Conceptualizing the Processing Model for the GCP Dataflow Service

GETTING STARTED WITH CLOUD DATAFLOW

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Understanding the Cloud Dataflow programming model

Executing streaming pipelines in Cloud Dataflow

Building and running a data pipeline using Java and Maven

Understanding Cloud Dataflow pricing

Prerequisites and Course Outline

Prerequisites

Comfortable programming in Java

Familiar with the Apache Beam programming model for streaming

Some familiarity with the Google Cloud Platform

Prerequisite Courses

Modeling Streaming Data for Processing with Apache Beam

Exploring the Apache Beam SDK for Modeling Streaming Data for Processing

Prerequisite Courses

Google Cloud Platform Fundamentals - Core Infrastructure

Course Outline

Getting Started with Cloud Dataflow

Monitoring Jobs in Cloud Dataflow

Optimizing Cloud Dataflow Pipelines

Running Cloud Dataflow Pipelines Using Templates

Introducing Apache Beam

Apache Beam

Open-source, unified model for defining both batch and streaming, data-parallel piplines.

Using Apache Beam

Write code for pipeline

Submit job for execution

Backend assigns workers to execute

Pipeline parallelized and executed

Writing Code

Java

Python

Go

Scio - a Scala interface

Driver Program

Driver program utilizes Beam SDKs

Defines pipeline

Input, transforms, outputs

Execution options for pipeline

Driver program is executed on one of the Apache Beam backends

Available Backends

Apache Flink

Apache Spark

Google Cloud Dataflow

Apache Samza

Hazelcast Jet

Beam and Runners

Apache Beam	Runners
API specification	API implementation
Platform-agnostic	Platform-dependent
Superset of all actually provided capabilities	Only subset of Apache Beam APIs implemented by each backend

Cloud Dataflow

Cloud Dataflow

Unified stream and batch data processing that is serverless, fast, and cost-effective

Cloud Dataflow

Unified stream and batch data processing that is **serverless**, fast, and cost-effective

