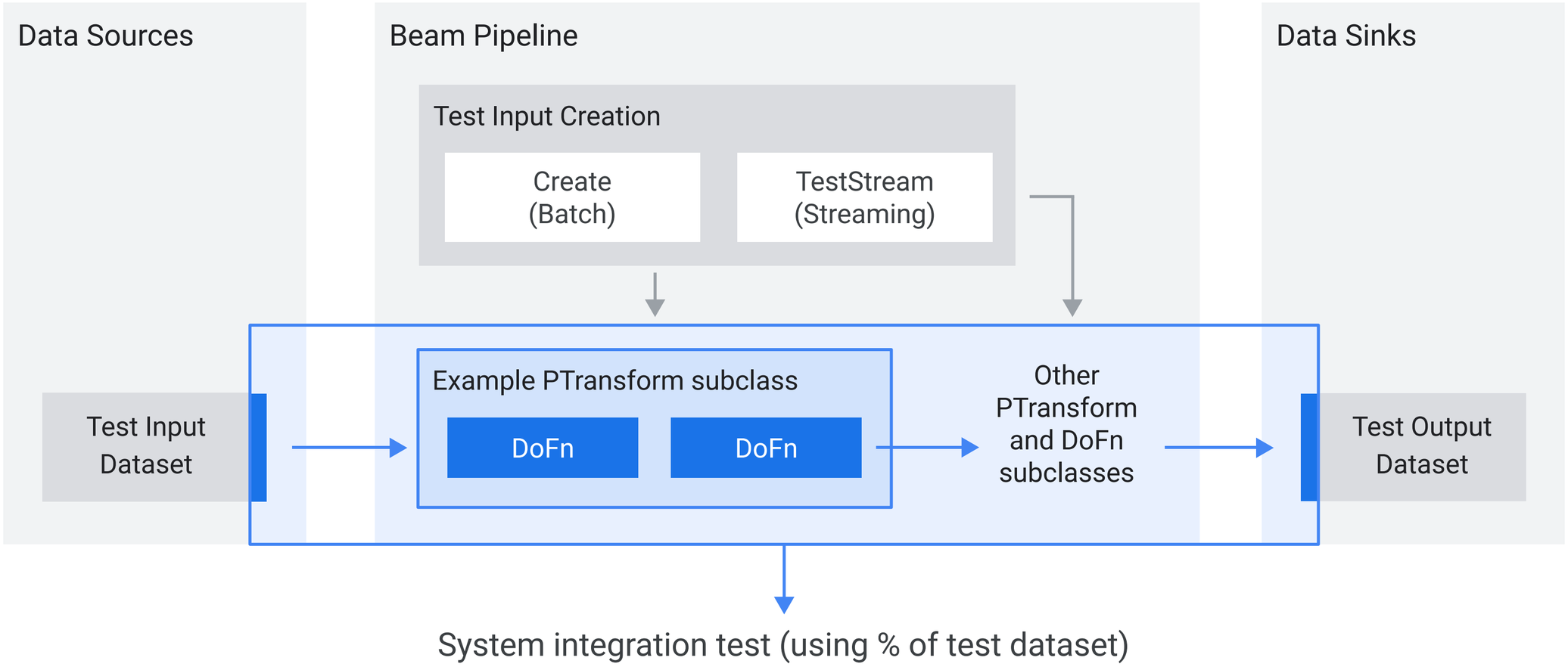
Testing in Beam: End-to-end tests



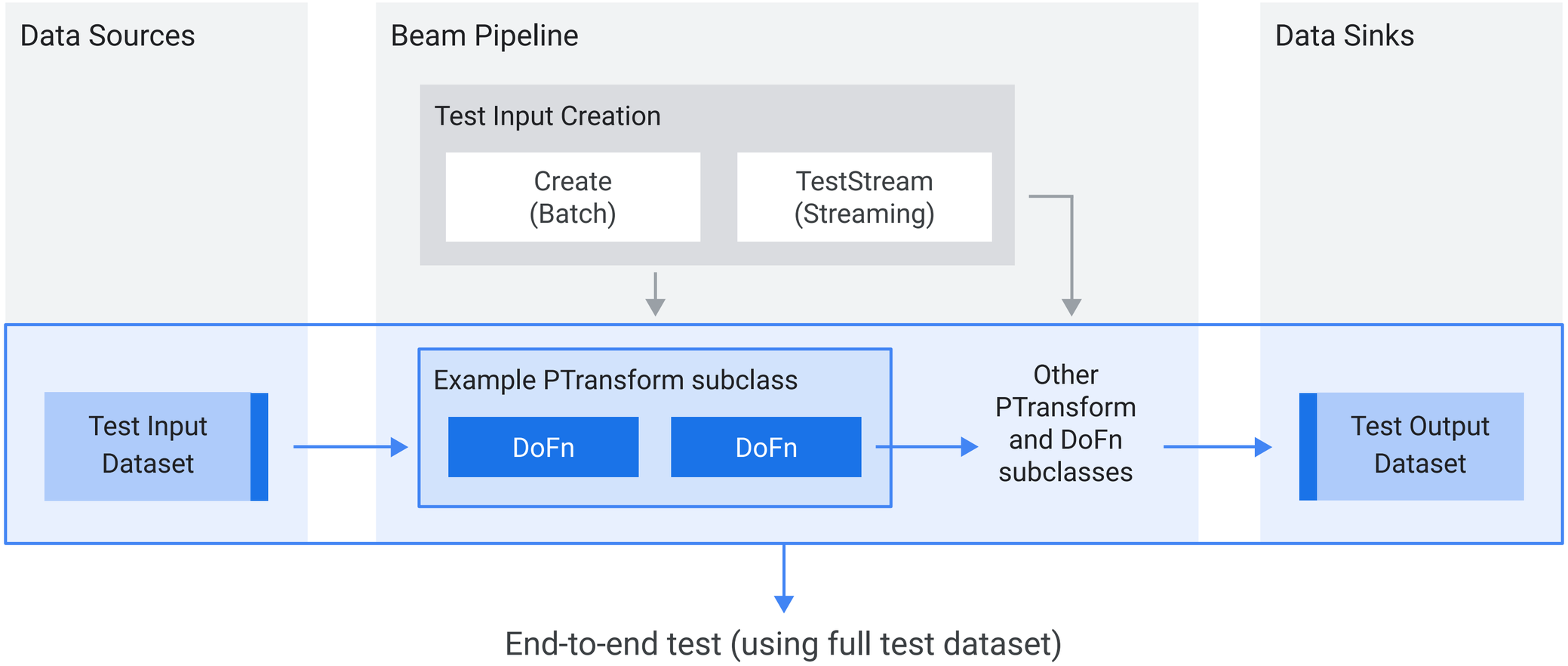
Testing in Beam: End-to-end tests



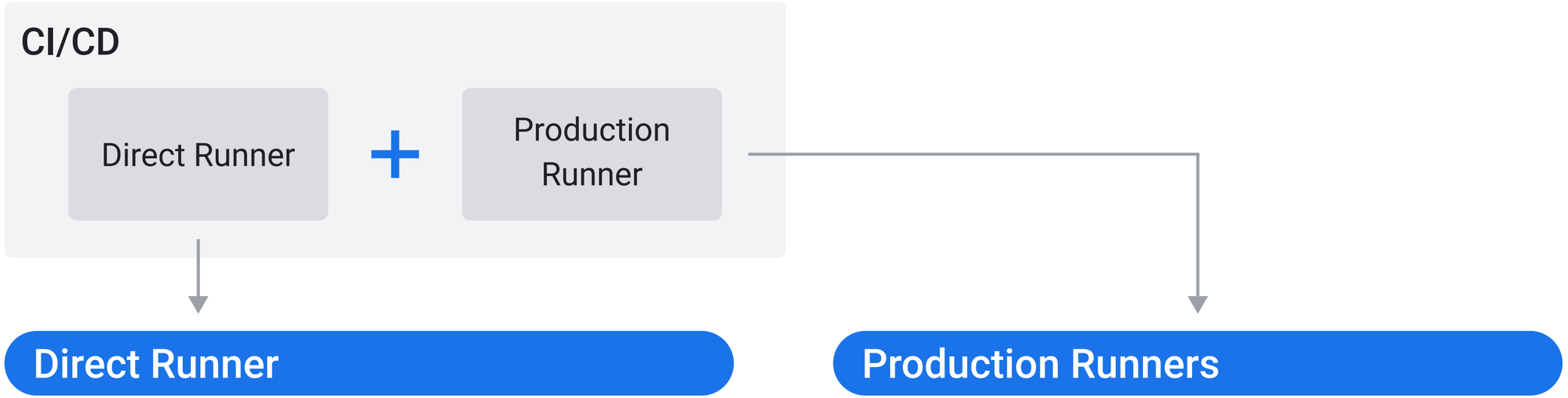
Testing in Beam: End-to-end tests



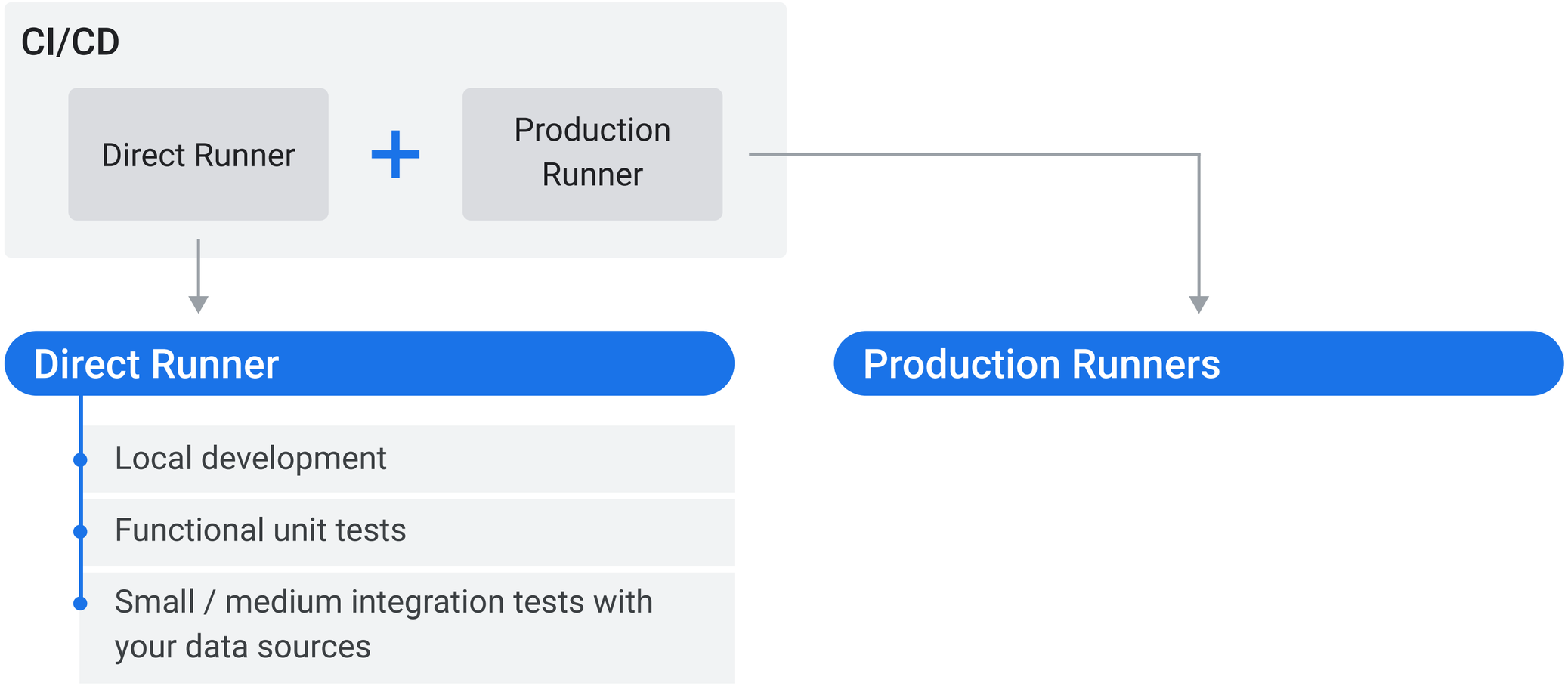
Testing in Beam: End-to-end tests



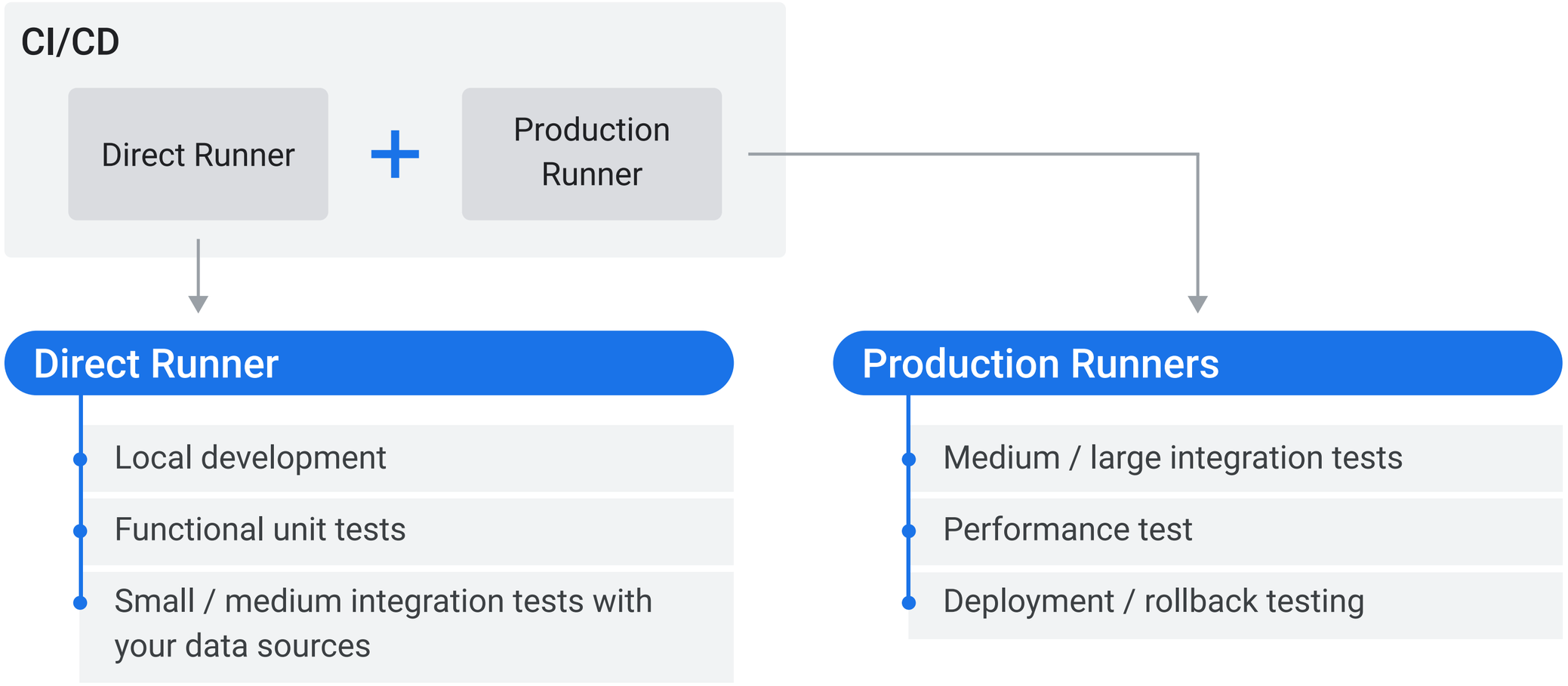
Testing environment



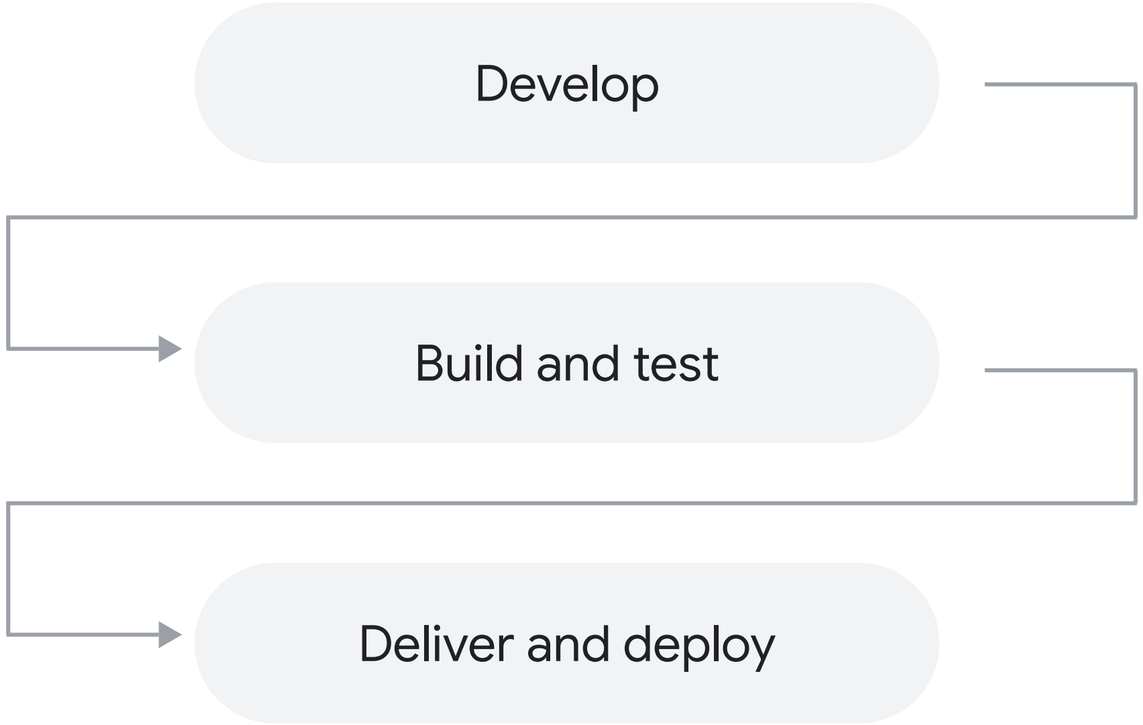
Testing environment



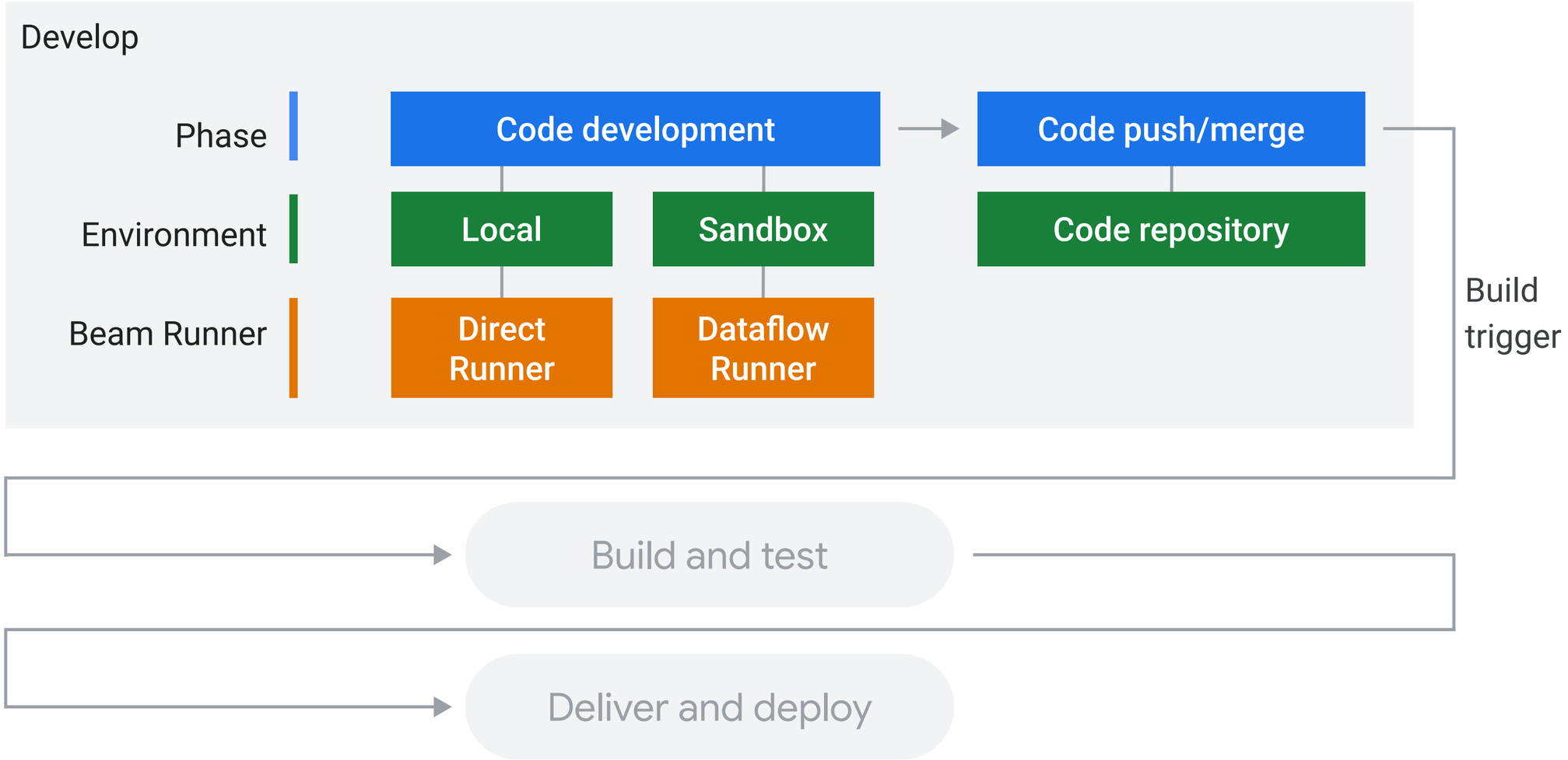
Testing environment



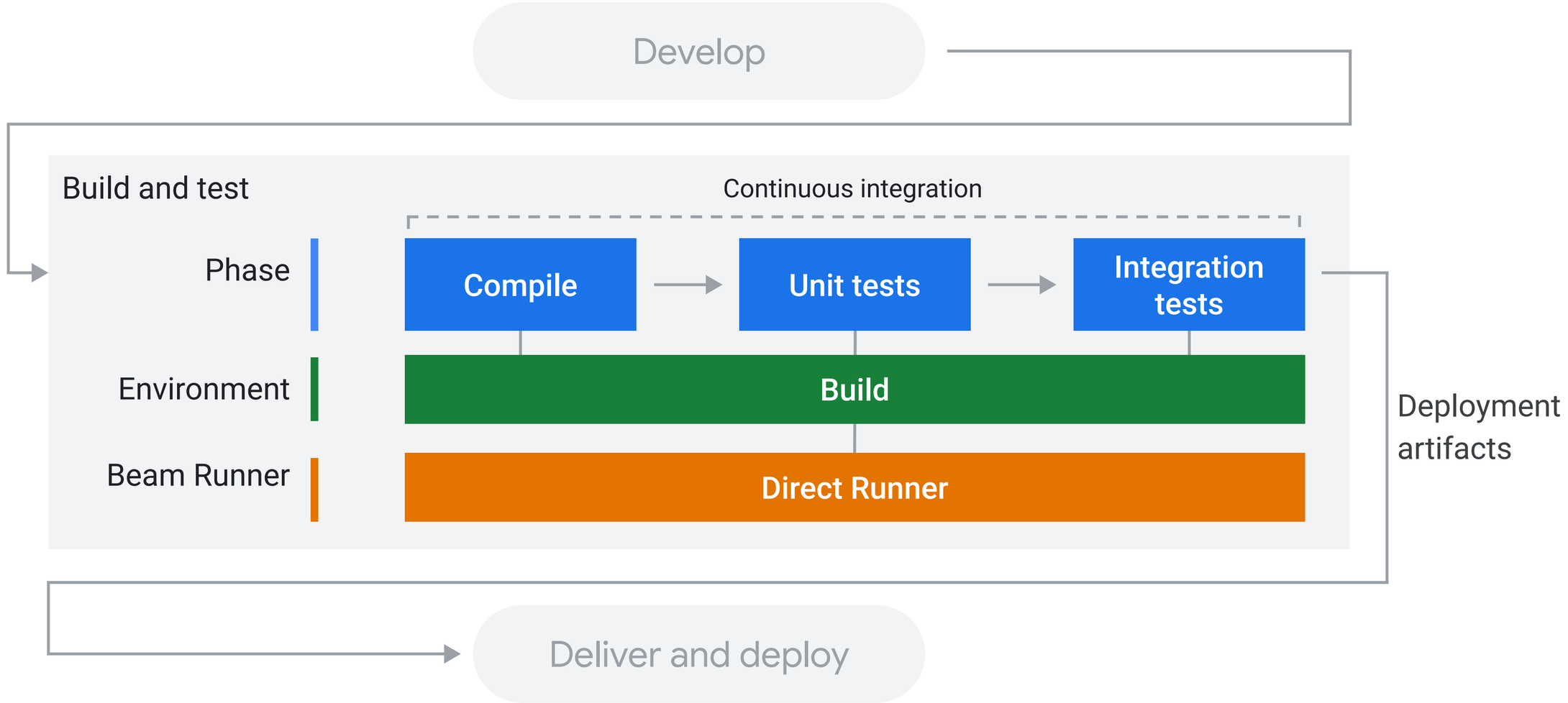
CI/CD lifecycle



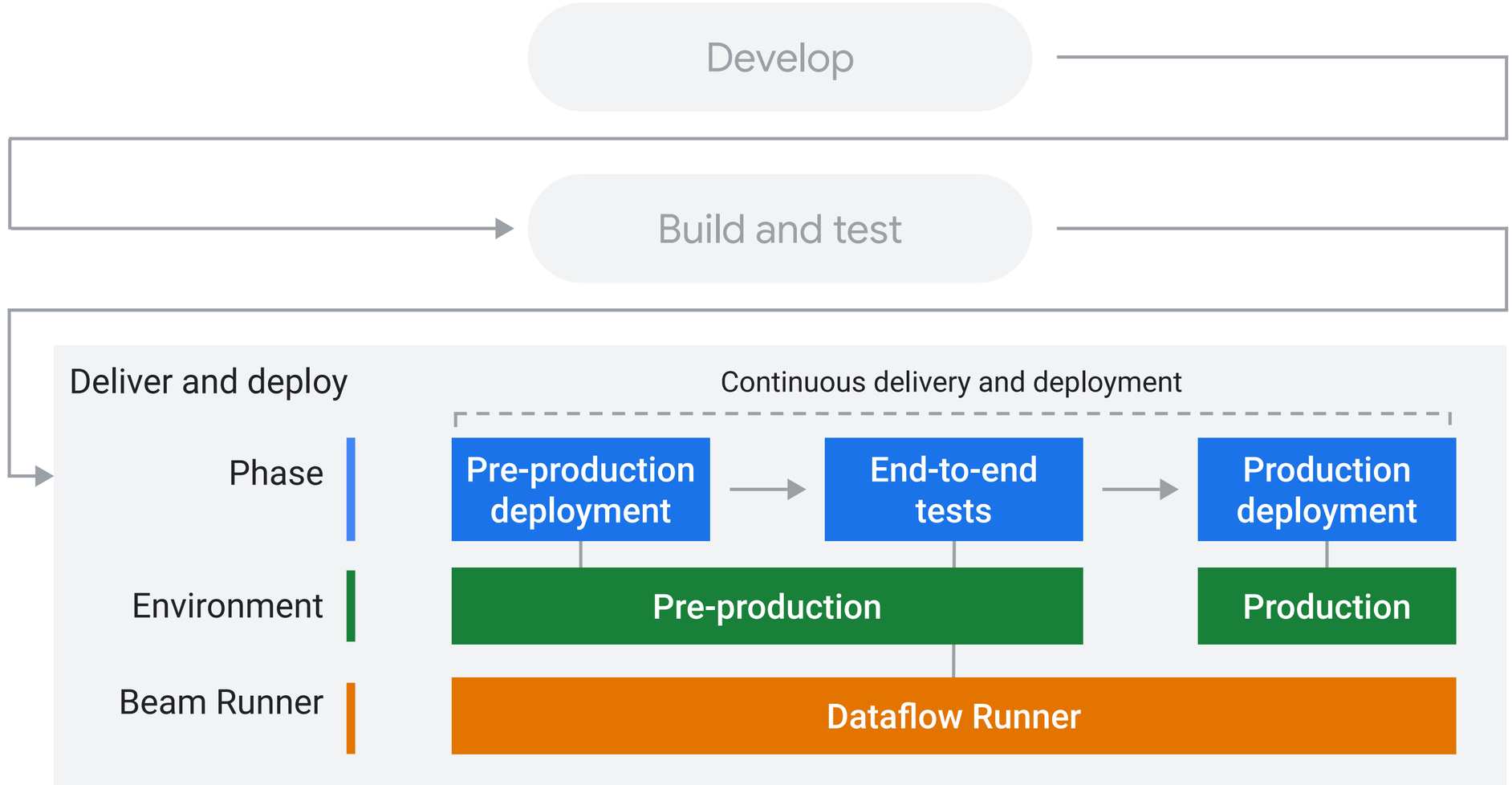
CI/CD lifecycle



CI/CD lifecycle



CI/CD lifecycle



Testing and CI/CD

Agenda

Testing and CI/CD overview

Unit testing

Integration testing

Artifact building

Deployment

Unit tests in Apache Beam

- Unit tests in Beam are mini pipelines that assert that the behavior of small portions of our pipeline is correct.

```
@Rule
public TestPipeline p =
TestPipeline.create();

@Test
public void testASingleTranform() {
  // Setup your PCollection
  // from an in-memory or local data source.
  ...
  // Apply your transform.
  PCollection<String> words = lines.apply(
    ParDo.of(new
WordCount.ExtractWordsFn()));

  // Setup assertions on the pipeline.
  ...
  p.run();
}
```

Unit tests in Apache Beam

- Unit tests in Beam are mini pipelines that assert that the behavior of small portions of our pipeline is correct.
- Unit test your DoFns or PTransforms.

```
@Rule
public TestPipeline p =
TestPipeline.create();

@Test
public void testASingleTranform() {
  // Setup your PCollection
  // from an in-memory or local data source.
  ...
  // Apply your transform.
  PCollection<String> words = lines.apply(
    ParDo.of(new
WordCount.ExtractWordsFn()));

  // Setup assertions on the pipeline.
  ...
  p.run();
}
```

Unit tests in Apache Beam

- Unit tests in Beam are mini pipelines that assert that the behavior of small portions of our pipeline is correct.
- Unit test your DoFns or PTransforms.
- Unit tests should run quickly, locally, and without dependencies on external systems.

```
@Rule
public TestPipeline p =
TestPipeline.create();

@Test
public void testASingleTranform() {
  // Setup your PCollection
  // from an in-memory or local data source.
  ...
  // Apply your transform.
  PCollection<String> words = lines.apply(
    ParDo.of(new
WordCount.ExtractWordsFn()));

  // Setup assertions on the pipeline.
  ...
  p.run();
}
```

Test setup for Apache Beam

Beam uses JUnit 4 for unit testing.

Java: Maven configuration from pom.xml

```
<hamcrest.version>2.1</hamcrest.version>
<junit.version>4.13-beta-3</junit.version>

<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-core</artifactId>
  <version>${hamcrest.version}</version>
</dependency>
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-library</artifactId>
  <version>${hamcrest.version}</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>${junit.version}</version>
</dependency>
```

Testing classes

TestPipeline

Java TestPipeline code

```
