CLOUD NATIVE WORKFLOWS

AWS BATCH

AZURE BATCH

GCP PROCESSING PIPELINES

DATA LIFECYCLE

CLOUD NATIVE WORKFLOWS

Batch workflows

- AWS batch
- Azure batch

Data processing pipelines

- GCP processing pipelines
- GCP data lifecycle

AWS BATCH

AWS BATCH
AZURE BATCH
GCP PROCESSING PIPELINES
DATA LIFECYCLE

AWS BATCH

Fully managed batch processing at any scale

enables developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS

There is no additional charge for AWS Batch

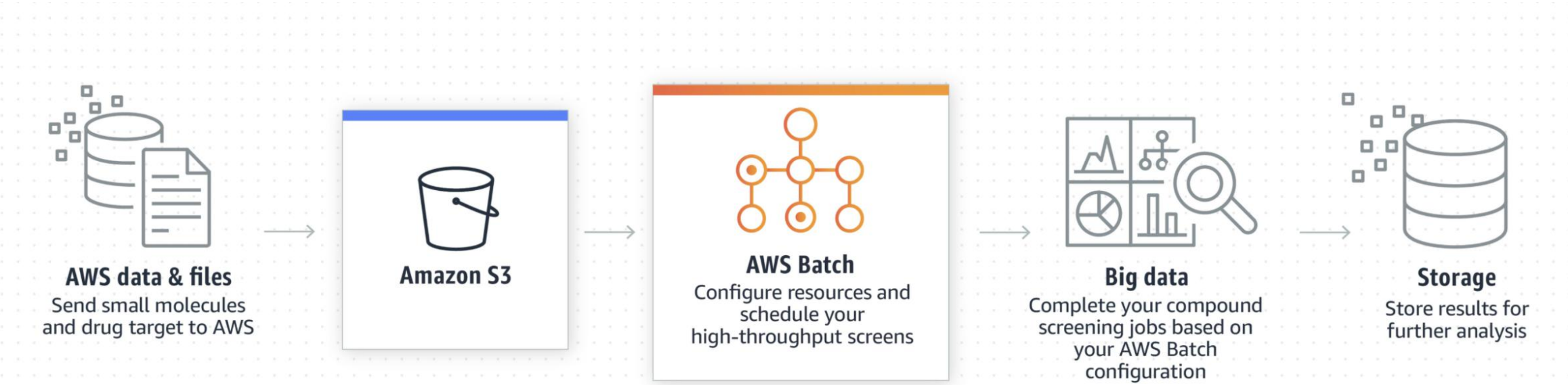
BENEFITS

Fully managed

Integrated with AWS

Cost optimized resource provisioning

LIFE SCIENCES: DRUG SCREENING FOR BIOPHARMA



AZURE BATCH

AWS BATCH
AZURE BATCH
GCP PROCESSING PIPELINES
DATA LIFECYCLE

AZURE BATCH

Cloud-scale job scheduling and compute management

- Scale to tens, hundreds, or thousands of virtual machines
- Cloud-enable batch and HPC applications
- Stage data and execute compute pipelines
- Choose Linux or Windows to run jobs
- Autoscale on work in the queue
- Pay for what you use with no capital investment

BENEFITS

Get batch computing power when you need it

Choose your operating system and tools

Cloud-enable your cluster applications

Imagine running at 100x scale

MORE BENEFITS

Tell us what to execute

- high-scale job scheduling engine that's available to you as a managed service. Use the scheduler in your application to dispatch work. Batch can also work with cluster job schedulers or behind the scenes of your software as a service (SaaS). You don't need to write your own work queue, dispatcher, or monitor. Batch gives you this as a service.

Let Batch take care of scale for you

- Batch starts a pool of compute virtual machines

Deliver solutions as a service

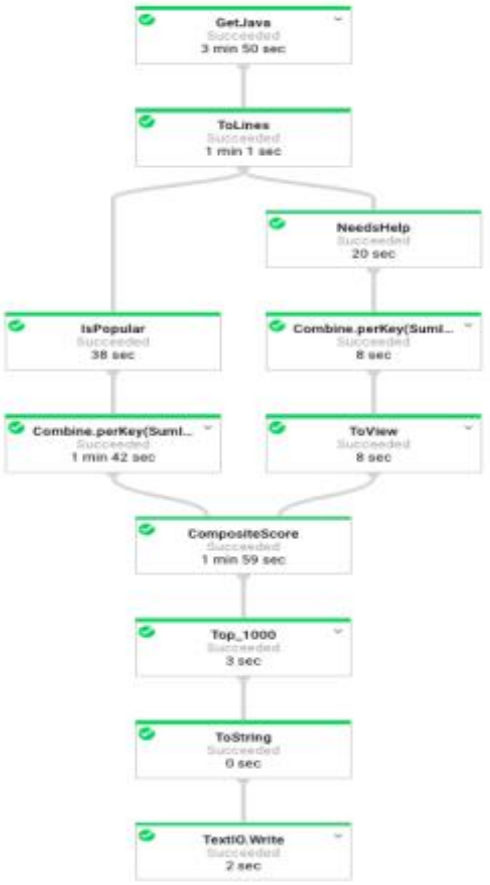
- Batch processes jobs on demand, not on a predefined schedule

GCP PROCESSING PIPELINES

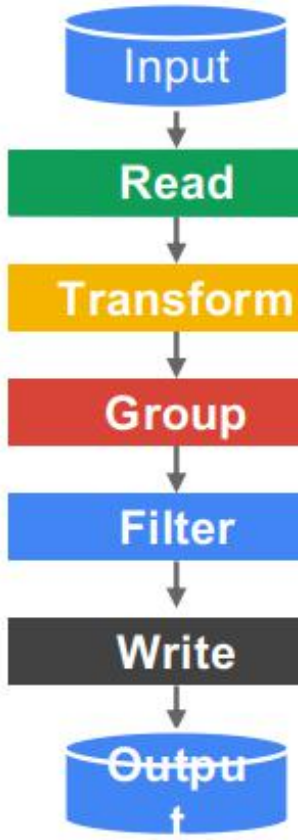
AWS BATCH
AZURE BATCH
GCP PROCESSING PIPELINES
DATA LIFECYCLE

ELASTIC DATA PROCESSING PIPELINE

```
        .apply(Sum.integersPerKey()) //
        .apply("ToView", View.asMap());

// packages in terms of use and which r
javaContent //
        .apply("IsPopular", ParDo.of(ne
            @ProcessElement
            public void processElement(
                String[] lines = c.elem
                String[] packages = par
                for (String packageName
                    c.output(KV.of(pack
```