Cloud Dataflow

Define execution pipelines using the Apache Beam unified processing model

Cloud Dataflow

A distributed processing execution backend for Apache Beam

Fully managed service on the GCP

Automated provisioning and management of resources

Horizontal autoscaling of workers

Reliable, consistent, exactly-once processing

Apache Beam Pipeline Components

Data source

Batch or streaming data to be processed

Transformations

Modify the data to get it in the right final form

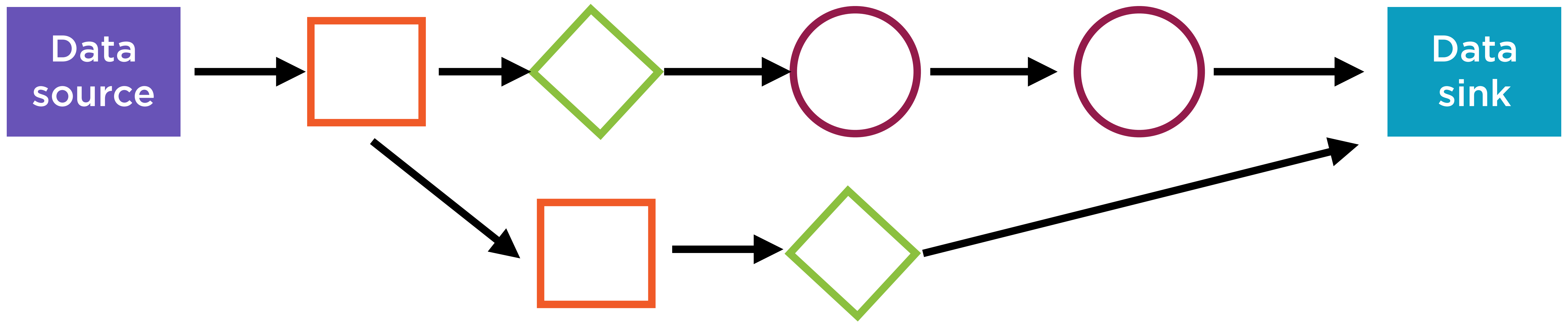
Data sink

Store the data in some kind of persistent storage

Pipeline

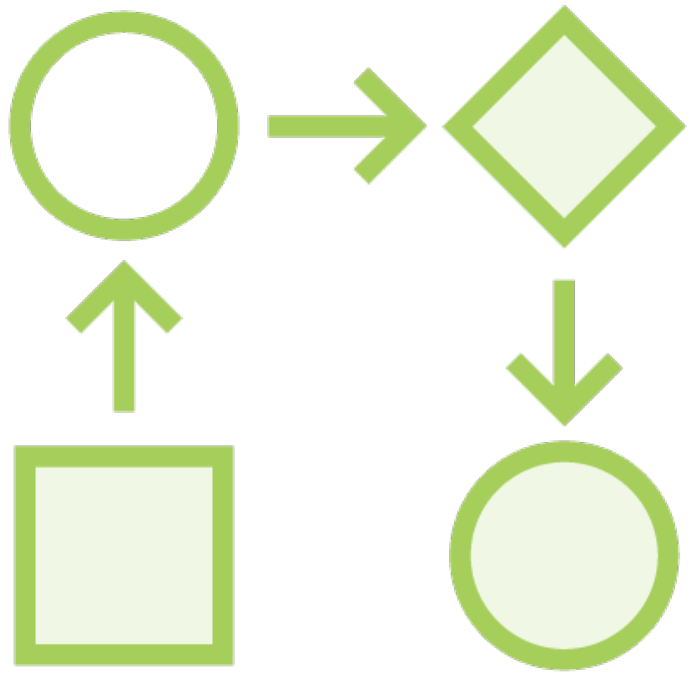
Encapsulates all data and steps in a data processing task in an object of the Pipeline class of the Beam SDK.

Pipeline



Directed Acyclic Graph (DAG)