@Rule
public final transient TestPipeline p =
   TestPipeline.create();
```

Python TestPipeline code

```
with TestPipeline as p:
    ...
```

Testing classes

PAssert

Java PAssert code

```
@Test
@Category(NeedsRunner.class)
public void myPipelineTest() throws
Exception {
  final PCollection<String> pcol =
p.apply(...)

PAssert.that(pcol).containsInAnyOrder(...);
  p.run();
}
```

Python PAssert code

```
from apache_beam.testing.util import
assert_that
from apache_beam.testing.util import
equal_to

output = ...

# Check whether a PCollection
# contains some elements in any order.
assert_that(
    output,
    equal_to(["elem1", "elem3", "elem2"]))
```

Beam unit testing basics

```
@Rule
public final transient TestPipeline p =
  TestPipeline.create();

@Test
@Category(NeedsRunner.class)
public void myPipelineTest() throws Exception {
  final PCollection<String> pcol = p.apply(...)
  PAssert.that(pcol).containsInAnyOrder(...);
  p.run();
}
```

Anti-pattern

Anonymous DoFns

```
PipelineOptions options =
  PipelineOptionsFactory.create();

Pipeline p = Pipeline.create(options)

PCollection<Integer> output =
    p.apply("Read from text",
TextIO.Read.from(...))
        .apply("Split words", ParDo.of(new DoFn()
{
          // Untestable anonymous transform 1
        }))
        .apply("Generate anagrams", ParDo.of(new
DoFn() {
          // Untestable anonymous transform 2
}))

        .apply("Count words", Count.perElement());
```

Anti-pattern

Anonymous DoFns

```
PipelineOptions options =
  PipelineOptionsFactory.create();

Pipeline p = Pipeline.create(options)

PCollection<Integer> output =
    p.apply("Read from text",
TextIO.Read.from(...))
        .apply("Split words", ParDo.of(new DoFn()
{
          // Untestable anonymous transform 1
        }))
        .apply("Generate anagrams", ParDo.of(new
DoFn() {
          // Untestable anonymous transform 2
}))

        .apply("Count words", Count.perElement());
```

Named subclasses

```
PipelineOptions options =
  PipelineOptionsFactory.create();

Pipeline p = Pipeline.create(options)

PCollection<Integer> output =
    p.apply("Read from text",
TextIO.Read.from(...))
        .apply("Split words",
            ParDo.of(new SplitIntoWordsFn()))
        .apply("Generate anagrams",
            ParDo.of(new GenerateAnagramsFn()))
        .apply("Count words",
            Count.perElement());
```

Testing named subclasses

```
@Rule
public final transient TestPipeline p = TestPipeline.create();

@Test
@Category(NeedsRunner.class)
public void testGenerateAnagramsFn() {
    // Create the test input
    PCollection<String> words = p.apply(Create.of("friend"));

    // Test a single DoFn using the test input
    PCollection<String> anagrams = words.apply("Generate anagrams",
      ParDo.of(new GenerateAnagramsFn()));

    // Assert correct output from
    PAssert.that(anagrams)
      .containsInAnyOrder("finder", "friend", "redfin", "refind");
    p.run();
}
```

Testing windowing behavior

```
@Test
@Category(NeedsRunner.class)
public void testWindowedData() {
