Reliability

Vince Gonzalez

Strategic Cloud Engineer,
Google Cloud

Agenda

Course Intro

Monitoring

Logging and Error Reporting

Troubleshooting and Debugging

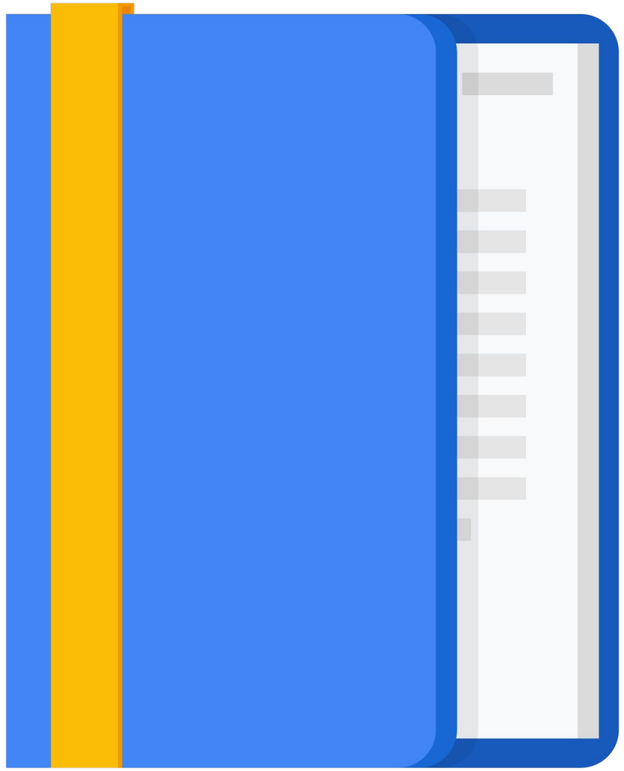
Performance

Testing and CI/CD

Reliability

Flex Templates

Course Summary



Introduction to reliability

Batch

- Rerun pipeline if a job fails
- Source data is not lost, and partial data written to sinks can be rewritten

Introduction to reliability

Batch

- Rerun pipeline if a job fails
- Source data is not lost, and partial data written to sinks can be rewritten

Streaming

- Protect against various failure modes
- Minimize / eliminate data loss
- Minimize downtime

Reliability failure modes

User code and data shape

Outages

Reliability failure modes

User code and data shape

- Transient errors
- Corrupted data

Outages

Reliability failure modes

User code and data shape

- Transient errors
- Corrupted data

Outages

- Service outage
- Zonal outage
- Regional outage

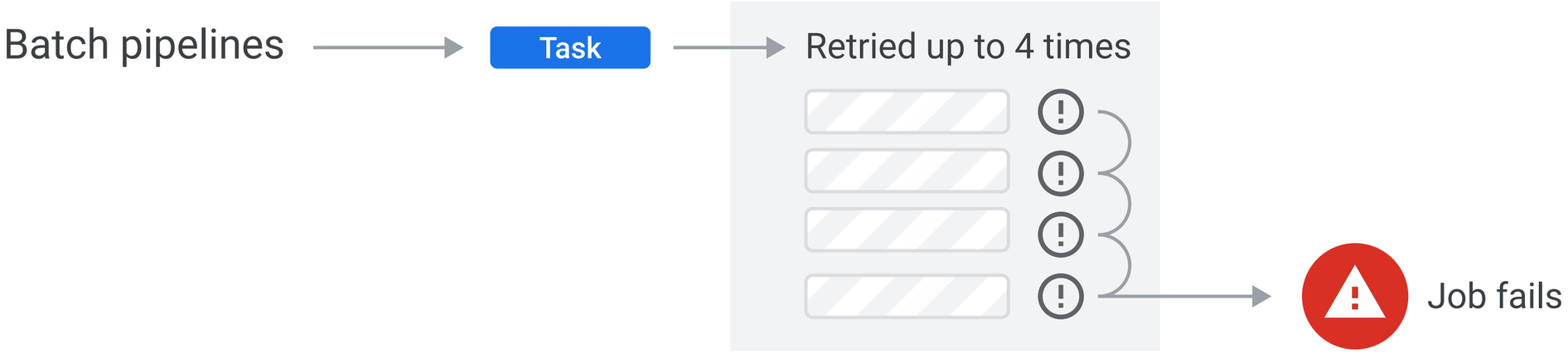
Reliability

Agenda

- Monitoring
- Geolocation
- Disaster recovery
- High availability

Failure during pipeline execution

Batch vs streaming



Failure during pipeline execution

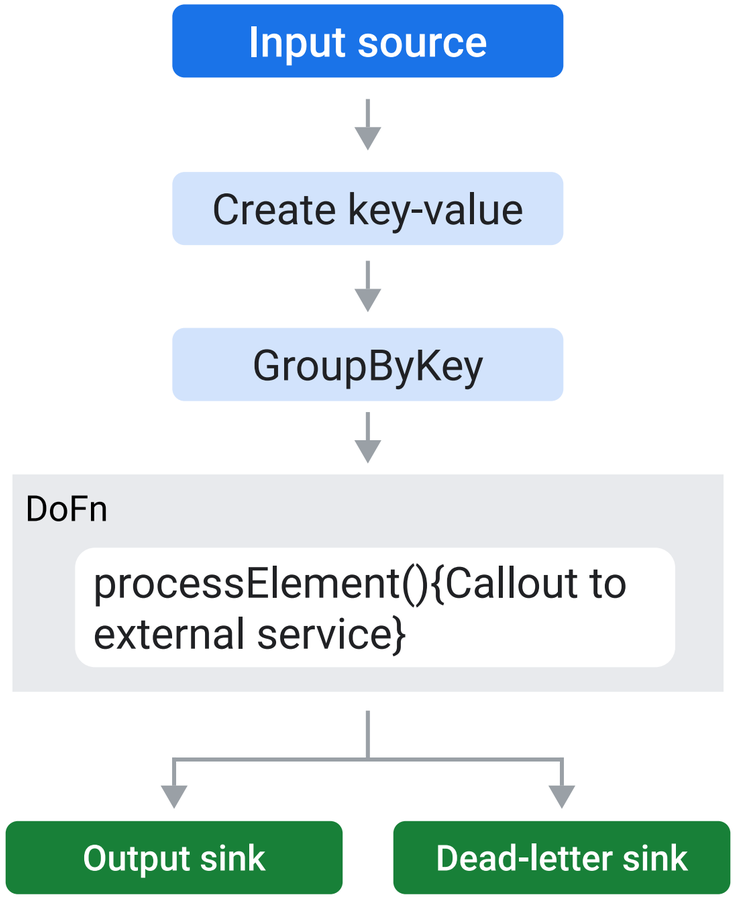
Batch vs streaming

Batch pipelines → Task → Retried up to 4 times → Job fails

Streaming pipelines → Task → Stall indefinitely →

- High data freshness
- Job logs
- Cloud Monitoring
- Metrics for pipeline progress and error count