OPEN-SOURCE API, GOOGLE INFRASTRUCTURE



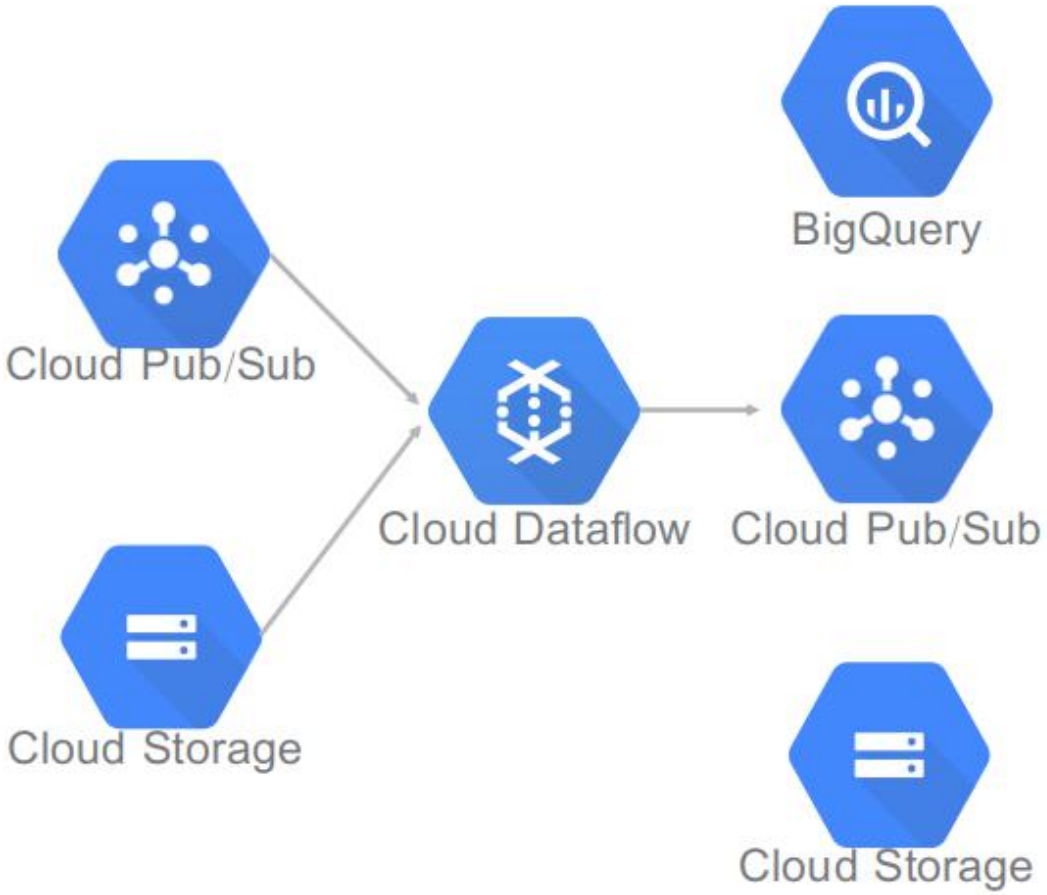

```
p = beam.Pipeline()
(p
    | beam.io.ReadFromText('gs://..')
    | beam.Map(Transform)
    | beam.GroupByKey()
    | beam.FlatMap(Filter)
    | beam.io.WriteToText('gs://...')
)
p.run();
```

Open-source API (Apache Beam) can be executed on Flink, Spark, etc. also

Parallel tasks (autoscaled by execution framework)

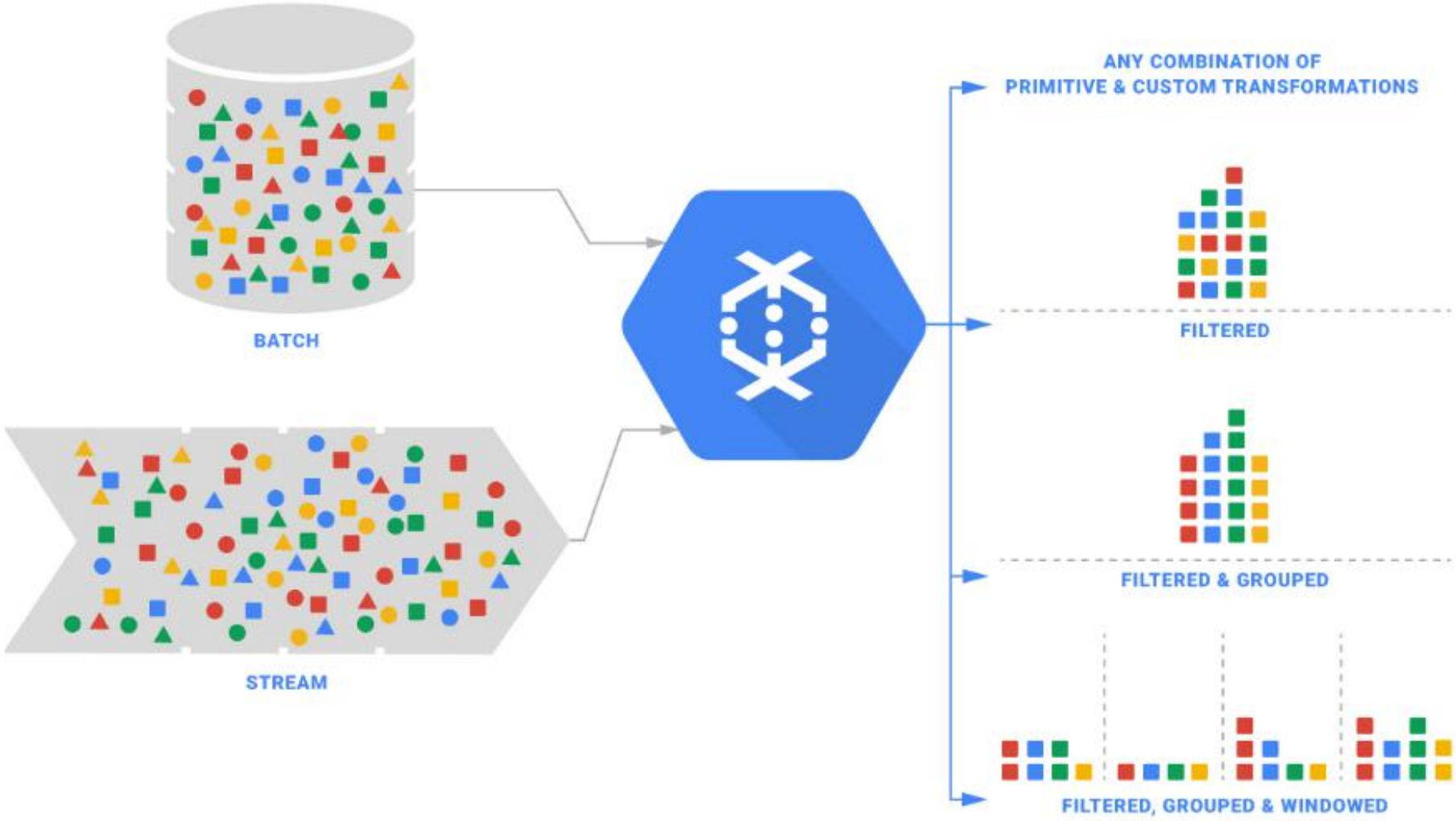
```
def Transform(line):
        return (parse_custid(line), 1)

def Filter(key, values):
        return sum(values) < 10
```