 PCollection<String> input =
  p.apply(Create.timestamped(
   TimestampedValue.of("a", new Instant(0L)),
   TimestampedValue.of("a", new Instant(0L)),
   TimestampedValue.of("b", new Instant(0L)),
   TimestampedValue.of("c", new Instant(0L)),
   TimestampedValue.of("c", new Instant(0L)
  .plus(WINDOW_DURATION)))
            .withCoder(StringUtf8Coder.of()));
```

Testing windowing behavior

```
@Test
@Category(NeedsRunner.class)
public void testWindowedData() {
 PCollection<String> input =
  p.apply(Create.timestamped(
   TimestampedValue.of("a", new Instant(0L)),
   TimestampedValue.of("a", new Instant(0L)),
   TimestampedValue.of("b", new Instant(0L)),
   TimestampedValue.of("c", new Instant(0L)),
   TimestampedValue.of("c", new Instant(0L)
  .plus(WINDOW_DURATION)))
            .withCoder(StringUtf8Coder.of()));
```

```
PCollection<KV<String, Long>> windowedCount =
 input.apply(

Window.into(FixedWindows.of(WINDOW_DURATION)
))
     .apply(Count.perElement());

   PAssert.that(windowedCount)
        .containsInAnyOrder(
            // Output from first window
            KV.of("a", 2L),
            KV.of("b", 1L),
            KV.of("c", 1L),
            // Output from second window
            KV.of("c", 1L));

   p.run();
}
```

Testing classes

TestStream

TestStream is a testing input that:

- Generates unbounded PCollection of elements
- Advances the watermark
- Processes time as elements are emitted
- Stops producing output after all specified elements are emitted

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing classes

TestStream

```
@Test
@Category(NeedsRunner.class)
public void testDroppedLateData() {
   TestStream<String> input =
      TestStream.create(StringUtf8Coder.of())
      .addElements(
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("a", new Instant(0L)),
         TimestampedValue.of("b", new Instant(0L)),
         TimestampedValue.of("c", new Instant(0L)
           .plus(Duration.standardMinutes(3))))
      .advanceWatermarkTo(new Instant(0L).plus(WINDOW_DURATION)
        .plus(Duration.standardMinutes(1)))
      .addElements(TimestampedValue.of("c", new Instant(0L)))
      .advanceWatermarkToInfinity();
      PCollection<KV<String, Long>> windowedCount = ...
   PAssert.that(windowedCount)
      .containsInAnyOrder(
        KV.of("a", 2L),
        KV.of("b", 1L),
        KV.of("c", 1L));
   p.run();
}
```

Testing streaming pipelines

Testing complex streaming interactions

```
IntervalWindow window =
    new IntervalWindow(instant, instant.plus(Duration.standardMinutes(5L)));
    PAssert.that(triggered).inFinalPane(window).containsInAnyOrder(1, 2, 3, 4, 5);
    PAssert.that(triggered).inOnTimePane(window).containsInAnyOrder(1, 2, 3);
    PAssert.that(count)
         .inWindow(window)
         .satisfies(
             input -> {
               for (Long count1 : input) {
                 assertThat(count1,
                   allOf(greaterThanOrEqualTo(3L), lessThanOrEqualTo(5L)));
               }
               return null;
             });
```