PipelineOptions



Configure using PipelineOptions objects

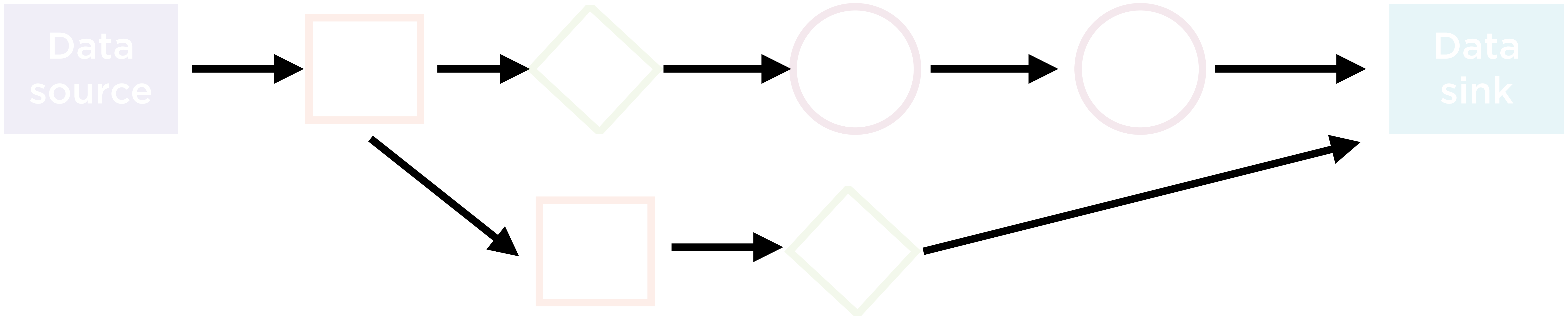
- Encapsulate key-value pairs

Includes choice of runner

Can do via command line arguments

Alternatively, create custom options

Pipeline

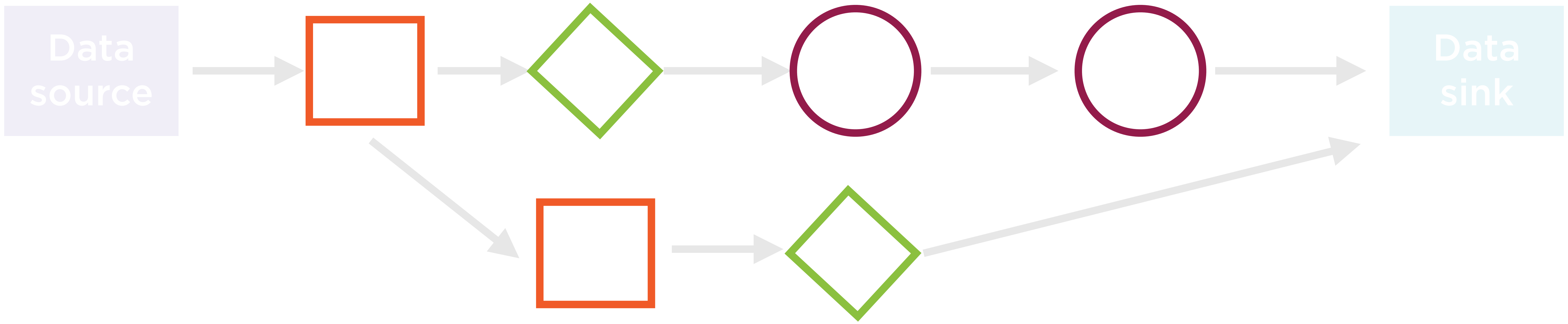


PCollections: Edges of DAG

PCollection

Interface in the Beam SDK; represents a multi-element data set which may or may not be distributed. Can be created by reading from an external data source, or by transforming another PCollection.

PTransform



PTransforms: Nodes in DAG

PTransform

Interface in the Beam SDK; represents single step of the pipeline that takes in an input PCollection and transforms it to zero or more output PCollections.