SAME CODE DOES REAL-TIME AND BATCH




```
Pipeline p = Pipeline.create();
  p.begin()
    .apply(PubsubIO.readStrings().fromTopic(topic))
    .apply(Window.into(SlidingWindows
                       .of(Duration.standardMinutes(60))
    .apply(ParDo.of(new Filter1()))
    .apply(new Group1())
    .apply(ParDo.of(new Filter2())
    .apply(new Transform1())
    .apply(BigQueryIO.writeTableRows().to(tbl));
  p.run();
```

DATAFLOW DOES INGEST, TRANSFORM, AND LOAD



A PIPELINE IS A DIRECTED GRAPH OF STEPS

Read in data, transform it, write out

- Can branch, merge, use if-then statements, etc.

```java
import org.apache.beam.sdk.Pipeline;  // etc.
public static void main(String[]args){
    // Create a pipeline parameterized by commandline flags.
    Pipeline p=Pipeline.create(PipelineOptionsFactory.fromArgs(args));

    p.apply(TextIO.read().from("gs://..."))   // Read input.
    .apply(new CountWords())                  // Do some processing.
    .apply(TextIO.write().to("gs://..."));   // Write output.
    // Run the pipeline.
    p.run();

}
```

PYTHON API CONCEPTUALLY SIMILAR

Read in data, transform it, write out

- Pythonic syntax

```
import apache_beam as beam

if __name__ == '__main__':
   # create a pipeline parameterized by commandline flags
   p = beam.Pipeline(argv=sys.argv)

   (p
      | beam.io.ReadFromText('gs://...') # read input
      | beam.FlatMap(lambda line: count_words(line)) # do some processing
      | beam.io.WriteToText('gs://...') # write output
   )

   p.run() # run the pipeline
```

APPLY TRANSFORM TO PCOLLECTION

Data in a pipeline are represented by PCollection

- Supports parallel processing
- Not an in-memory collection; can be unbounded

```
1 PCollection<String>  lines = p.apply(...)   //
```

Apply Transform to PCollection; returns PCollection

```
1 PCollection<Integer> sizes =
2   lines.apply("Length", ParDo.of(new DoFn<String, Integer>() {
3   @ProcessElement
4   public void processElement(ProcessContext c) throws Exception {
5   String line = c.element();
6   c.output(line.length());
7 }}))
```

APPLY TRANSFORM TO PCOLLECTION (PYTHON)

- Data in a pipeline are represented by PCollection
  - Supports parallel processing
  - Not an in-memory collection; can be unbounded

```
lines = p | ...
```

  - Apply Transform to PCollection; returns PCollection

```
sizes = lines | 'Length' >> beam.Map(lambda line: len(line) )
```

INGESTING DATA INTO A PIPELINE

- Read data from file system, GCS, BigQuery, Pub/Sub
  - Text formats return String

```
PCollection<String> lines = p.apply(TextIO.read().from('gs://.../input-*.csv.gz');
```

```
PCollection<String> lines = p.apply(PubsubIO.readStrings().fromTopic(topic));
```

  - BigQuery returns a TableRow

```
String javaQuery = 'SELECT x, y, z FROM [project:dataset.tablename]';
PCollection<TableRow> javaContent = p.apply(BigQueryIO.read().fromQuery(javaQuery))
```

CAN WRITE DATA OUT TO SAME FORMATS

- Write data to file system, GCS, BigQuery, Pub/Sub

```
lines.apply(TextIO.write().to('/data/output').withSuffix('.txt'))
```

- Can prevent sharding of output (do only if it is small)

```
.apply(TextIO.write().to('/data/output').withSuffix('.csv').withoutSharding())
```

- May have to transform PCollection<Integer>, etc. to PCollection<String> before writing out

EXECUTING PIPELINE (JAVA)

- Simply running main( ) runs pipeline locally

```
java -classpath …        com…
```

```
mvn compile -e exec:java -Dexec.mainClass=    MAIN
```

- To run on cloud, submit job to Dataflow

```
mvn compile -e exec:java \
      -Dexec.mainClass=    MAIN \
      -Dexec.args="--project=    PROJECT \
      --stagingLocation=gs://    BUCKET/staging/ \
      --tempLocation=gs://    BUCKET/staging/ \
      --runner=DataflowRunner"
```

EXECUTING PIPELINE (PYTHON)

- Simply running main( ) runs pipeline locally

```
python ./grep.py
```

- To run on cloud, specify cloud parameters

```
python ./grep.py \
    --project=   PROJECT \
    --job_name=myjob \
    --staging_location=gs://   BUCKET/staging/ \
    --temp_location=gs://   BUCKET/staging/ \
    --runner=DataflowRunner
```