Testing and CI/CD

Agenda

Testing and CI/CD overview

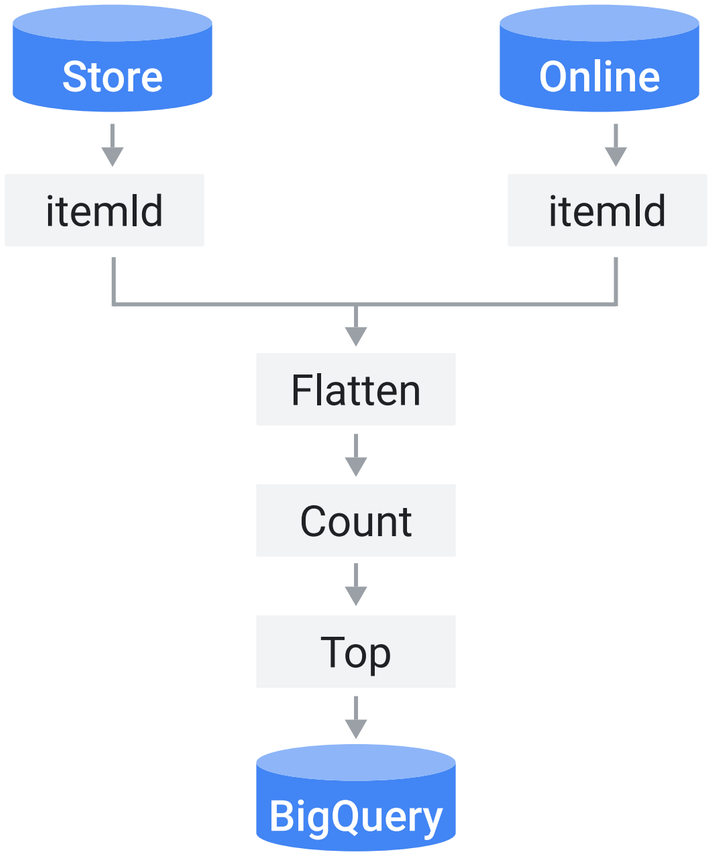
Unit testing

Integration testing

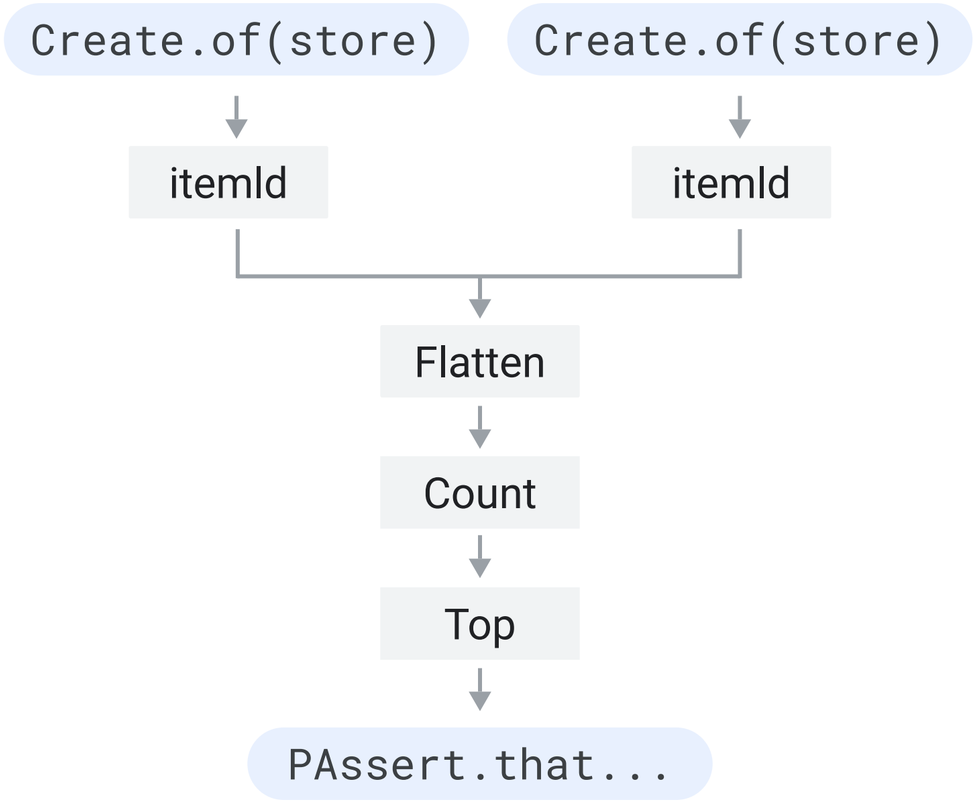
Artifact building

Deployment

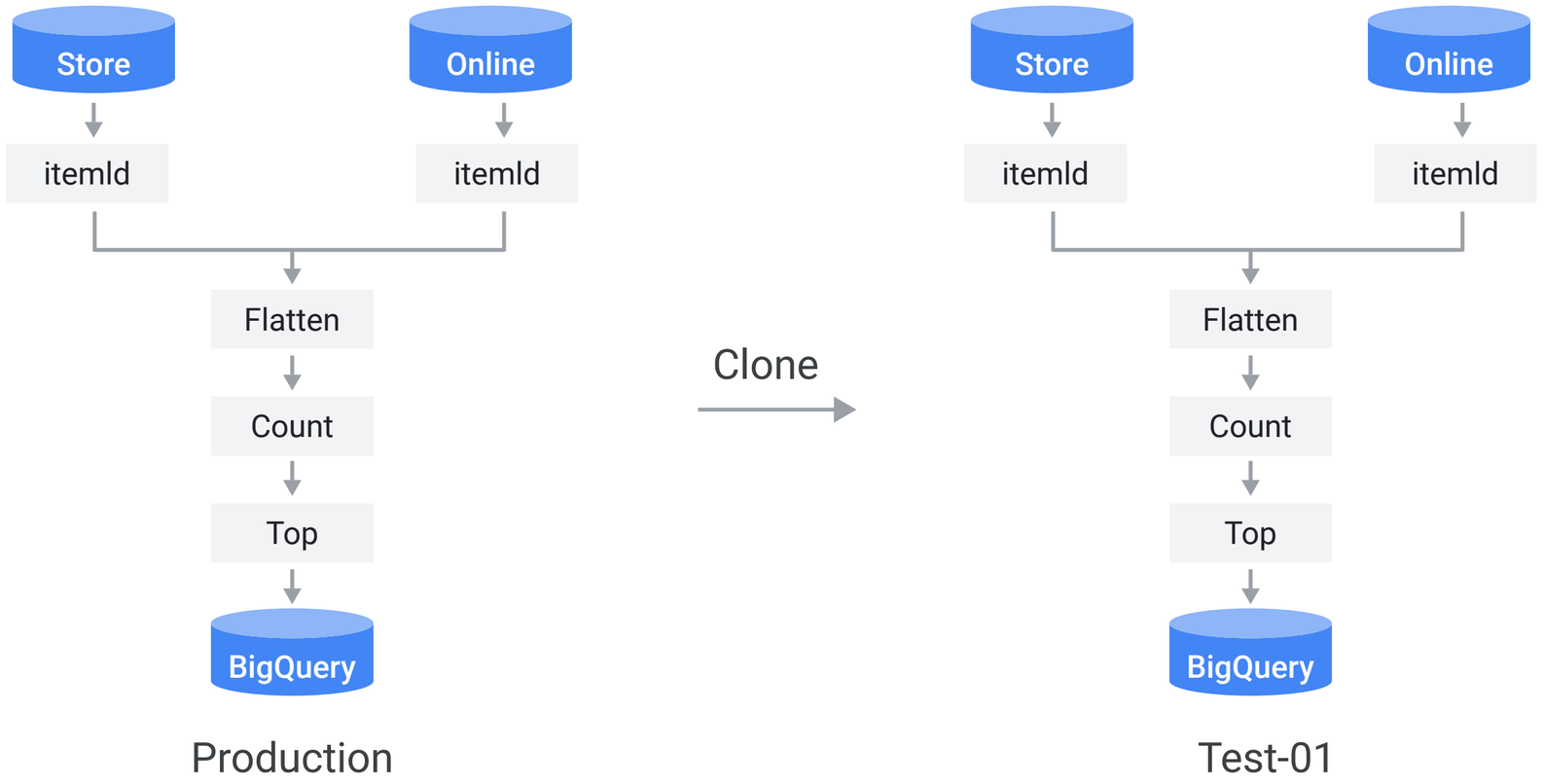
Testing pipelines end to end



Testing pipelines end to end: Small integration

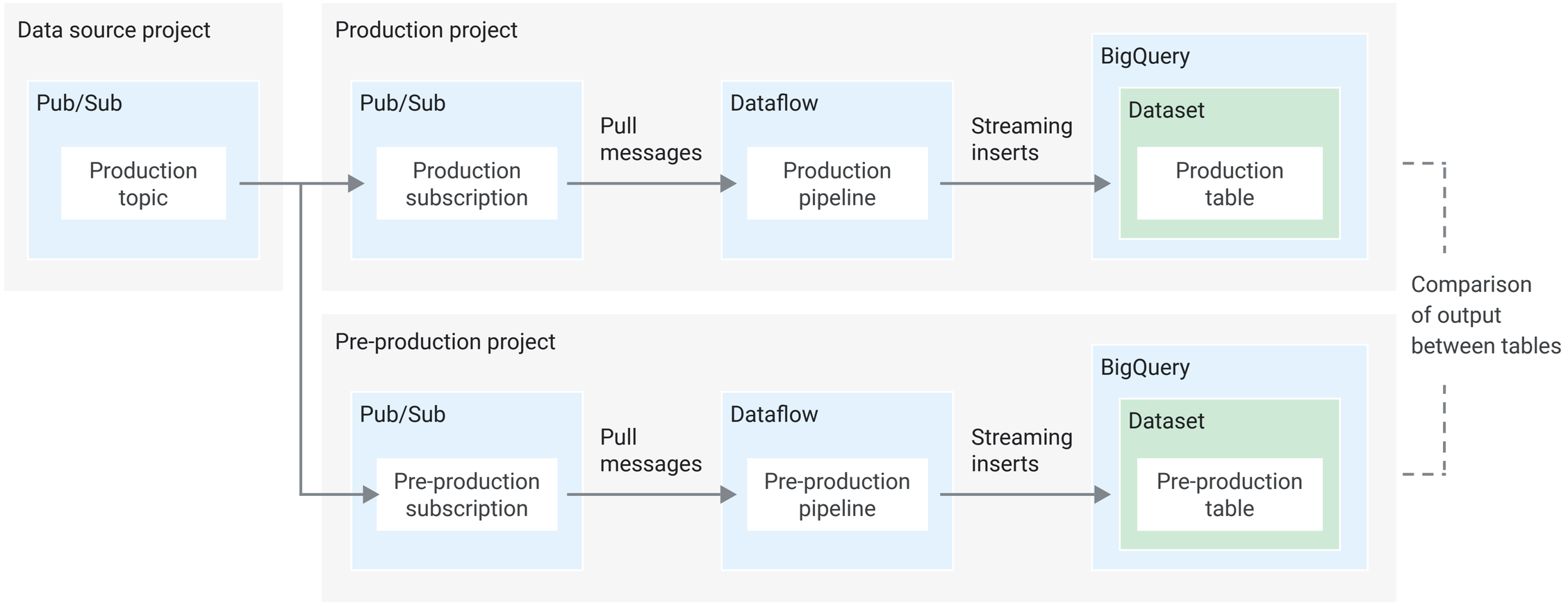


Testing pipelines end to end: Large integration



Testing streaming pipelines: Large integration

Google Cloud



Google Cloud

Beam integration test basics

```
private class WeatherStatsPipeline extends
  PTransform<PCollection<Integer>, PCollection<WeatherSummary>> {
    @Override
    public PCollection<WeatherSummary> expand(PCollection<Integer> input) {
    // Pipeline transforms …
    }
}

@Rule
public final transient TestPipeline p = TestPipeline.create();

@Test
@Category(NeedsRunner.class)
public void testWeatherPipeline() {
  PCollection<Integer> tempCelsius =
    p.apply(Create.of(24, 22, 20, 22, 21, 21, 20));
  PCollection<WeatherSummary> result = tempCelsius.apply(
    "Calculate weather statistics", new WeatherStatsPipeline());
   PAssert.thatSingleton(result)
    .isEqualTo(new WeatherSummary.Builder()
       .withAverageTemp(21)
       .withMaxTemp(24)
       .withMinTemp(20)
       .build());
   p.run();
}
```

Beam integration test basics

```
private class WeatherStatsPipeline extends
  PTransform<PCollection<Integer>, PCollection<WeatherSummary>> {
    @Override
    public PCollection<WeatherSummary> expand(PCollection<Integer> input) {
    // Pipeline transforms …
    }
}

@Rule
public final transient TestPipeline p = TestPipeline.create();

@Test
@Category(NeedsRunner.class)
public void testWeatherPipeline() {
  PCollection<Integer> tempCelsius =
    p.apply(Create.of(24, 22, 20, 22, 21, 21, 20));
  PCollection<WeatherSummary> result = tempCelsius.apply(
    "Calculate weather statistics", new WeatherStatsPipeline());
   PAssert.thatSingleton(result)
    .isEqualTo(new WeatherSummary.Builder()
       .withAverageTemp(21)
       .withMaxTemp(24)
       .withMinTemp(20)
       .build());
   p.run();
}
```