Dead-letter sinks and error logging



Dead-letter sinks and error logging

Java

```
final TupleTag successTag;
final TupleTag deadLetterTag;
PCollection input = /* ... */;

PCollectionTuple outputTuple = input.apply(ParDo.of(new DoFn(){
    @Override
    void processElement(ProcessContext ctxt) {
        try {
            c.output(process(c.element));
        } catch(MyException ex) {
            // Optional Logging at debug level
            c.sideOutPut(deadLetterTag, c.element);
        }
    }
})).writeOutPutTags(successTag, TupleTagList.of(deadLetterTag));

// Write dead letter elements to separate sink
outputTuple.get(deadLetterTag).apply(BigQuery.write(...));

// Process the successful element differently.
PCollection success = outputTuple.get(successTag);
```

Dead-letter sinks and error logging

Java

```
final TupleTag successTag;
final TupleTag deadLetterTag;
PCollection input = /* ... */;

PCollectionTuple outputTuple = input.apply(ParDo.of(new DoFn(){
    @Override
    void processElement(ProcessContext ctxt) {
        try {
            c.output(process(c.element));
        } catch(MyException ex) {
            // Optional Logging at debug level
            c.sideOutPut(deadLetterTag, c.element);
        }
    }
})).writeOutPutTags(successTag, TupleTagList.of(deadLetterTag));

// Write dead letter elements to separate sink
outputTuple.get(deadLetterTag).apply(BigQuery.write(...));

// Process the successful element differently.
PCollection success = outputTuple.get(successTag);
```

Dead-letter sinks and error logging

Java

```
final TupleTag successTag;
final TupleTag deadLetterTag;
PCollection input = /* ... */;

PCollectionTuple outputTuple = input.apply(ParDo.of(new DoFn(){
    @Override
    void processElement(ProcessContext ctxt) {
        try {
            c.output(process(c.element));
        } catch(MyException ex) {
            // Optional Logging at debug level
            c.sideOutPut(deadLetterTag, c.element);
        }
    }
})).writeOutPutTags(successTag, TupleTagList.of(deadLetterTag));

// Write dead letter elements to separate sink
outputTuple.get(deadLetterTag).apply(BigQuery.write(...));

// Process the successful element differently.
PCollection success = outputTuple.get(successTag);
```

Monitoring and alerting policies

- Catch issues before they bring down production systems

Monitoring and alerting policies

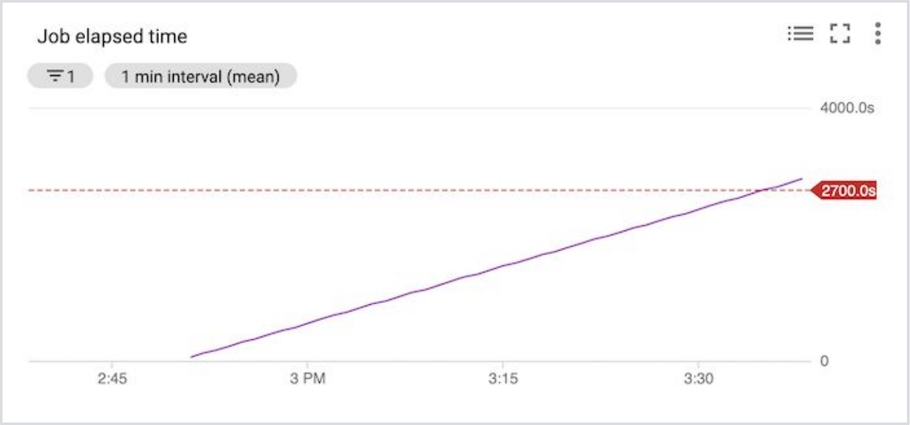
- Catch issues before they bring down production systems
- Dataflow's Job Metrics tab provides an integrated monitoring experience

Monitoring and alerting policies

- Catch issues before they bring down production systems
- Dataflow's Job Metrics tab provides an integrated monitoring experience
- Cloud Monitoring integration extends capabilities

Monitoring and alerting policies

Batch

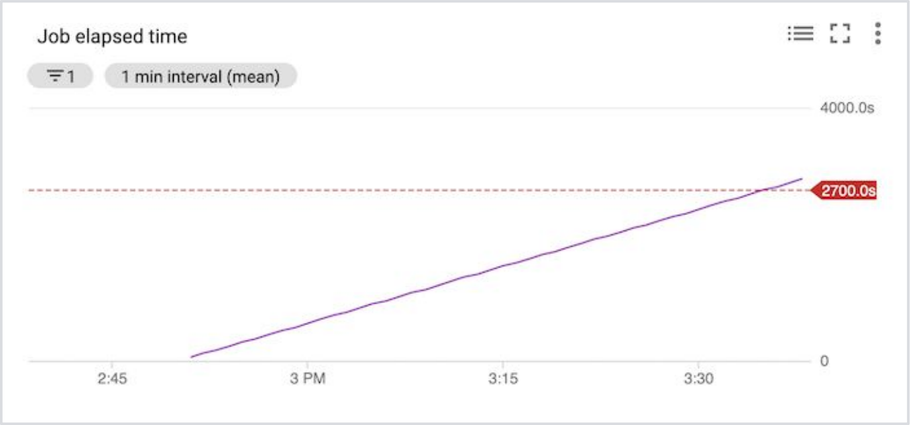


Potential metrics to alert on:

- Job status
- Elapsed time

Monitoring and alerting policies

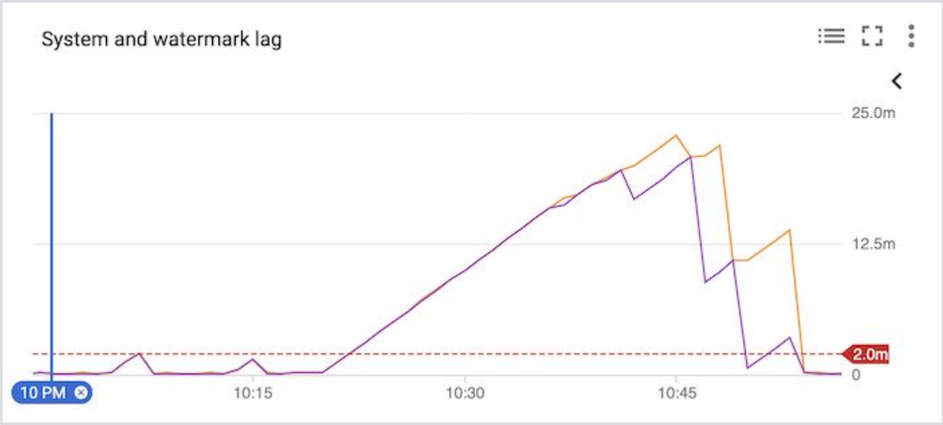
Batch



Potential metrics to alert on:

- Job status
- Elapsed time

Streaming



Potential metrics to alert on:

- Data freshness
- System latency

Reliability

Agenda

- Monitoring
- Geolocation
- Disaster recovery
- High availability

Worker region

DO: Specify **region**

```
$ python3 -m apache_beam.examples.wordcount
\
  --input
gs://dataflow-samples/shakespeare/kinglear.t
xt \
  --output gs://$BUCKET/results/outputs
--runner DataflowRunner  \
  --project $PROJECT  --temp_location
gs://$BUCKET/tmp/  \
  --region $REGION
```

Worker region

DO: Specify **region**

```
$ python3 -m apache_beam.examples.wordcount
\
  --input
gs://dataflow-samples/shakespeare/kinglear.t
xt \
  --output gs://$BUCKET/results/outputs
--runner DataflowRunner  \
  --project $PROJECT  --temp_location
gs://$BUCKET/tmp/  \
  --region $REGION
```

DON'T: Specify **region & worker_zone**

```
$ python3 -m apache_beam.examples.wordcount
\
  --input
gs://dataflow-samples/shakespeare/kinglear.t
xt \
  --output gs://$BUCKET/results/outputs
--runner DataflowRunner  \
  --project $PROJECT  --temp_location
gs://$BUCKET/tmp/  \
  --region $REGION  --worker_zone
```

Worker region

DO: Specify **region**

```
$ python3 -m apache_beam.examples.wordcount
\
  --input
gs://dataflow-samples/shakespeare/kinglear.t
xt \
  --output gs://$BUCKET/results/outputs
--runner DataflowRunner  \
  --project $PROJECT  --temp_location
gs://$BUCKET/tmp/  \
  --region $REGION
```

DON'T: Specify **region & worker_zone**

```
$ python3 -m apache_beam.examples.wordcount
\
  --input
gs://dataflow-samples/shakespeare/kinglear.t
xt \
  --output gs://$BUCKET/results/outputs
--runner DataflowRunner  \
  --project $PROJECT  --temp_location
gs://$BUCKET/tmp/  \
  --region $REGION  --worker_zone
```

Warning: You cannot change the location of a job after it has started.

Follow isolation principles

DO: Isolate processing in one region

Storage
us-central1

↓

Dataflow
us-central1

↓

BigQuery
us-central1

Follow isolation principles

DO: Isolate processing in one region

Storage
us-central1

↓

Dataflow
us-central1

↓

BigQuery
us-central1

DO: Use multi-regional locations for sources and sinks

Storage
us

↓

Dataflow
us-central1

↓

BigQuery
us

Follow isolation principles

DO: Isolate processing in one region

Storage
us-central1

↓

Dataflow
us-central1

↓

BigQuery
us-central1

DO: Use multi-regional locations for sources and sinks

Storage
us

↓

Dataflow
us-central1

↓

BigQuery
us

DON'T: Deploy cross-region dependencies

Storage
us-central1

↓

Dataflow
us-central1

↓

BigQuery
us-east4

Reliability

Agenda

- Monitoring
- Geolocation
- Disaster recovery
- High availability

Disaster recovery on Dataflow

Disaster recovery (DR)

Disaster recovery (DR) protects you from losing your most precious asset: data.

Disaster recovery on Dataflow

Disaster recovery (DR)

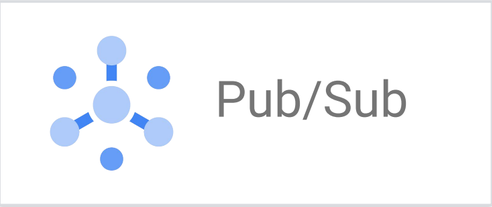
Disaster recovery (DR) protects you from losing your most precious asset: data.

Source snapshots

Source snapshots are one way of preempting data loss in the case of pipeline failure.

Pub/Sub Snapshots

Google Cloud Pub/Sub supports this capability



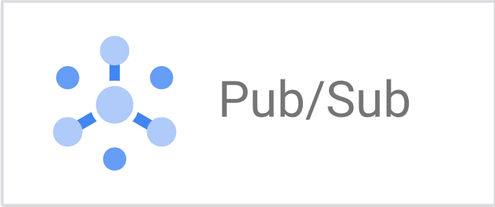
Snapshots and Seek

Pub/Sub Snapshots

Google Cloud Pub/Sub supports this capability.