EXAMPLE TEMPLATES FOR BASIC TASKS

WordCount

Cloud Pub/Sub to BigQuery

Cloud Storage Text to Cloud Pub/Sub

Cloud Pub/Sub to Cloud Storage Text

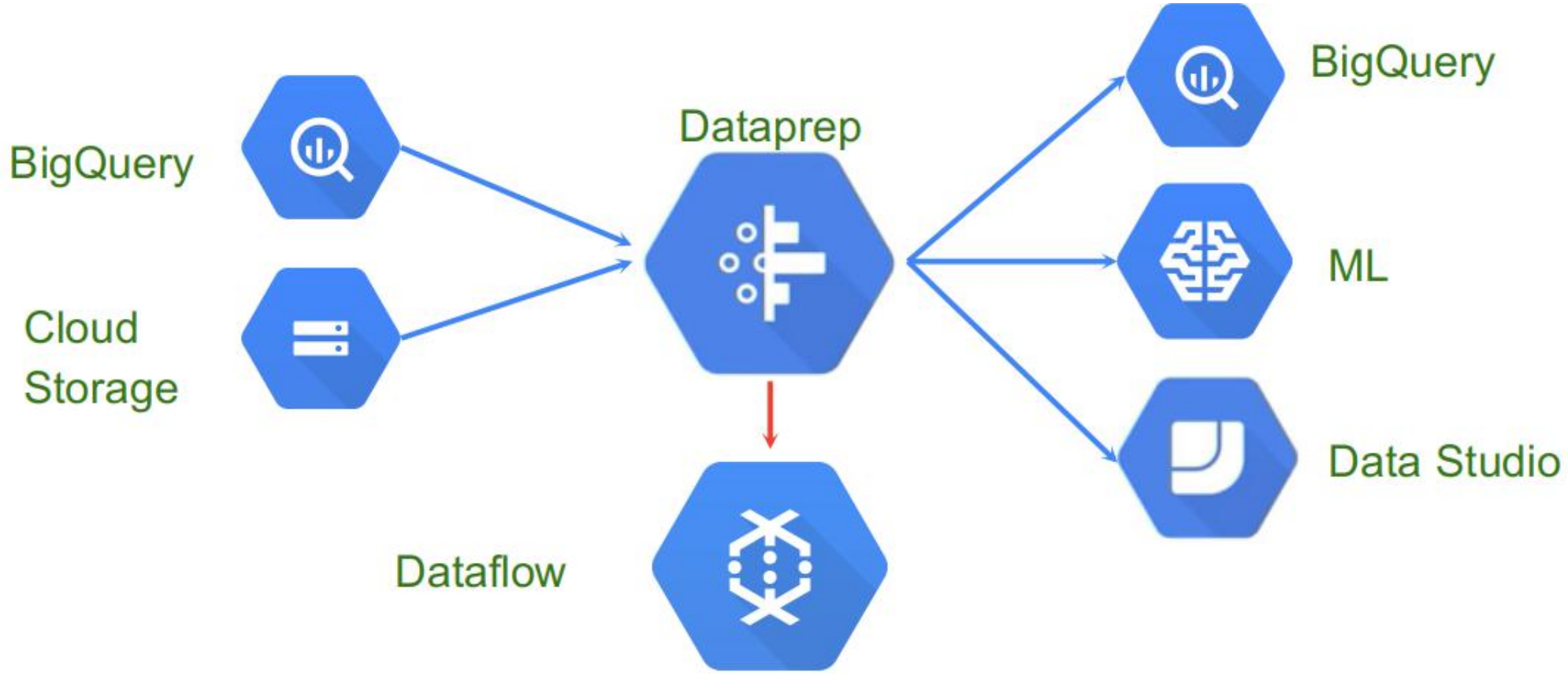
Cloud Datastore to Cloud Storage Text

Cloud Storage Text to BigQuery

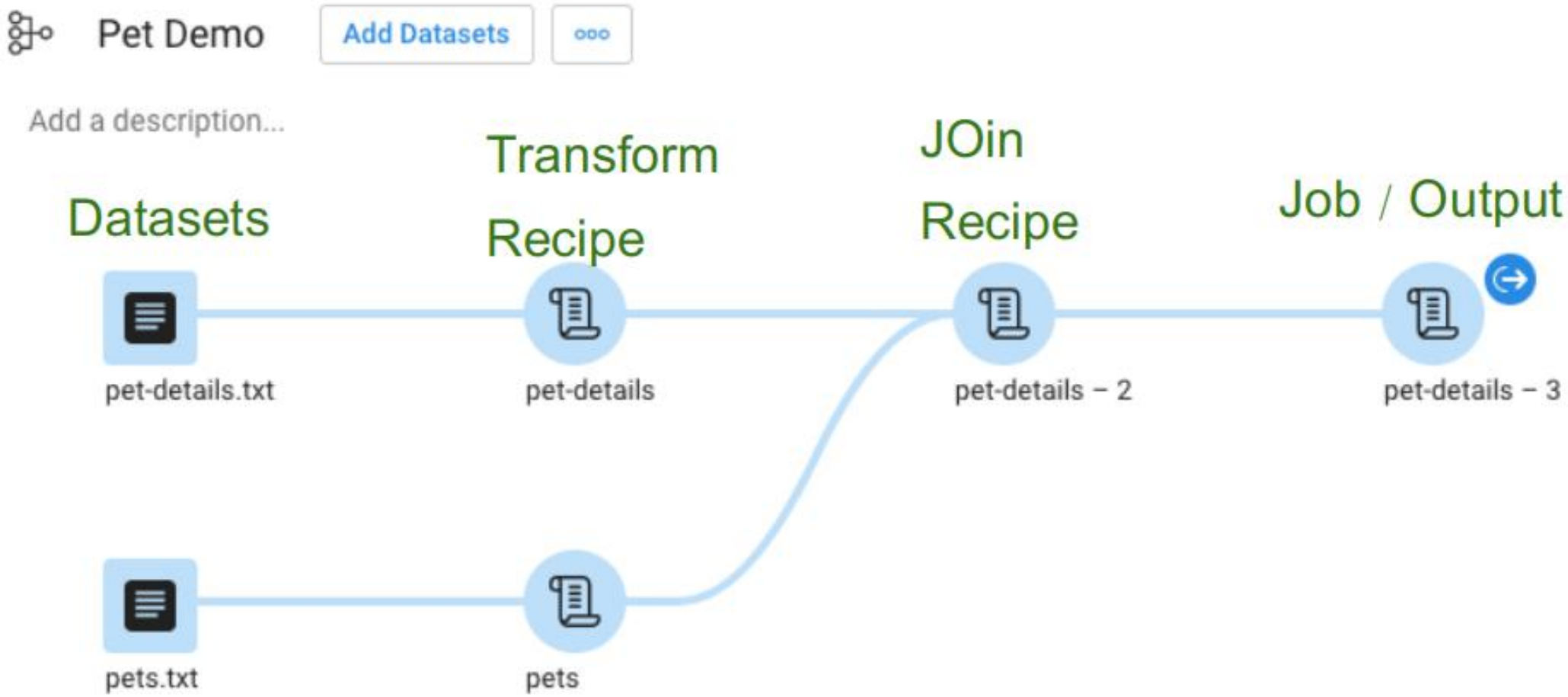
Cloud Storage Text to Cloud Datastore

Bulk Decompress Cloud Storage Files

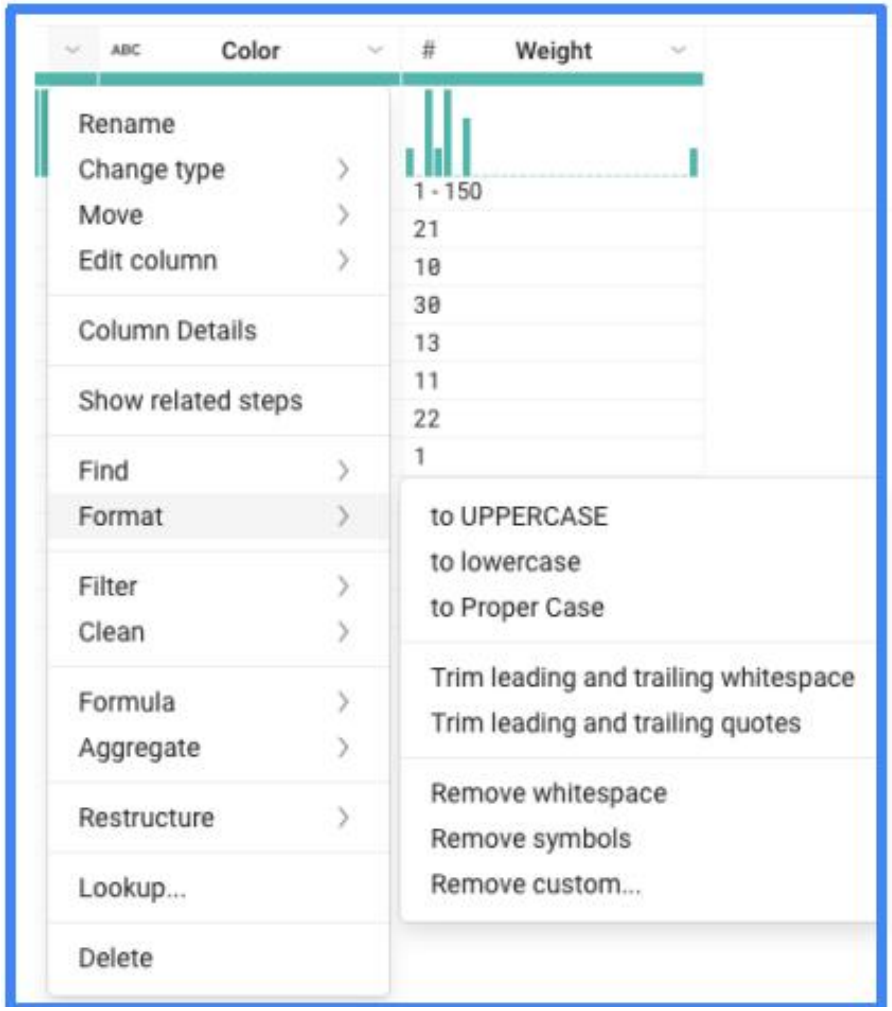
DATAPREP



CREATE PIPELINES IN DATAPREP FLOWS



CLEANUP, STRUCTURING, AND TRANSFORMATION



Clean
Structure
Prepare

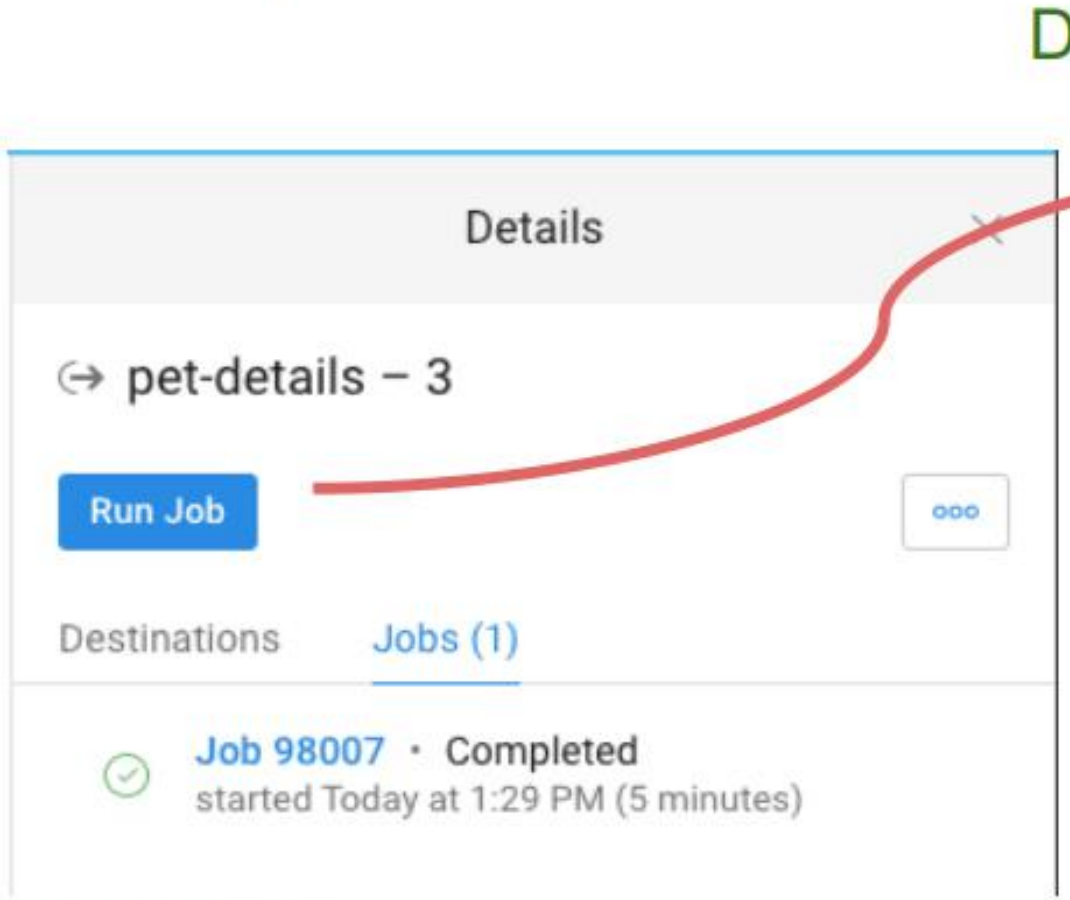
pet-details
Pet Demo • Full Data

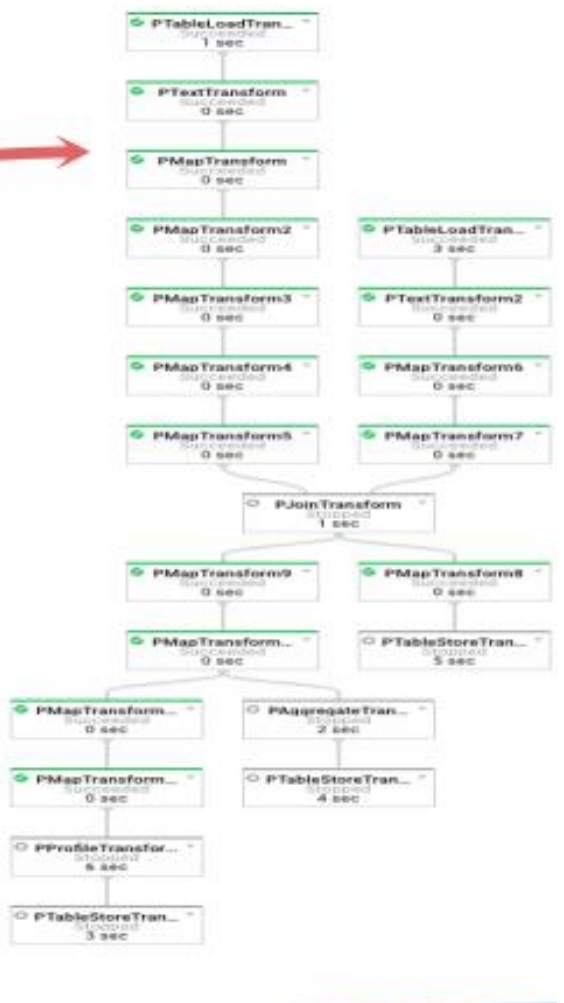
Grid | Columns

Find column