Beam integration test basics

```
private class WeatherStatsPipeline extends
  PTransform<PCollection<Integer>, PCollection<WeatherSummary>> {
    @Override
    public PCollection<WeatherSummary> expand(PCollection<Integer> input) {
    // Pipeline transforms …
    }
}

@Rule
public final transient TestPipeline p = TestPipeline.create();

@Test
@Category(NeedsRunner.class)
public void testWeatherPipeline() {
  PCollection<Integer> tempCelsius =
    p.apply(Create.of(24, 22, 20, 22, 21, 21, 20));
  PCollection<WeatherSummary> result = tempCelsius.apply(
    "Calculate weather statistics", new WeatherStatsPipeline());
   PAssert.thatSingleton(result)
    .isEqualTo(new WeatherSummary.Builder()
       .withAverageTemp(21)
       .withMaxTemp(24)
       .withMinTemp(20)
       .build());
   p.run();
}
```

Testing and CI/CD

Agenda

Testing and CI/CD overview

Unit testing

Integration testing

Artifact building

Deployment

Apache Beam SDK versions

Major.minor.incremental are incremented as follows:

Apache Beam SDK versions

Major.minor.incremental are incremented as follows:

1. Major version for incompatible API changes

Apache Beam SDK versions

Major.minor.incremental are incremented as follows:

1. Major version for incompatible API changes
2. Minor version for new functionality added in a backward-compatible manner

Apache Beam SDK versions

Major.minor.incremental are incremented as follows:

1. Major version for incompatible API changes
2. Minor version for new functionality added in a backward-compatible manner
3. Incremental version for forward-compatible bug fixes

Apache Beam SDK — Maven Central

Core Java SDK

```
<groupId>org.apache.beam</groupId>
<artifactId>beam-sdks-java-core</artifactId>
```

Dataflow Runner

```
<groupId>org.apache.beam</groupId>
<artifactId>beam-runners-google-cloud-dataflow-java</artifactId>
```

IOs

```
<groupId>org.apache.beam</groupId>
<artifactId>beam-sdks-java-io-google-cloud-platform</artifactId>
```

Testing and CI/CD

Agenda

Testing and CI/CD overview

Unit testing

Integration testing

Artifact building

Deployment

Job deployment

1 Deployment

2 In-flight

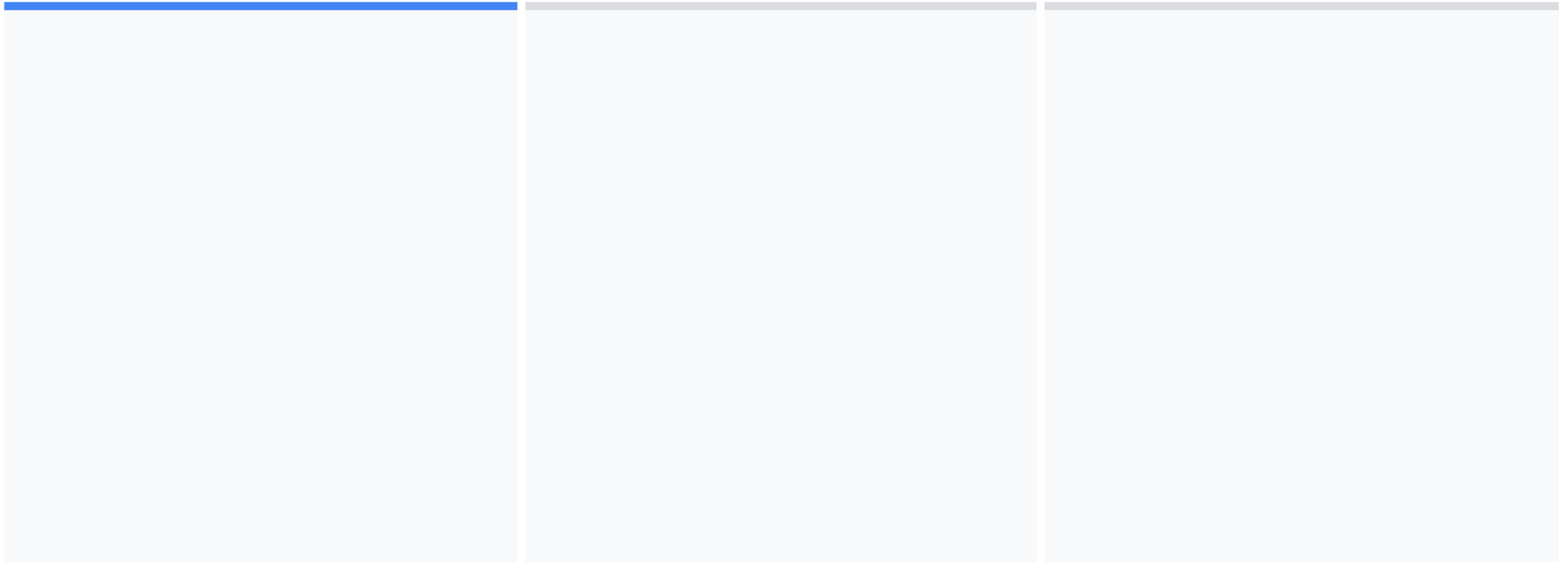
3 Termination

Job deployment

1 Deployment

2 In-flight

3 Termination



Job deployment

1 Deployment

2 In-flight

3 Termination

Direct launch

Launch from development environment (Gradle or Maven for Java, executing the program for Python).

Pipeline type

Batch & Streaming

Job deployment

1 Deployment

2 In-flight

3 Termination

Direct launch

Launch from development environment (Gradle or Maven for Java, executing the program for Python).

Pipeline type

Batch & Streaming

Templates

Launch job outside of developer environment (UI, command line, or REST API call).

Pipeline type

Batch & Streaming

In-flight actions

1 Deployment

2 In-flight

3 Termination

In-flight actions

1 Deployment

2 In-flight

3 Termination

Snapshot

Save the state of a streaming pipeline and launch new versions without losing state.

Pipeline type

Streaming

In-flight actions

1 Deployment

2 In-flight

3 Termination

Snapshot

Save the state of a streaming pipeline and launch new versions without losing state.

Pipeline type

Streaming

Update

Replace your existing job with a new job that runs updated pipeline code.