Snapshots

Save your subscription's acknowledgement state



Snapshots and Seek

Pub/Sub Snapshots

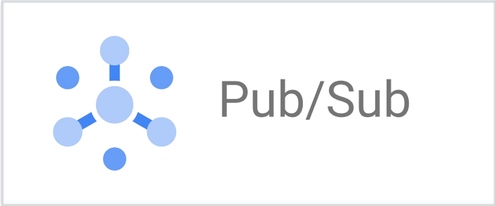
Google Cloud Pub/Sub supports this capability.

Snapshots

Save your subscription's acknowledgement state

Seek

Revert messages to a prior acknowledgement state



Snapshots and Seek

Using Pub/Sub Snapshots for disaster recovery

1. Make a snapshot of a subscription*

```
gcloud pubsub snapshots create
my-snapshot --subscription=my-sub
```

*These steps also can be accomplished in the Console UI

Using Pub/Sub Snapshots for disaster recovery

1. Make a snapshot of a subscription*

```
gcloud pubsub snapshots create
my-snapshot --subscription=my-sub
```

2. Stop and drain your Dataflow job*

```
gcloud dataflow jobs drain [job-id]
```

*These steps also can be accomplished in the Console UI

Using Pub/Sub Snapshots for disaster recovery

1. Make a snapshot of a subscription*

```
gcloud pubsub snapshots create
my-snapshot --subscription=my-sub
```

2. Stop and drain your Dataflow job*

```
gcloud dataflow jobs drain [job-id]
```

3. Seek your subscription to the snapshot*

```
gcloud pubsub subscriptions seek
my-sub --snapshot=my-snapshot
```

*These steps also can be accomplished in the Console UI

Using Pub/Sub Snapshots for disaster recovery

1. Make a snapshot of a subscription*

```
gcloud pubsub snapshots create
my-snapshot --subscription=my-sub
```

2. Stop and drain your Dataflow job*

```
gcloud dataflow jobs drain [job-id]
```

3. Seek your subscription to the snapshot*

```
gcloud pubsub subscriptions seek
my-sub --snapshot=my-snapshot
```

4. Resubmit the pipeline*

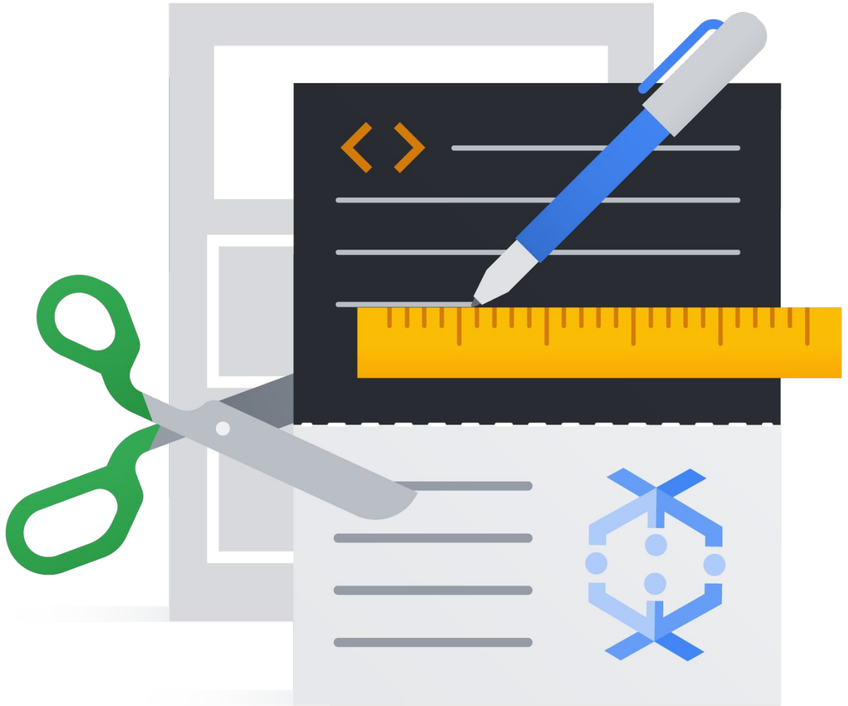
```
gcloud dataflow jobs run my-job-name
--gcs_location =my_gcs_bucket
```

*These steps also can be accomplished in the Console UI

Using Pub/Sub Snapshots for disaster recovery

Best practices

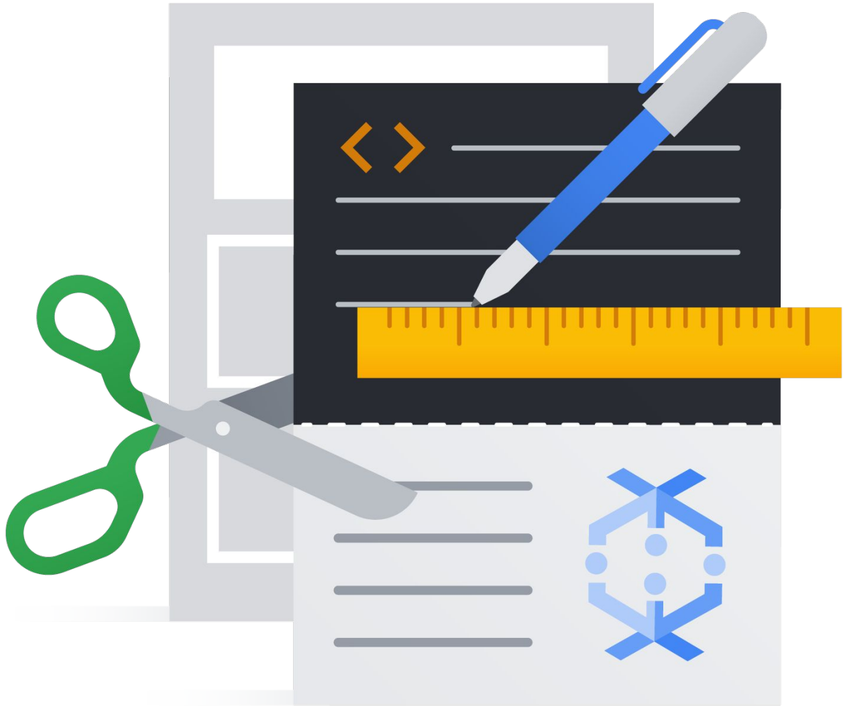
Pub/Sub messages have a maximum data retention of 7 days