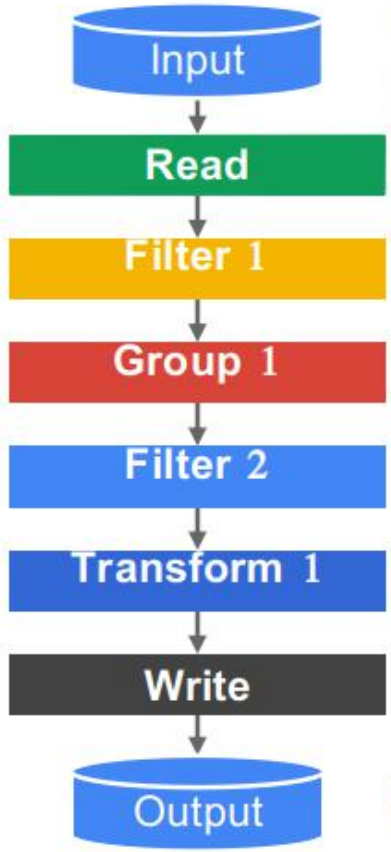
	ABC column2	ABC PetName	ABC Breed	ABC Color	#
	4 Categories	11 Categories	11 Categories	6 Categories	1 - 150
·	Dog	Noir	Schnoodle	Black	21
·	Dog	Bree	MaltePoo	White	10
·	Dog	Pickles	Golden Retriever	Yellow	30
·	Dog	Sparky	Mutt	Gray	13
·	Cat	Tom	Alley	Yellow	11
·	Cat	Cleo	Siamese	Gray	22
·	Frog	Kermit	Bull	Green	1
·	Pig	Bacon	Pot Belly	Pink	30
·	Pig	Babe	Domestic	White	150
·	Dog	Rusty	Poodle	White	20
·	Cat	Joe	Mix	Black	15

DATAPREP TO GENERATE DATAFLOW PIPELINES





OPEN-SOURCE API, GOOGLE INFRASTRUCTURE




```
Pipeline p = Pipeline.create( );
p
        .apply(TextIO.read( ).from("gs://…"))
        .apply(ParDo.of(new Filter1( )))
        .apply(new Group1( ))
        .apply(ParDo.of(new Filter2( ))
        .apply(new Transform1( ))
        .apply(TextIO.write( ).to("gs://…"));
p.run( );
```

Open-source API (Apache Beam) can be executed on Flink, Spark, etc. also

Parallel task (autoscaled by execution framework)

```
class Filter1 extends DoFn<...I {
  public void
  processElement(ProcessContext c) {
      ... = c.element( );
      ...
      c.output( ...);
  }
```

DATA LIFECYCLE

AWS BATCH
AZURE BATCH
GCP PROCESSING PIPELINES
DATA LIFECYCLE

GCP DATA LIFECYCLE

Ingest:

- pull in the raw data

Store:

- data needs to be stored in a format that is durable and can be easily accessed.