Pipeline type

Streaming

In-flight actions

Snapshot

1. Testing and rolling back updates

In-flight actions

Snapshot

1. Testing and rolling back updates
2. Backup and recovery

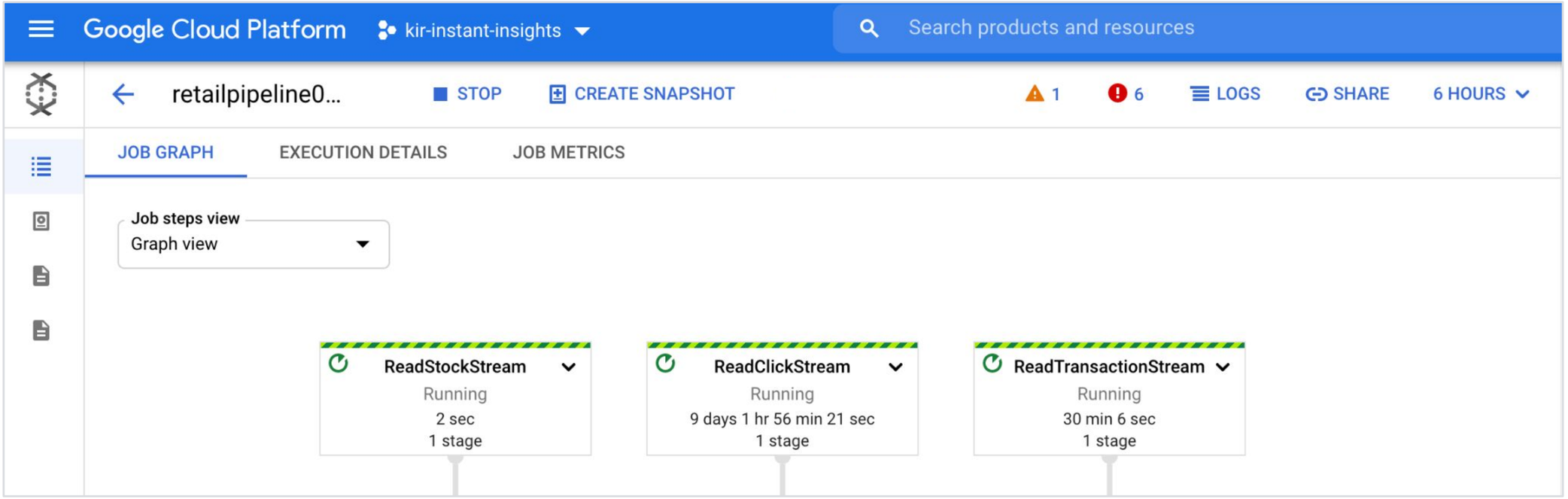
In-flight actions

Snapshot

1. Testing and rolling back updates
2. Backup and recovery
3. Migrate to Streaming Engine

In-flight actions

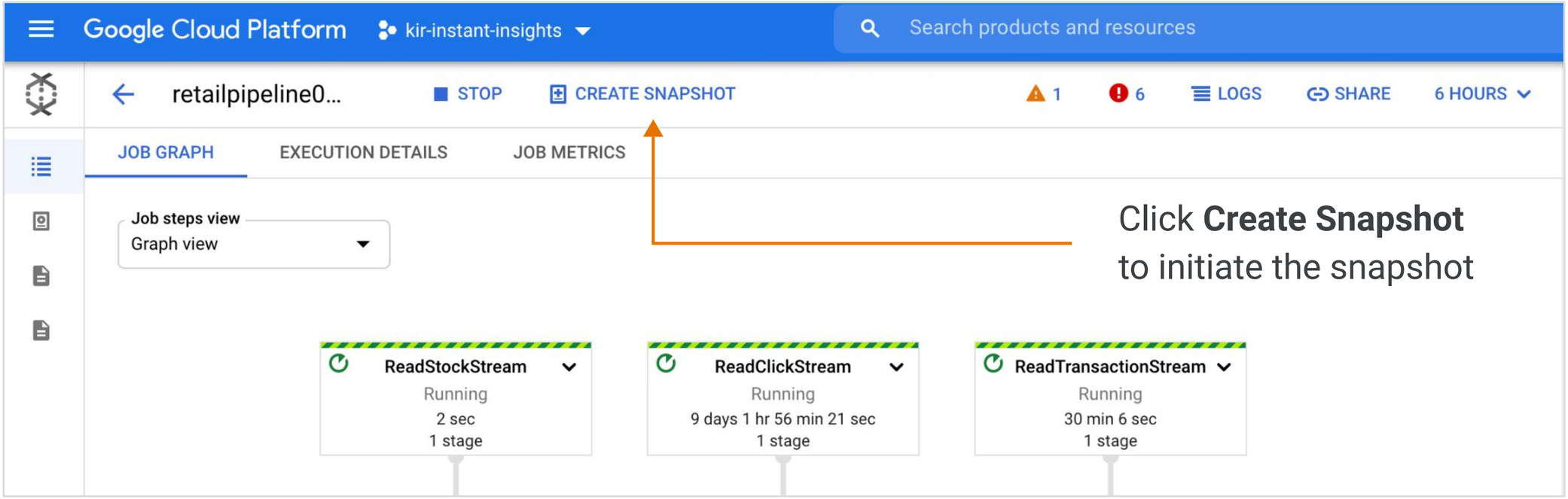
Snapshotting your job



*Job snapshots can also be created via the CLI and API

In-flight actions

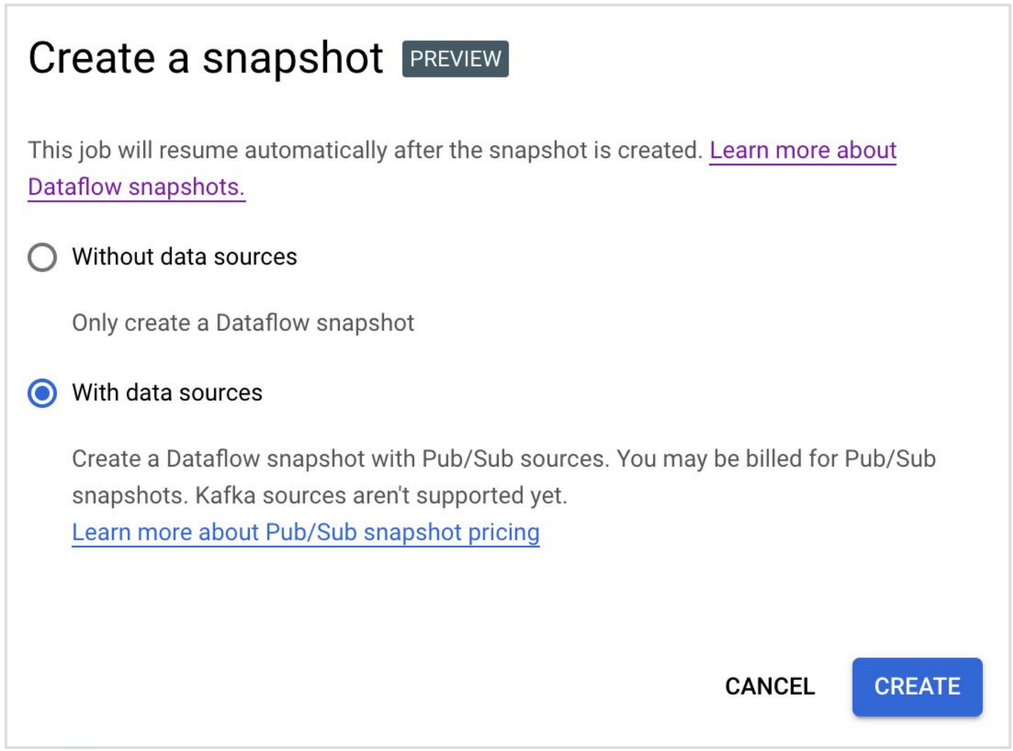
Snapshotting your job



*Job snapshots can also be created via the CLI and API

In-flight actions

Snapshotting your job



*Job snapshots can also be created via the CLI and API

In-flight actions

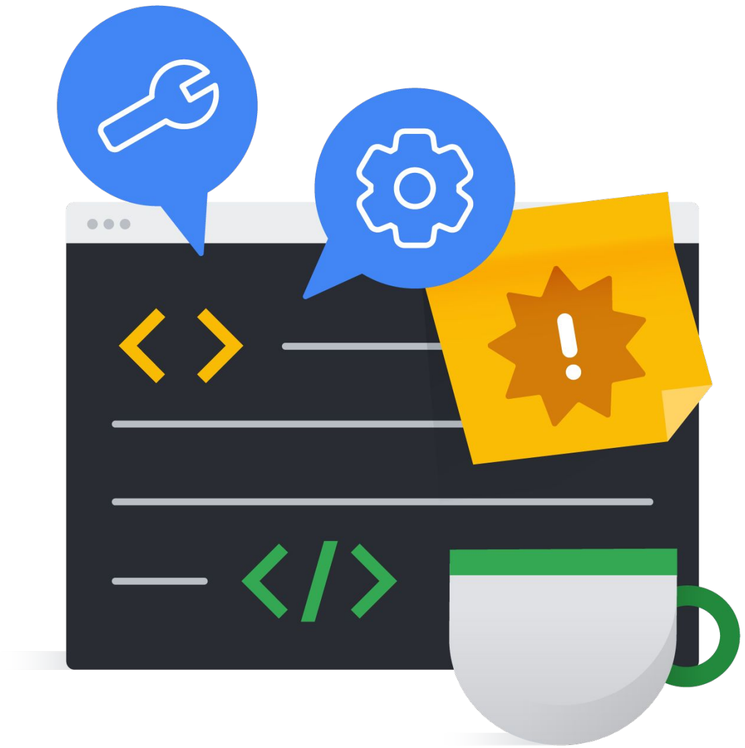
Creating a job from a snapshot

Java example using Maven

```
mvn -Pdataflow-runner compile exec:java \
    -Dexec.mainClass=org.apache.beam.examples.WordCount \
    -Dexec.args="--project=PROJECT_ID \
    --stagingLocation=gs://STORAGE_BUCKET/staging/ \
    --inputFile=gs://apache-beam-samples/shakespeare/* \
    --output=gs://STORAGE_BUCKET/output \
    --runner=DataflowRunner \
    --enableStreamingEngine \
    --createFromSnapshot=SNAPSHOT_ID \
    --region=REGION"
```

In-flight actions

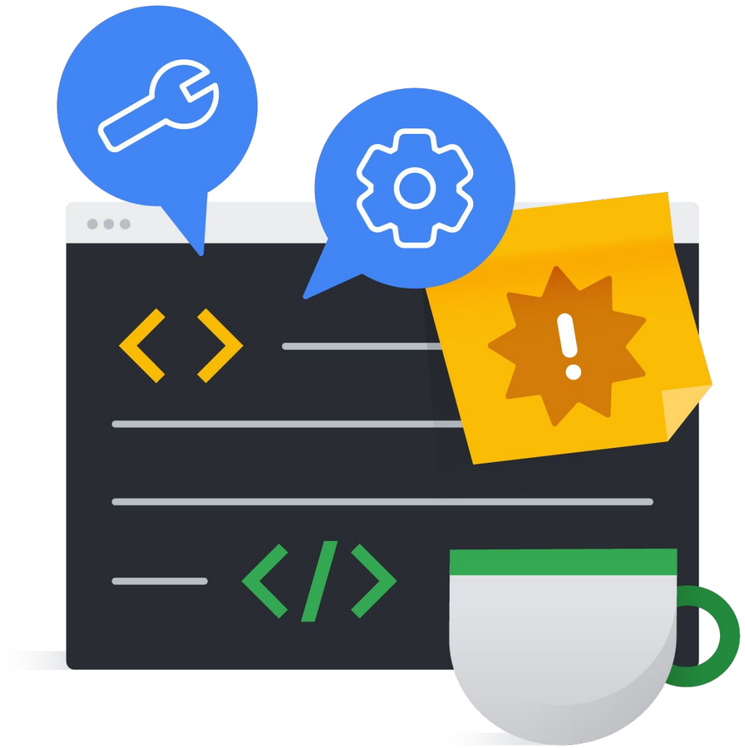
Update



In-flight actions

Update

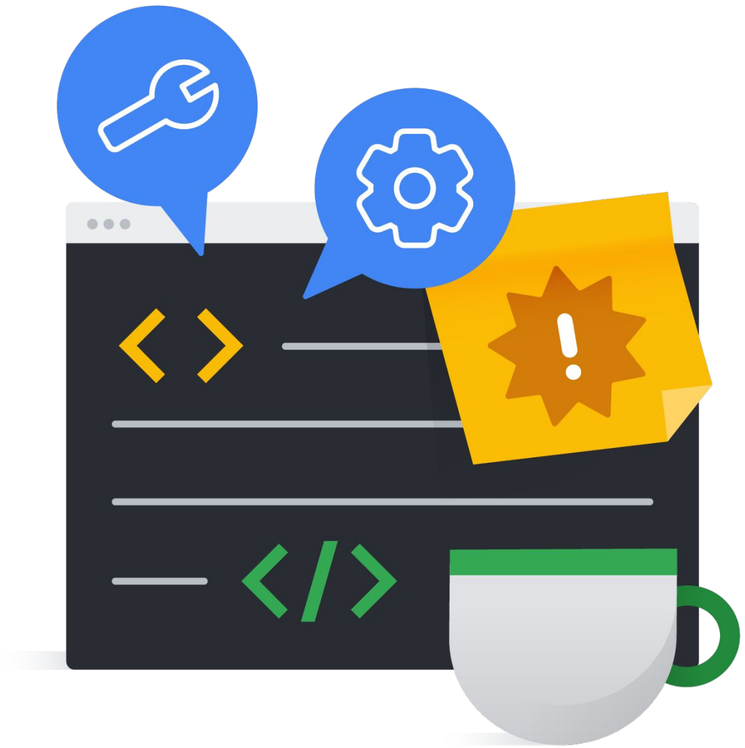
1. Improve your pipeline code



In-flight actions

Update

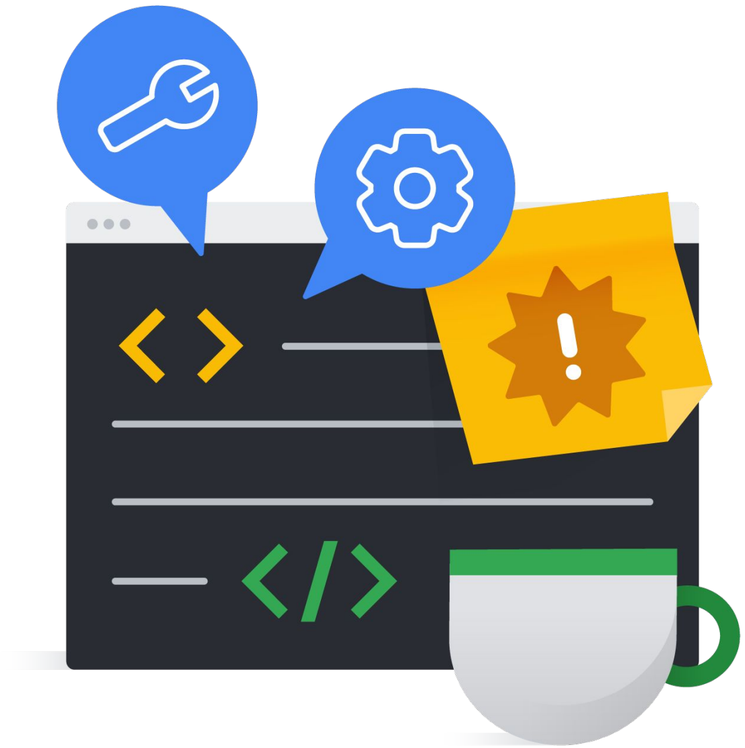
1. Improve your pipeline code
2. Fix bugs in your pipeline code



In-flight actions

Update

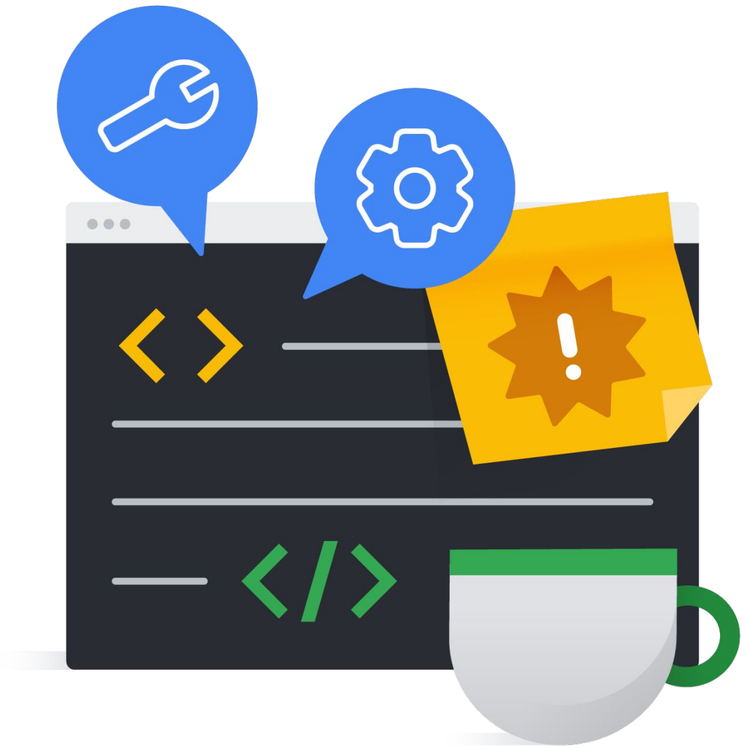
1. Improve your pipeline code
2. Fix bugs in your pipeline code
3. Update your pipeline



In-flight actions

Update

1. Improve your pipeline code
2. Fix bugs in your pipeline code
3. Update your pipeline
4. Account for changes in your data source



In-flight actions

Update

Java command

```
mvn -Pdataflow-runner compile exec:java \

-Dexec.mainClass=org.apache.beam.examples.WordCount \
    -Dexec.args="--project=PROJECT_ID \
    --stagingLocation=gs://STORAGE_BUCKET/staging/ \
--inputFile=gs://apache-beam-samples/shakespeare/* \
    --output=gs://STORAGE_BUCKET/output \

    --runner=DataflowRunner \
    --update \
    --jobName [prior job name]  \
--transformNameMapping='{"oldTransform1":"newTransform1","oldTransform2":"newTransform2",...}'
    --region=REGION"
```

Python command

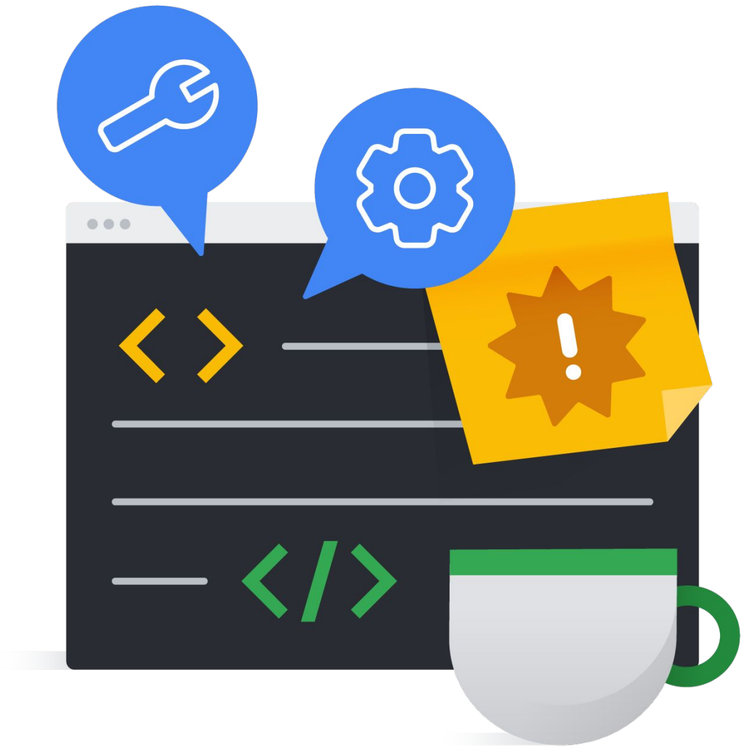
```
python -m apache_beam.examples.wordcount \


  --project $PROJECT \
  --staging_location gs://$BUCKET/tmp/
  --input
gs://dataflow-samples/shakespeare/kinglear.txt \
  --output gs://$BUCKET/results/outputs \
  --runner DataflowRunner \
  --update \
  --job_name [prior job name] \
--transform_name_mapping=='{"oldTransform1":"newTransform1","oldTransform2":"newTransform2",...}'
  --region $REGION \
```