Using Pub/Sub Snapshots for disaster recovery

Best practices

Pub/Sub messages have a maximum data retention of 7 days—so take a Pub/Sub Snapshot at least weekly to ensure no data loss



Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing
- Disrupts exactly-once processing

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing
- Disrupts exactly-once processing

Dataflow

Snapshots

Dataflow Snapshots saves the state of streaming pipelines:

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing
- Disrupts exactly-once processing

Dataflow

Snapshots

Dataflow Snapshots saves the state of streaming pipelines:

- Restart a pipeline without reprocessing in-flight data

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing
- Disrupts exactly-once processing

Dataflow

Snapshots

Dataflow Snapshots saves the state of streaming pipelines:

- Restart a pipeline without reprocessing in-flight data
- No data loss with minimal downtime

Dataflow Snapshots

Pub/Sub

Snapshots and Seek

Caveats for using Pub/Sub Snapshots and Seek:

- Reconciliation of data written to sink
- Message reprocessing
- Disrupts exactly-once processing

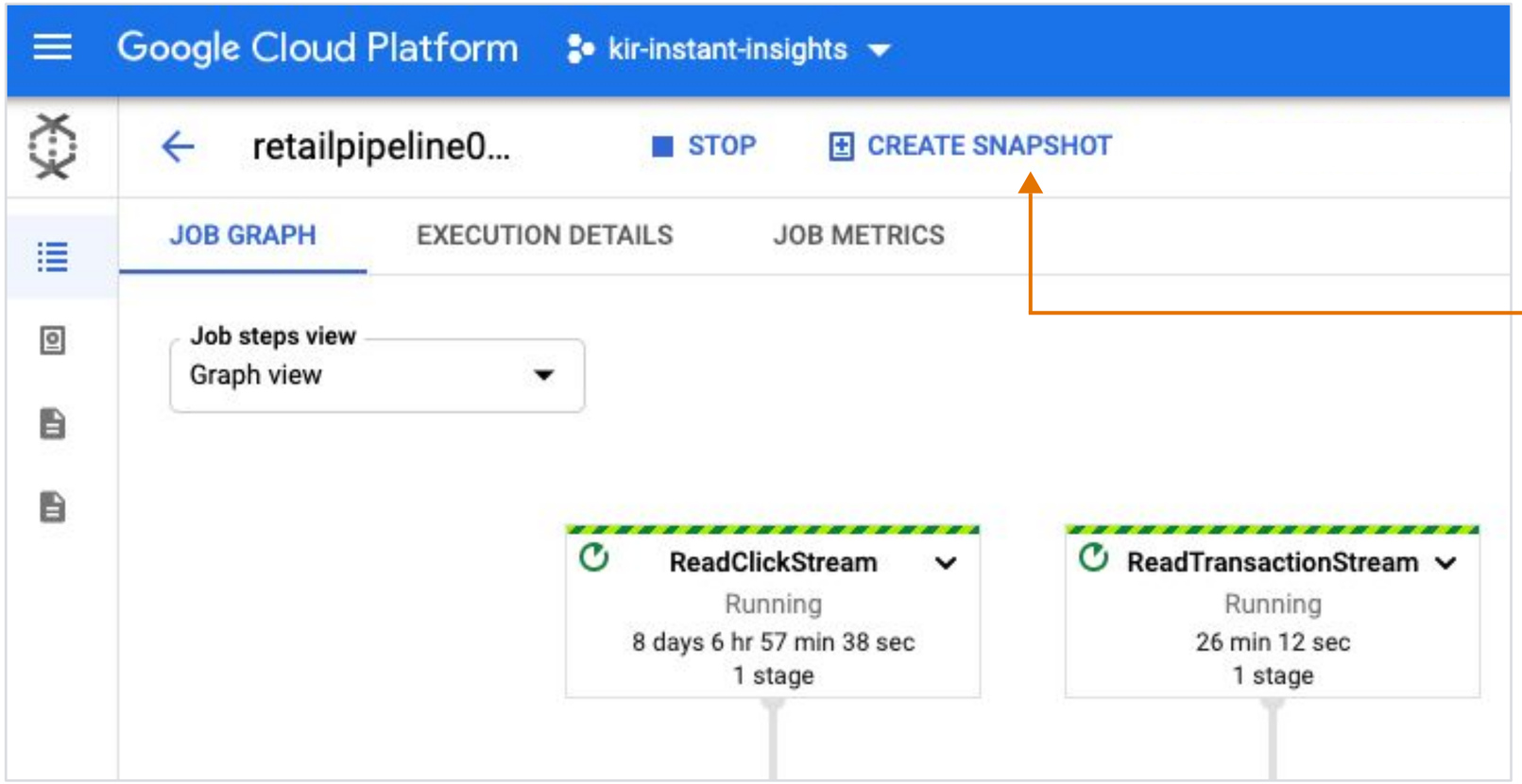
Dataflow

Snapshots

Dataflow Snapshots saves the state of streaming pipelines:

- Restart a pipeline without reprocessing in-flight data
- No data loss with minimal downtime
- Option to create a snapshot with Pub/Sub source