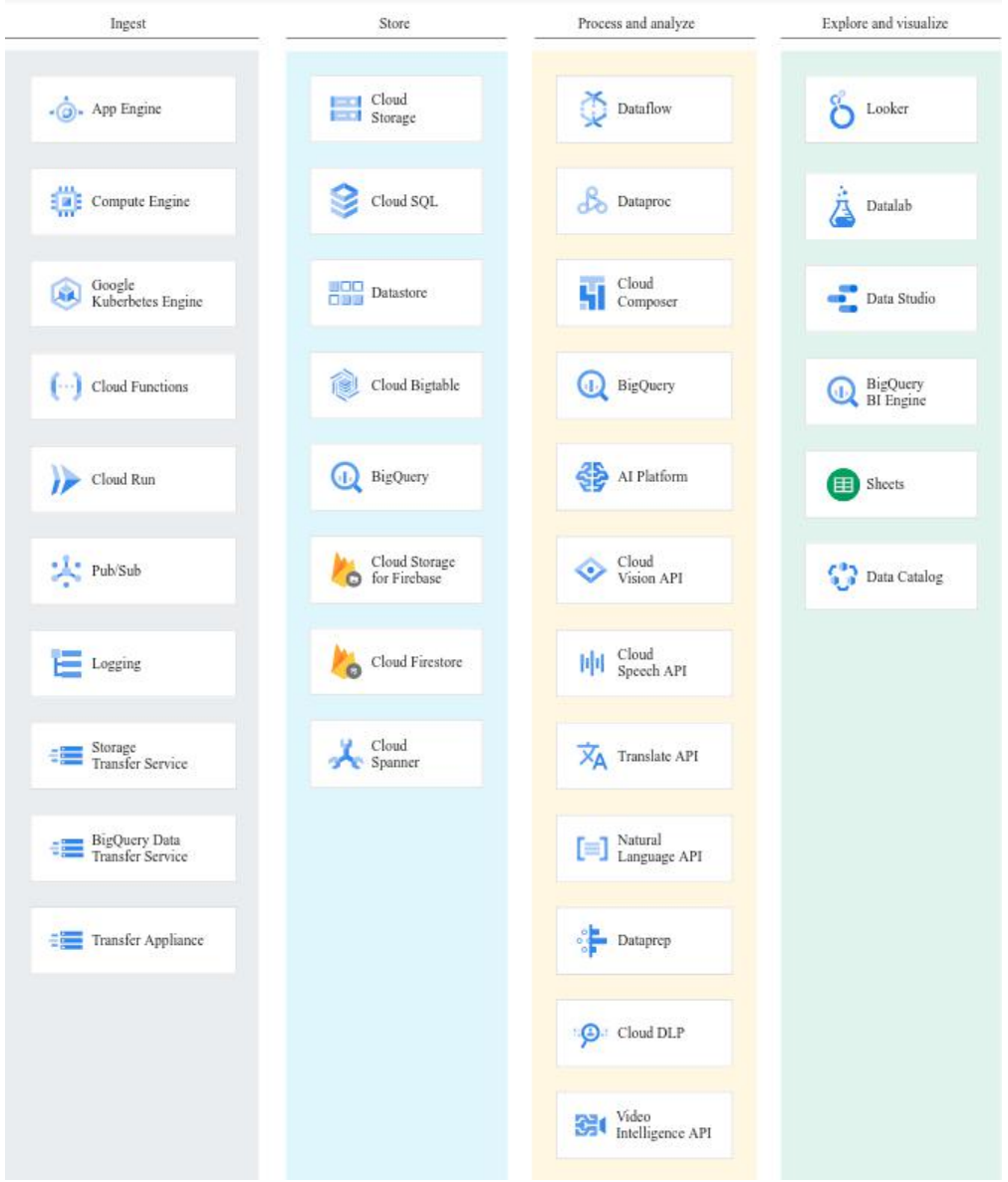
Process and analyze:

- the data is transformed from raw form into actionable information.

Explore and visualize:

- The final stage is to convert the results of the analysis into a format that is easy to draw insights from and to share with colleagues and peers.

LIFECYCLE TOOLS



INGEST

App:

- Data from app events, such as log files or user events

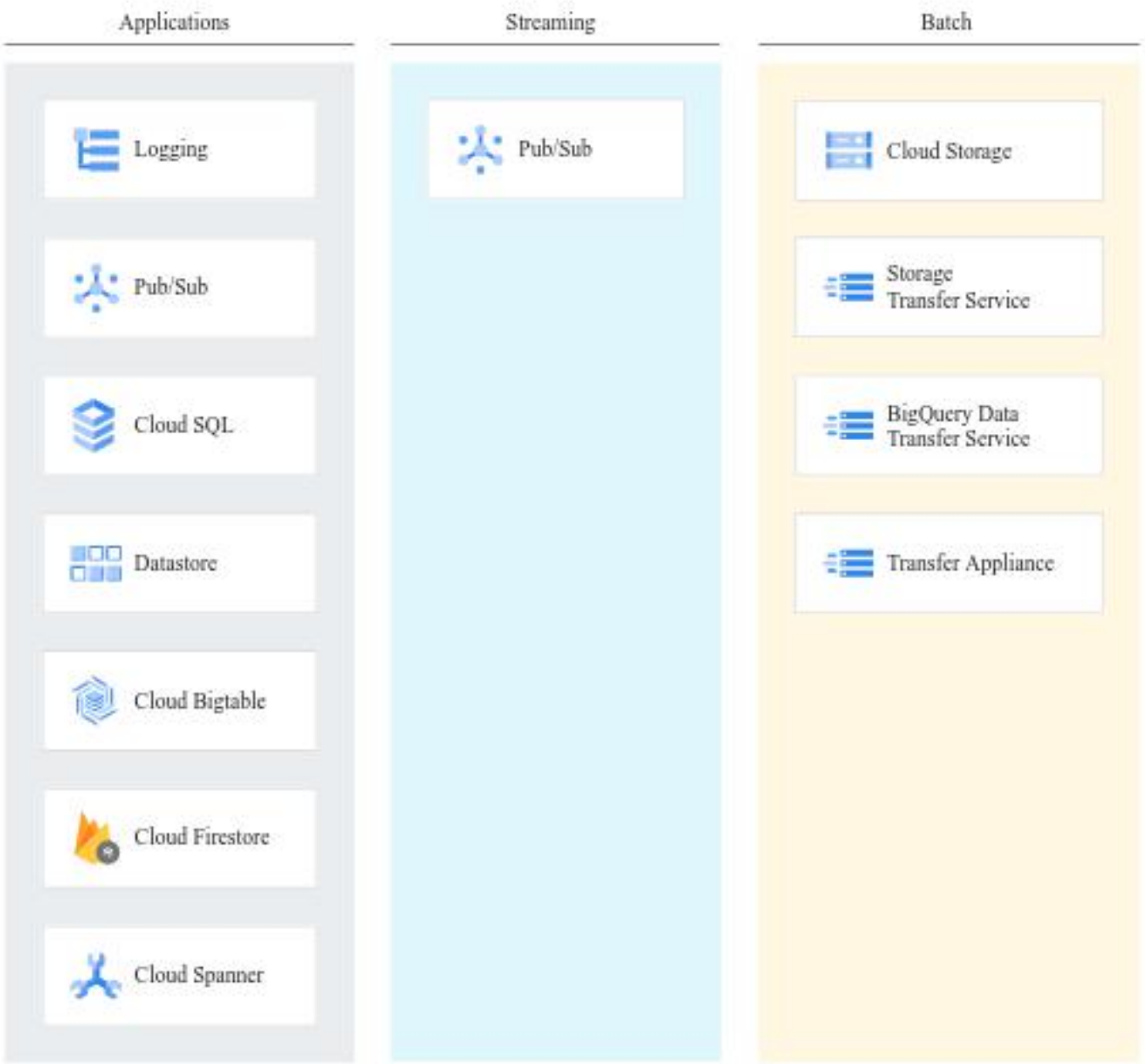
Streaming:

- The data consists of a continuous stream of small, asynchronous messages.

Batch: Large amounts of data are stored in a set of files

- that are transferred to storage in bulk.

INGEST



INGESTING STREAMING DATA

Telemetry data:

- Internet of Things (IoT) devices

User events and analytics:

- A mobile app might log events

INGESTING BULK DATA

Scientific workloads:

- Genetics data stored in Variant Call Format (VCF)

Migrating to the cloud:

- Moving data stored in an on-premises Oracle database

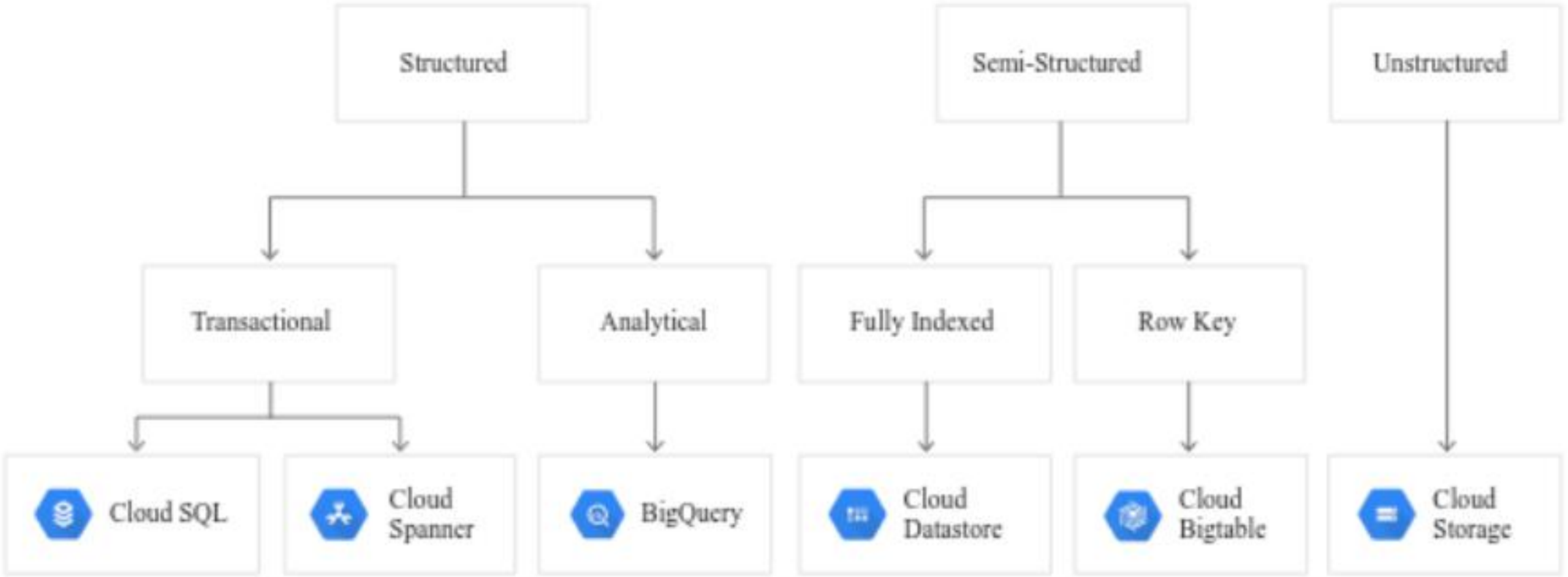
Backing up data:

- Replicating data stored in an AWS bucket

Importing legacy data:

- Copying ten years worth of website log data

STORE



PROCESS AND ANALYZE

Processing:

- Data from source systems is cleansed, normalized

Analysis:

- Processed data is stored in systems that allow for ad-hoc querying

Understanding: train and test

CONGRATS ON COMPLETION

TODO

Will you discuss running BEAM on AWS and Azure? E.g., using AWS Kinesis with BEAM and/or Flink?

Could you say anything about Airflow as compared to BEAM, and when to use which (general task execution vs. dataflows)? E.g., is it worth mentioning Amazon Managed Workflows for Apache Airflow (MWAA), or GCP CloudComposer (built on Airflow)?

I like the discussion of BEAM and GCP, but not all workflows are dataflows.