*Job updates can also be created via the API

In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

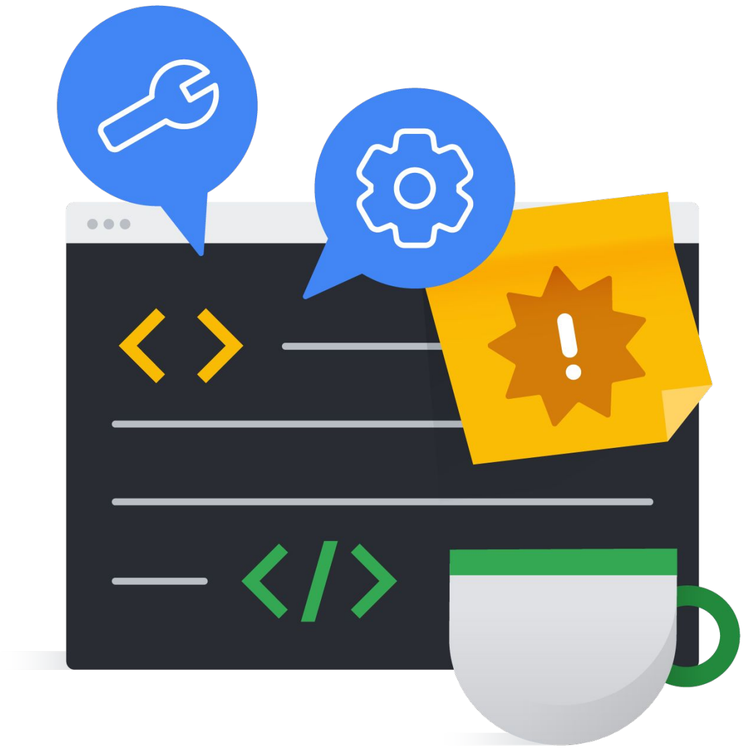


In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

- Modifying pipeline graph without a transform mapping

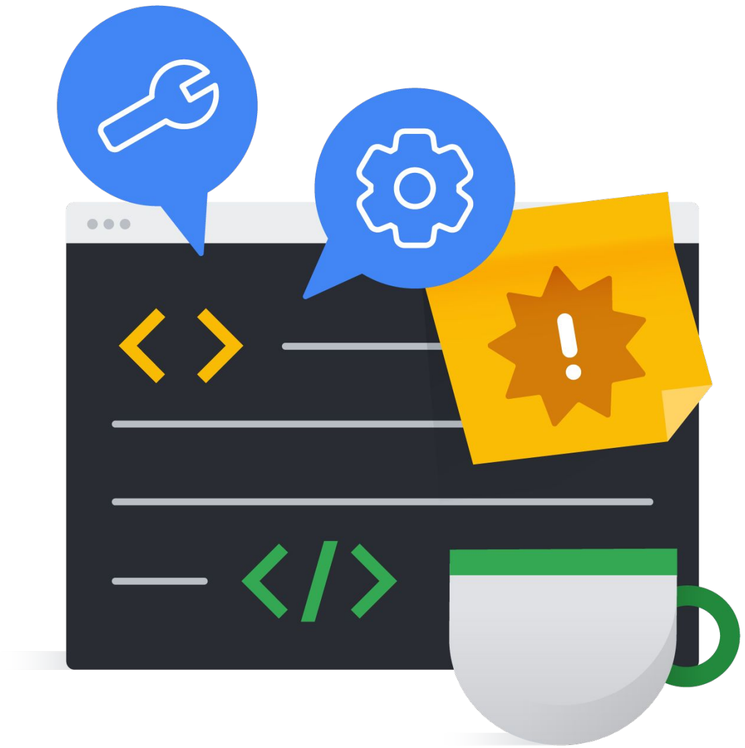


In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

- Modifying pipeline graph without a transform mapping
- Adding/removing side inputs

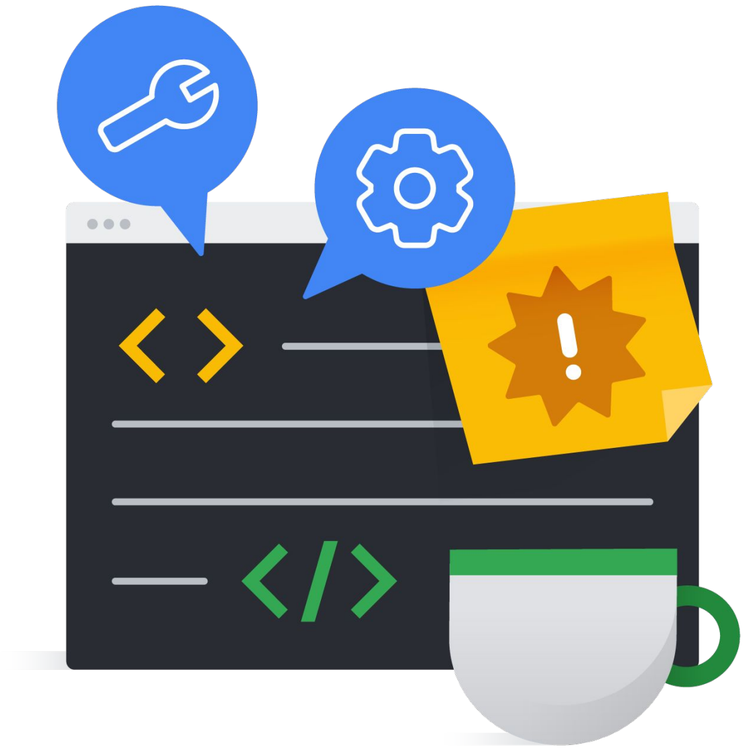


In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

- Modifying pipeline graph without a transform mapping
- Adding/removing side inputs
- Changing coders

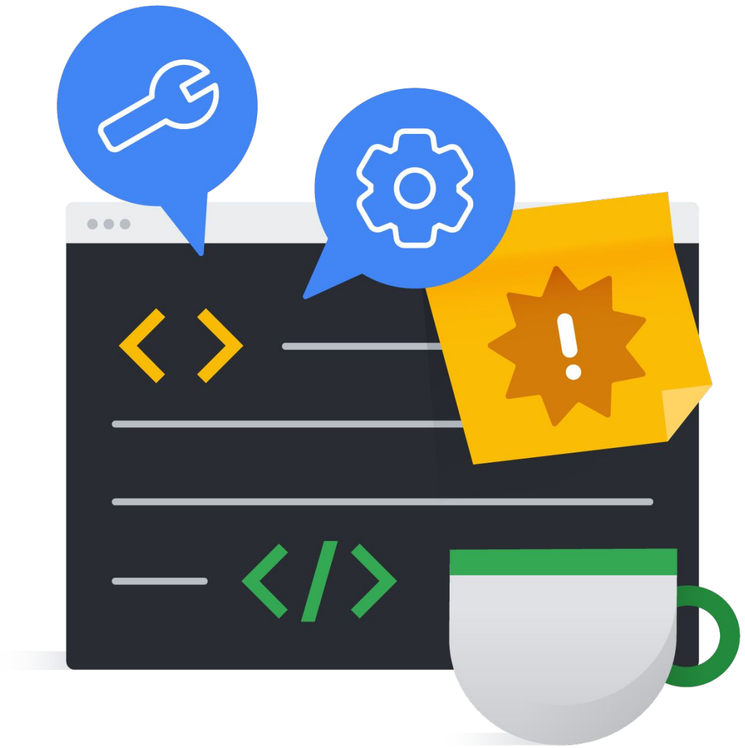


In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

- Modifying pipeline graph without a transform mapping
- Adding/removing side inputs
- Changing coders
- Switching locations

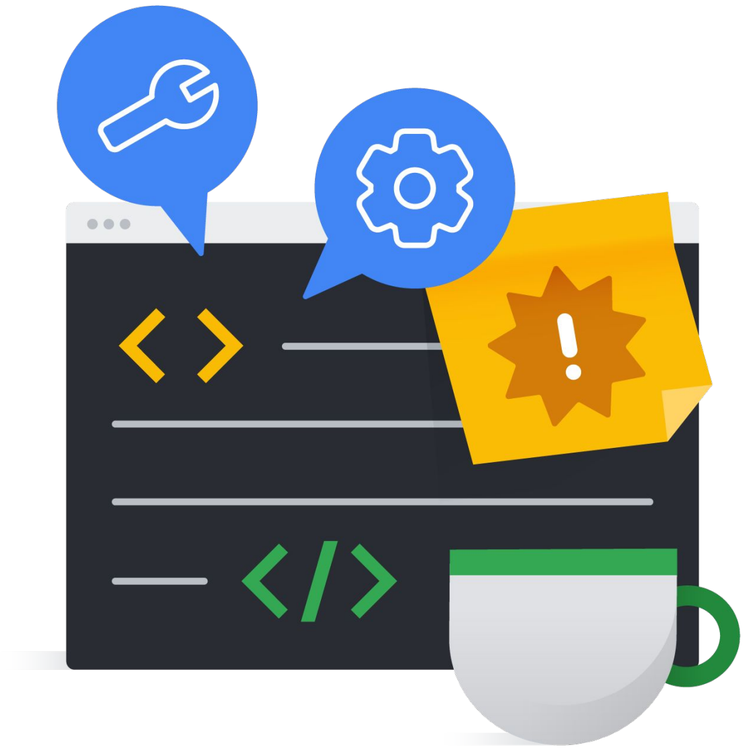


In-flight actions

Preventing compatibility breaks

Common **compatibility check** failures:

- Modifying pipeline graph without a transform mapping
- Adding/removing side inputs
- Changing coders
- Switching locations
- Removing stateful operations

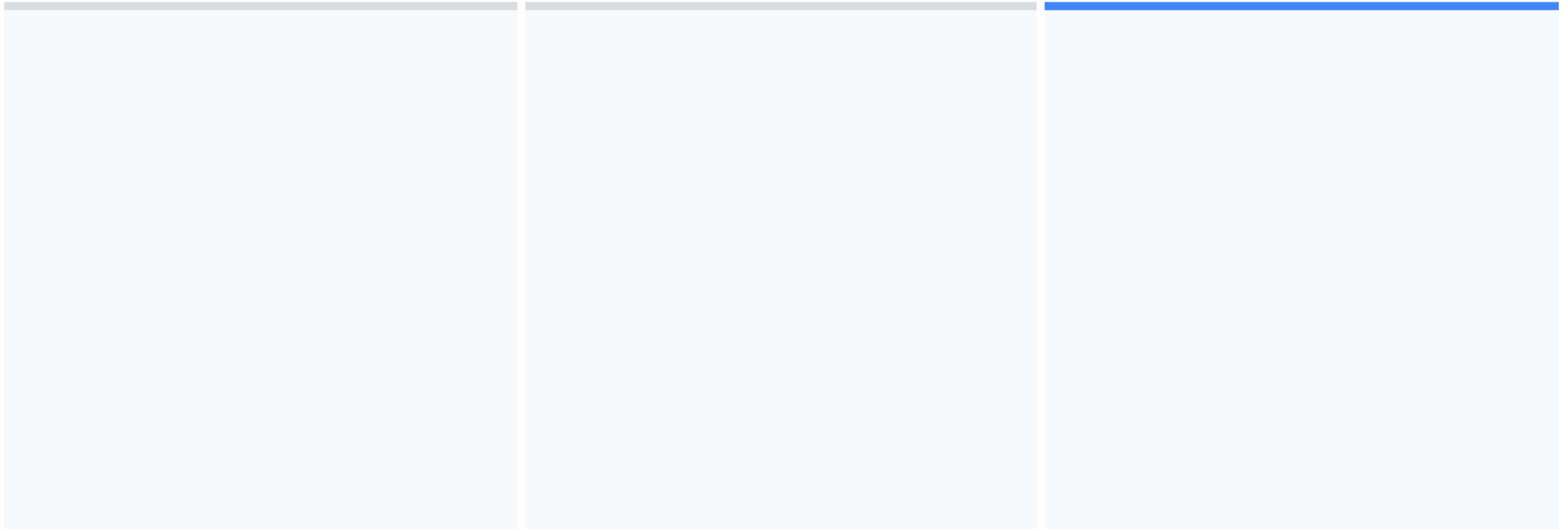


Pipeline termination

1 Deployment

2 In-flight

3 Termination



Pipeline termination

1 Deployment

2 In-flight

3 Termination

Drain

Stop data ingestion immediately and continue data compute until all buffered data has been processed.