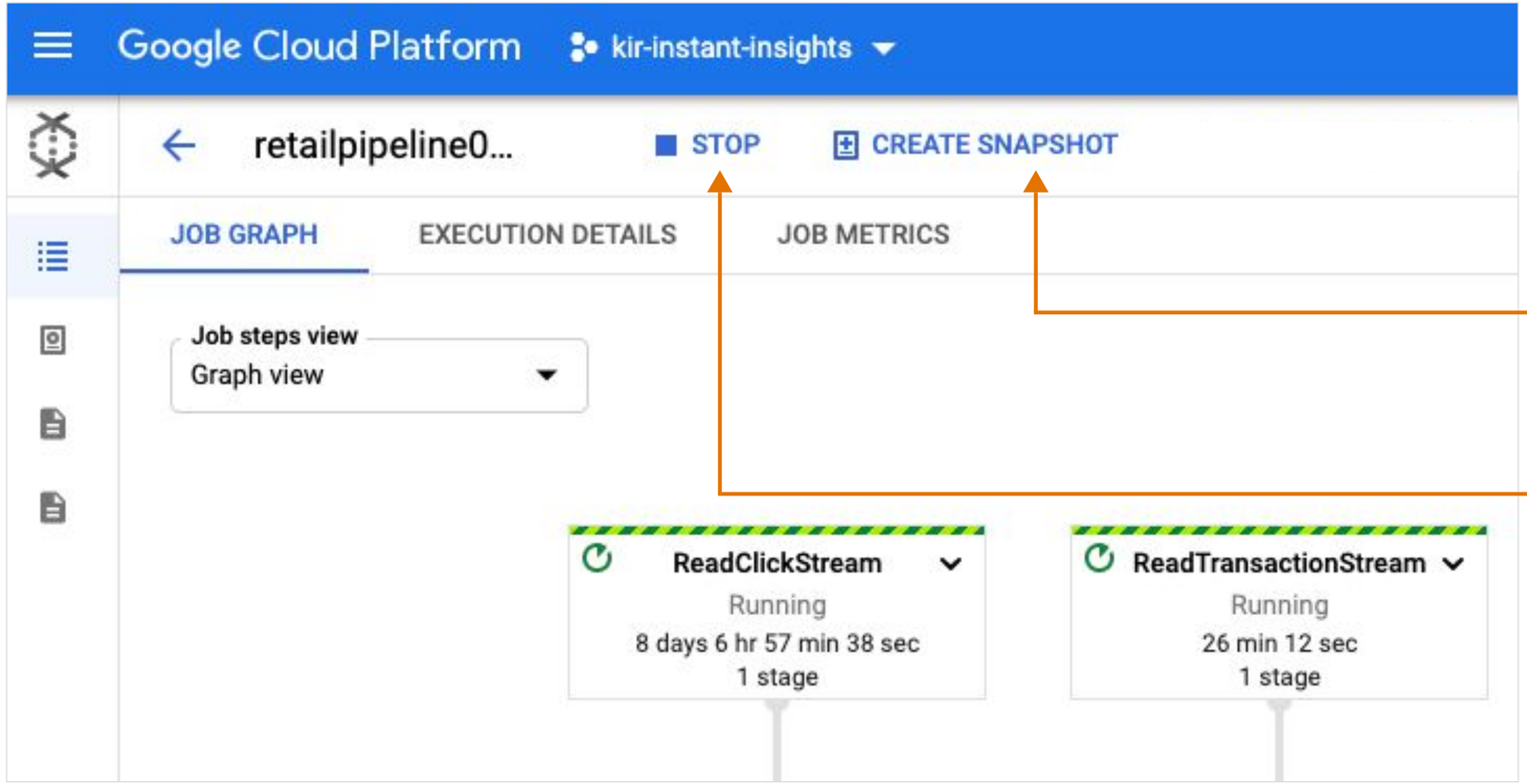
Using Dataflow Snapshots for disaster recovery



1 Take Dataflow snapshots with Pub/Sub sources*

*These steps also can be accomplished with the command line interface

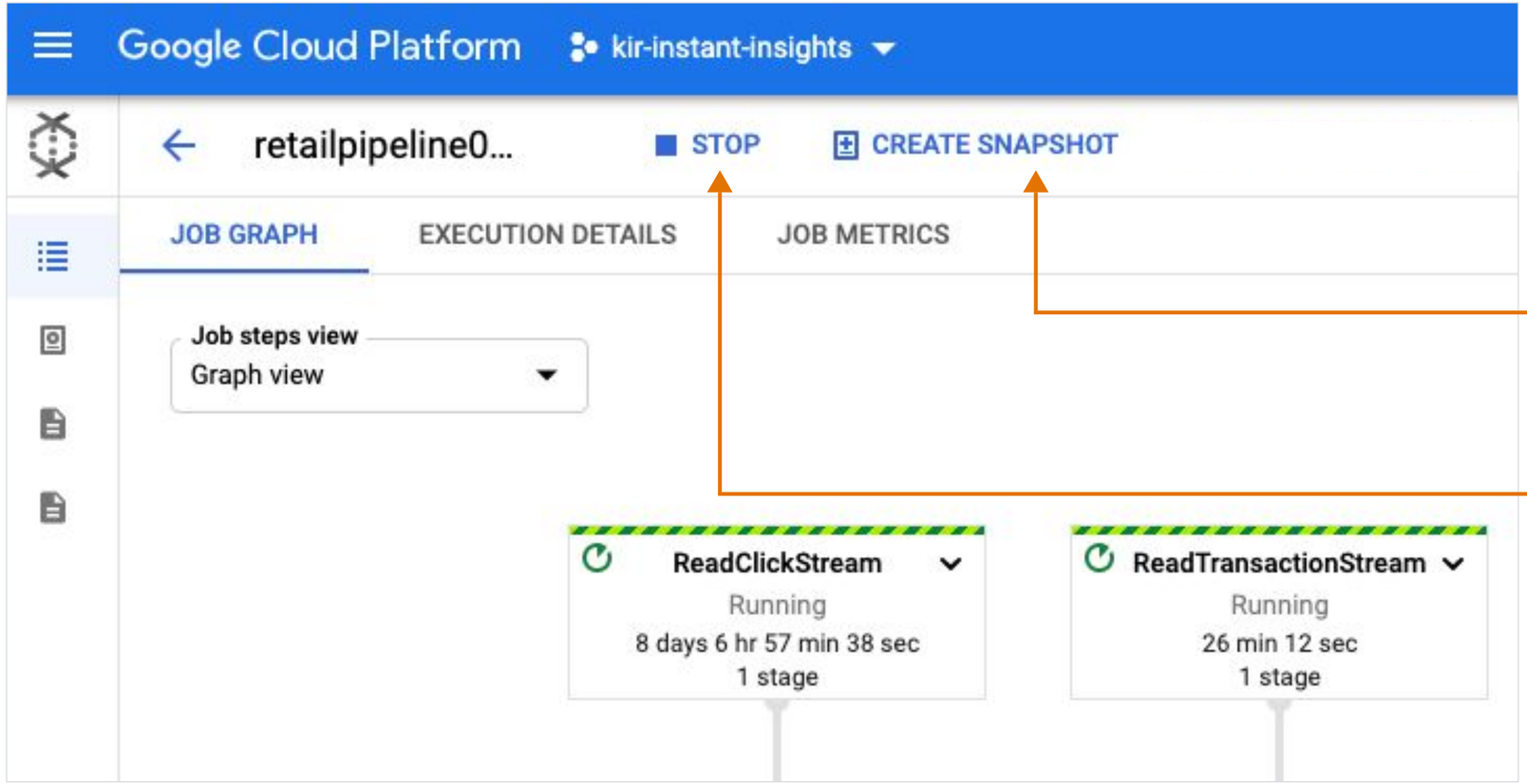
Using Dataflow Snapshots for disaster recovery