Pipeline type

Streaming

Pipeline termination

1 Deployment

2 In-flight

3 Termination

Drain

Stop data ingestion immediately and continue data compute until all buffered data has been processed.

Pipeline type

Streaming

Cancel

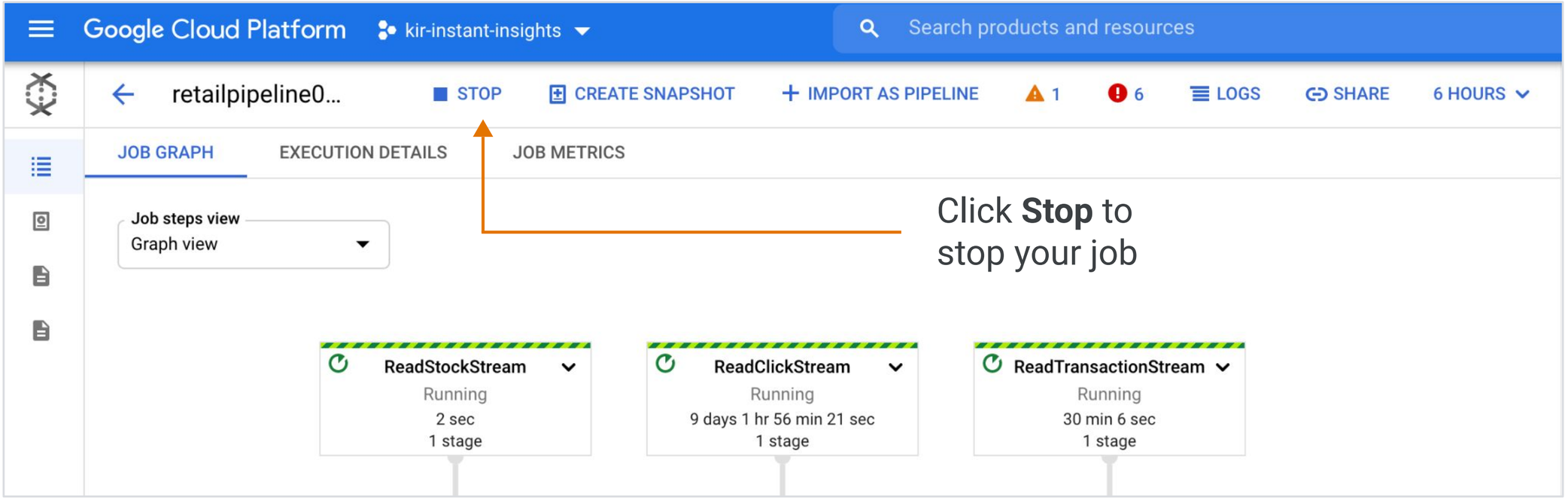
Stop data ingestion and data compute immediately.

Pipeline type

Batch & Streaming

Pipeline termination

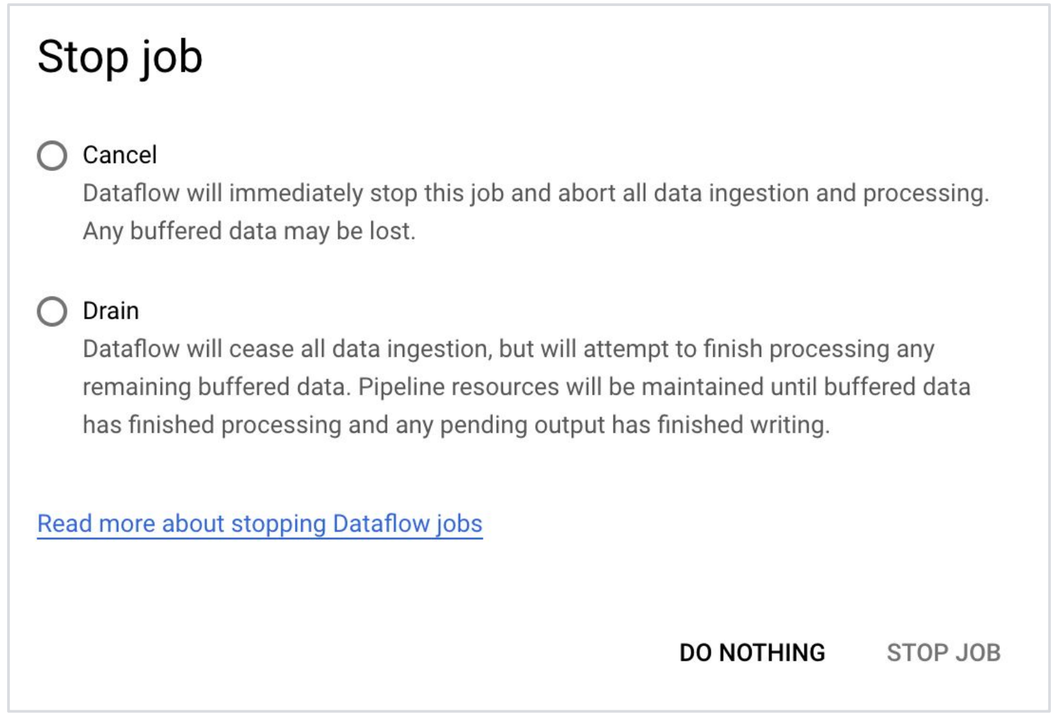
Stopping your job



*Jobs can also be terminated via the CLI

Pipeline termination

Stopping your job



*Jobs can also be terminated via the CLI

Pipeline termination

Drain vs. cancel

Drain

Stops pulling source data, finishes processing buffered data

- No data is lost

Cancel

Pipeline termination

Drain vs. cancel

Drain

Stops pulling source data, finishes processing buffered data

- No data is lost
- All windows are closed, resulting in incomplete aggregations

Cancel

Pipeline termination

Drain vs. cancel

Drain

Stops pulling source data, finishes processing buffered data

- No data is lost
- All windows are closed, resulting in incomplete aggregations

Pro-tip: Use Beam PaneInfo object to identify & filter incomplete windows

Cancel

Pipeline termination

Drain vs. cancel

Drain

Stops pulling source data, finishes processing buffered data

- ✓ No data is lost
- ✗ All windows are closed, resulting in incomplete aggregations

Pro-tip: Use Beam PaneInfo object to identify & filter incomplete windows

Cancel

Stops pulling data & terminates data processing

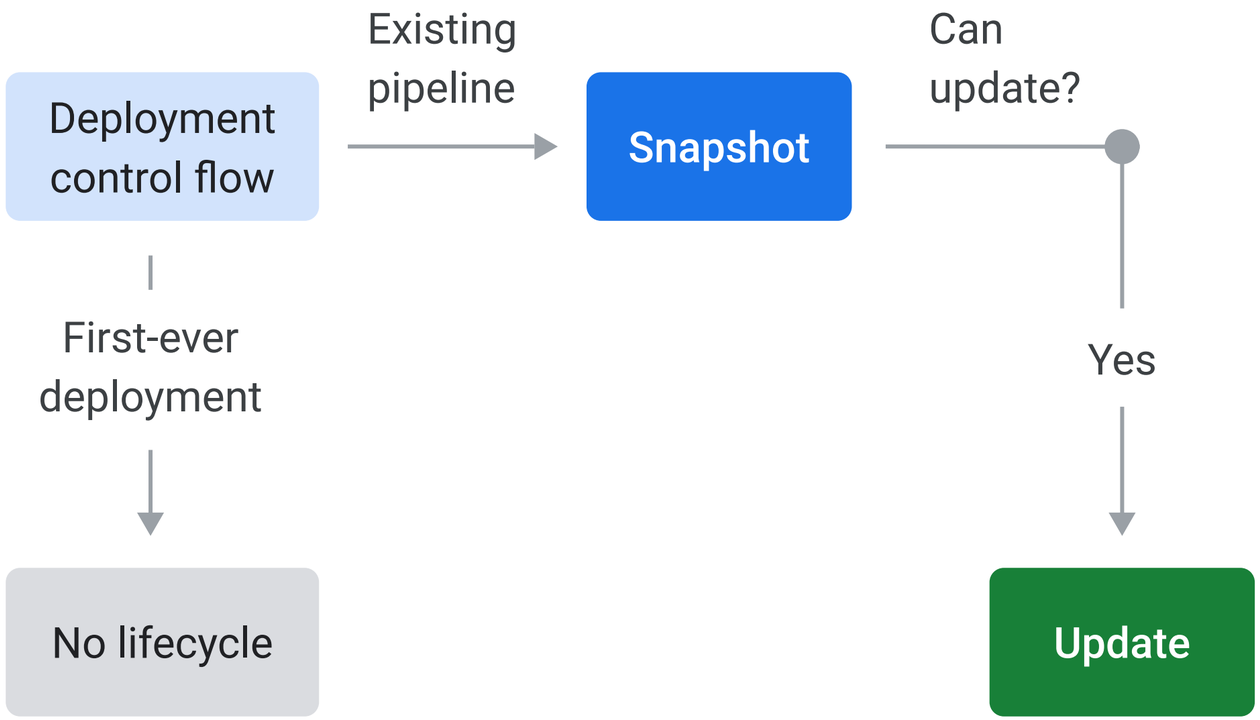
- ✓ Easy for non–mission critical workloads
- ✗ Data is lost (unless backed up by the source)

Deployment flowchart

Decision tree

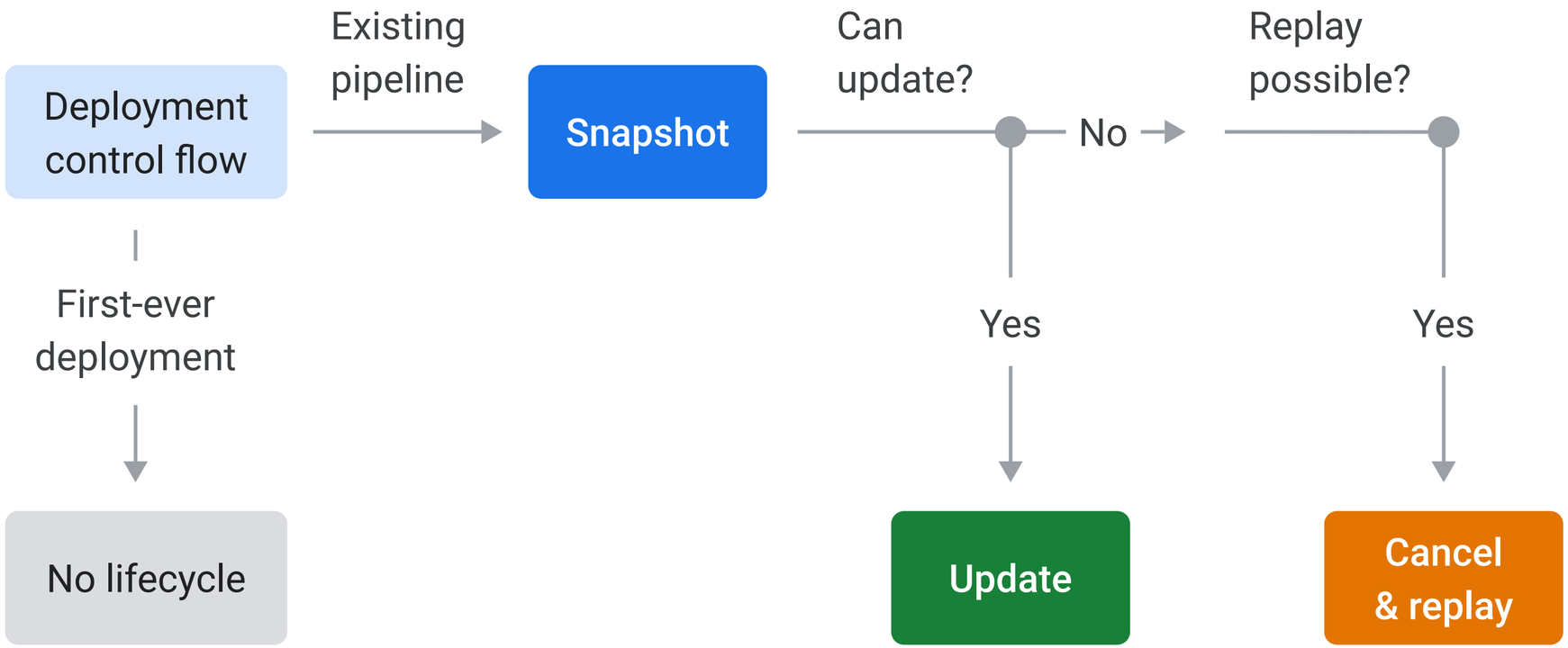
Deployment flowchart

Decision tree



Deployment flowchart

Decision tree



Deployment flowchart

Decision tree