1. Take Dataflow snapshots with Pub/Sub sources*
2. Stop and drain your Dataflow job*

*These steps also can be accomplished with the command line interface

Using Dataflow Snapshots for disaster recovery



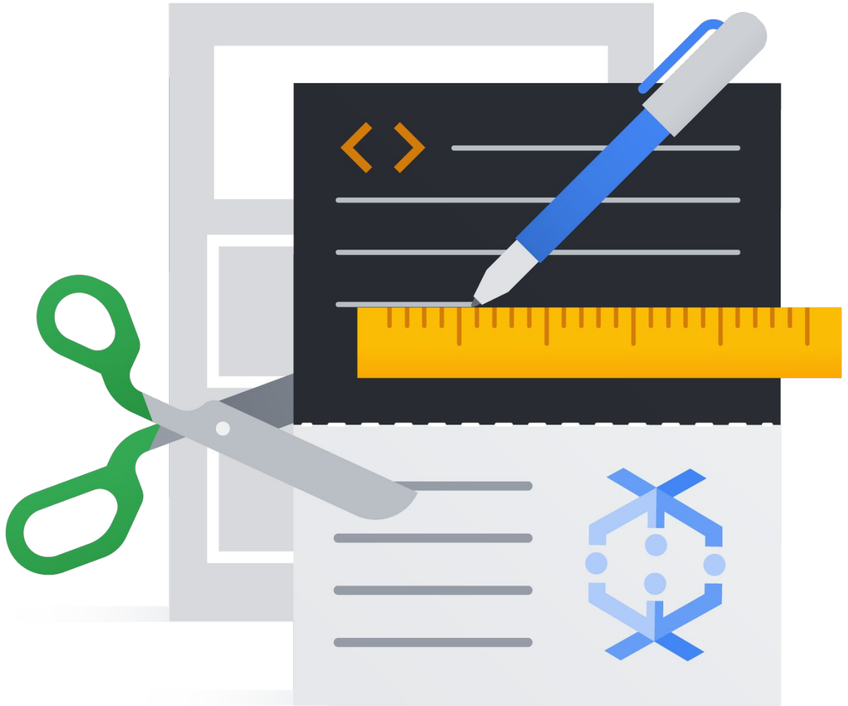
1. Take Dataflow snapshots with Pub/Sub sources*
2. Stop and drain your Dataflow job*
3. Create job with snapshot from your IDE

*These steps also can be accomplished with the command line interface

Using Dataflow Snapshots for disaster recovery

Best practices

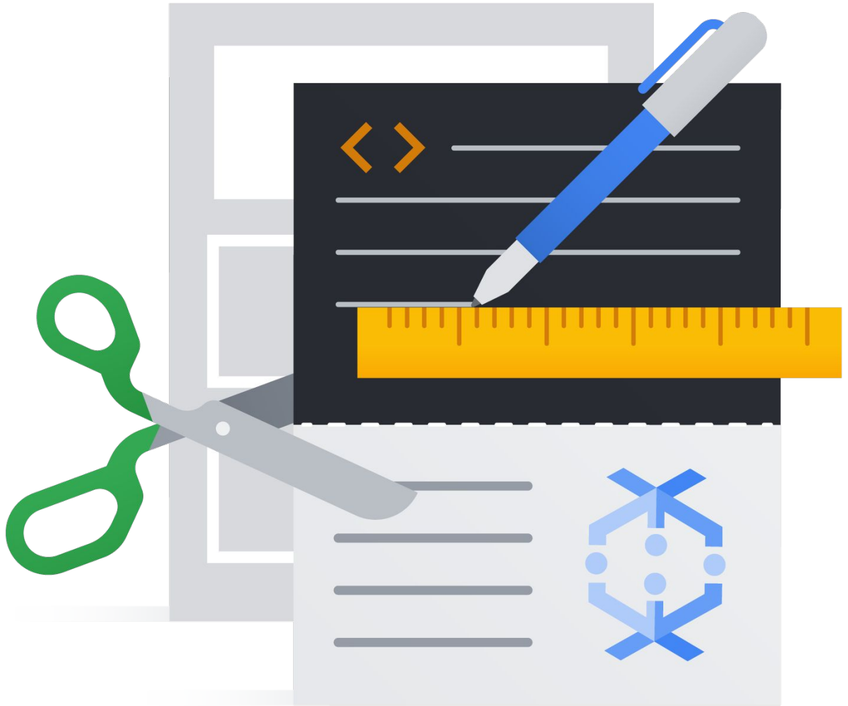
- Schedule a Dataflow snapshot at least once a week using **Cloud Composer** or **Cloud Scheduler** to ensure you never lose data in the event of an outage.



Using Dataflow Snapshots for disaster recovery

Best practices

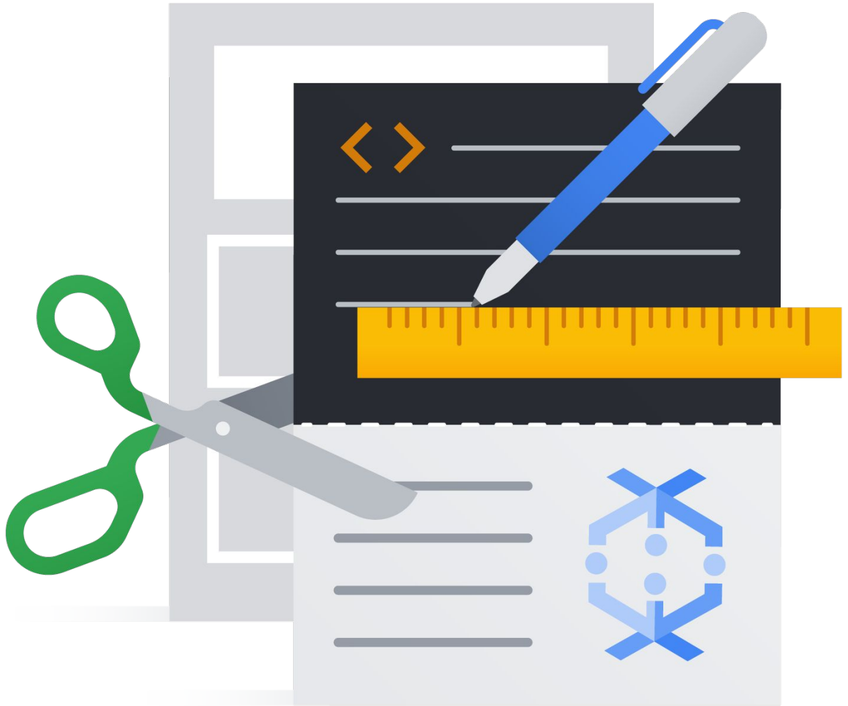
- Schedule a Dataflow snapshot at least once a week using **Cloud Composer** or **Cloud Scheduler** to ensure you never lose data in the event of an outage.
- Snapshots are stored in the region of their job. When you create a job from a snapshot, it must be in the **same region.**



Using Dataflow Snapshots for disaster recovery

Best practices

- Schedule a Dataflow snapshot at least once a week using **Cloud Composer** or **Cloud Scheduler** to ensure you never lose data in the event of an outage.
- Snapshots are stored in the region of their job. When you create a job from a snapshot, it must be in the **same region.**
- In case of regional outage, wait for the region to come back online.



Reliability

Agenda

- Monitoring
- Geolocation
- Disaster recovery
- High availability

High availability on Dataflow

High availability (HA) requirements come from your business needs.

High availability on Dataflow

High availability (HA) requirements come from your business needs.

Three considerations when determining your HA needs:

High availability on Dataflow

High availability (HA) requirements come from your business needs.

Three considerations when determining your HA needs:

- Downtime

High availability on Dataflow

High availability (HA) requirements come from your business needs.

Three considerations when determining your HA needs:

- Downtime
- Data loss

High Availability on Dataflow

High availability (HA) requirements come from your business needs.

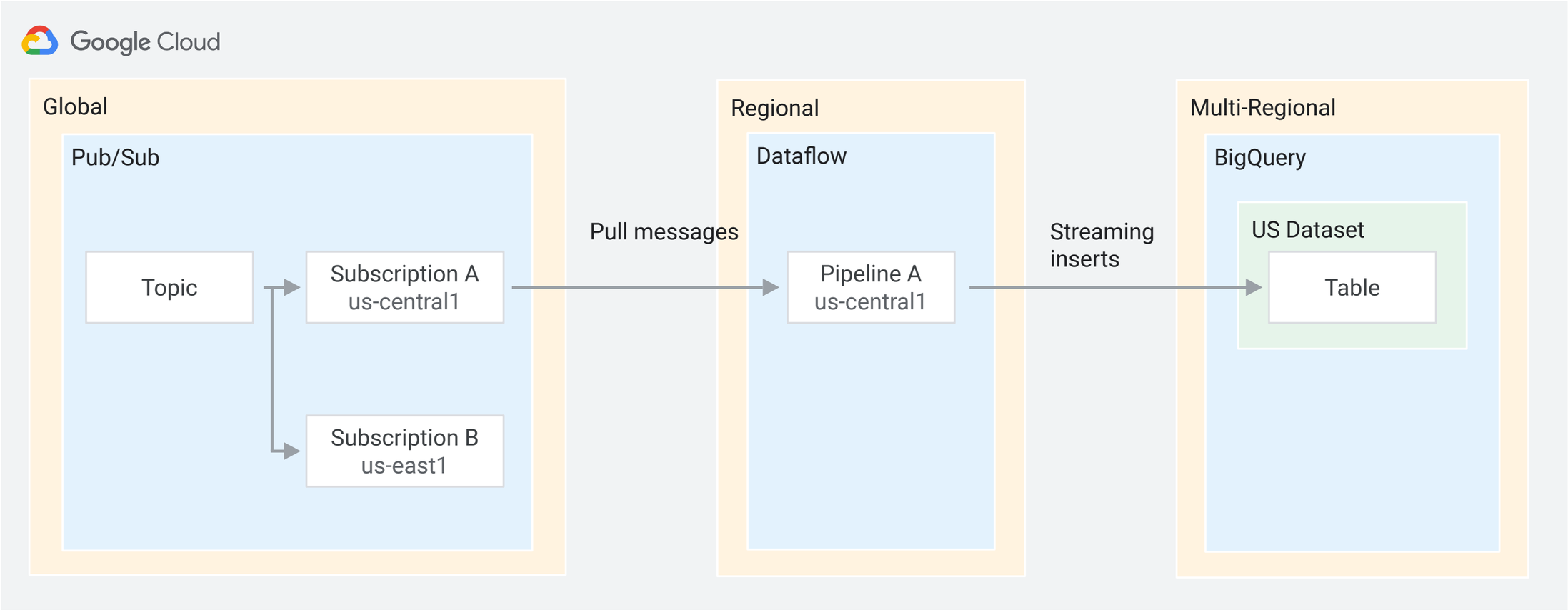
Three considerations when determining your HA needs:

- Downtime
- Data loss
- Cost

High availability: Configuration 1

Redundant sources

✓ Downtime ✗ Data loss ✓ Cost



High availability: Configuration 2

Redundant pipelines

✓ Downtime ✓ Data loss ✗ Cost