Executing Cloud Dataflow Pipelines

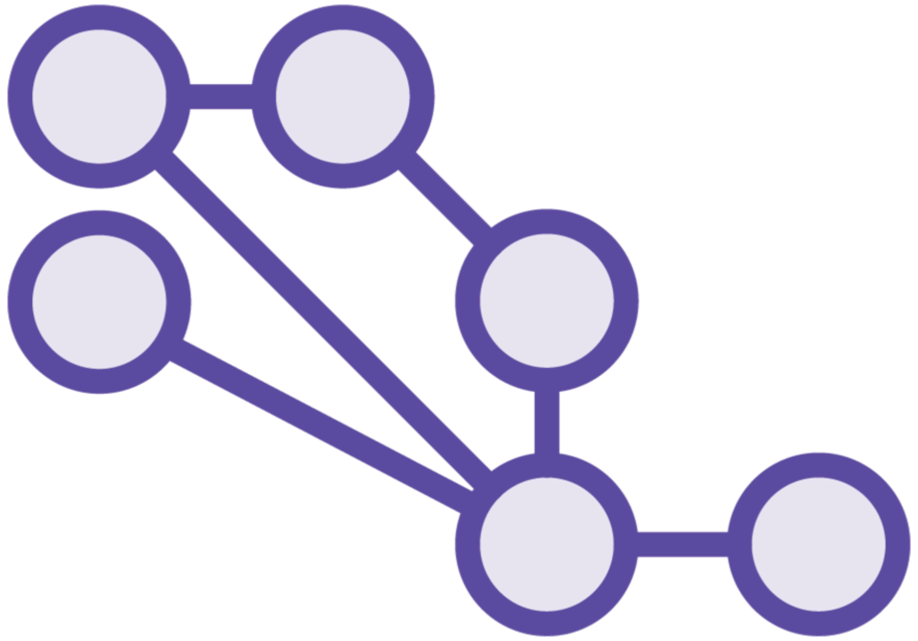
Dataflow Service

Fully managed service to execute pipelines

Uses other GCP services such as Compute Engine and Cloud Storage

Automatically spins up and tears down resources to run your job

Graph Construction Time



Dataflow creates an execution graph including all processing operations

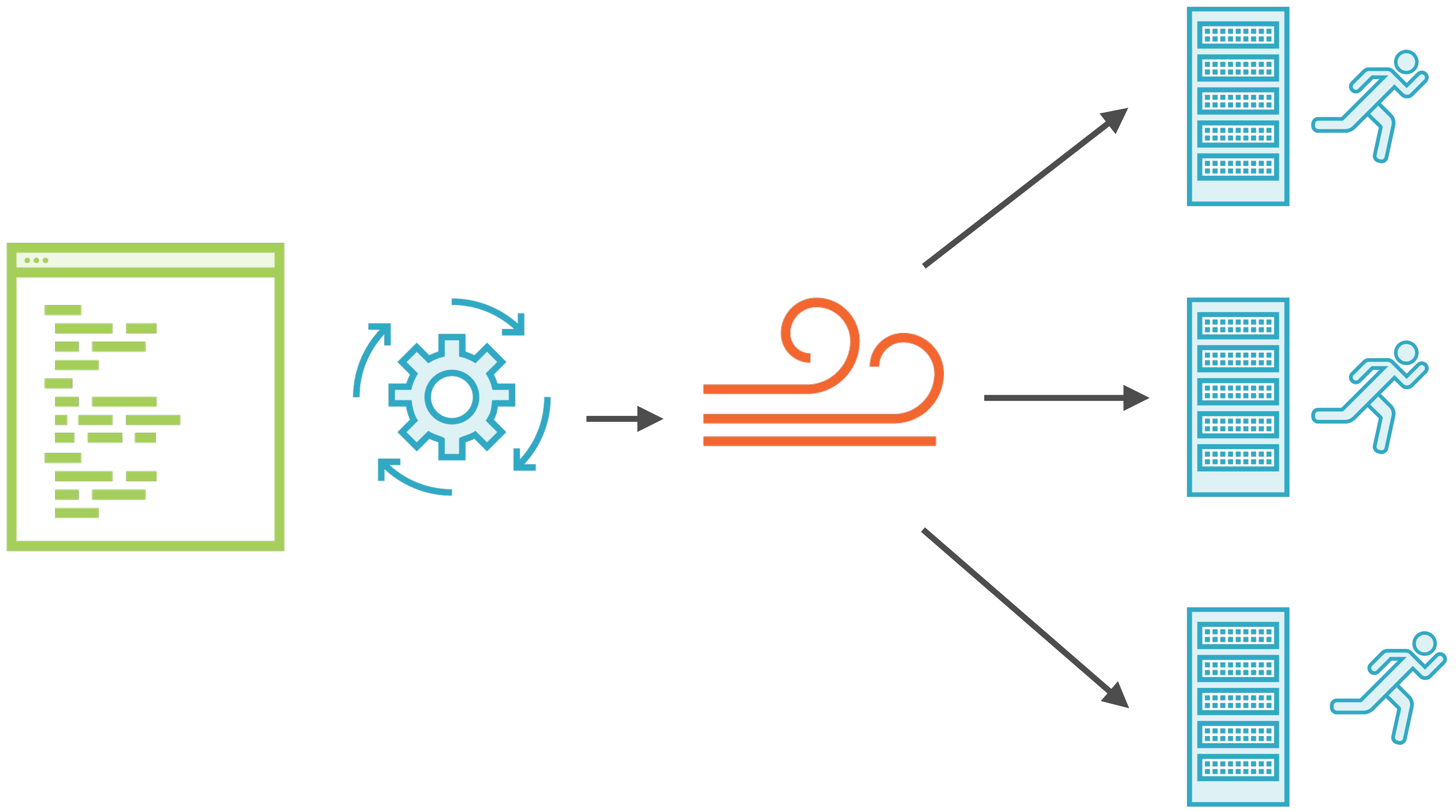
Runs locally where pipeline is executed

Validates resources referenced by the pipeline

Existence of Cloud Storage buckets, BigQuery tables, Pub/Sub Topics

Checks for errors and illegal operations

Parallelization and Distribution



Dataflow automatically partitions your data and distributes your worker code for parallel processing

Execution Using Workers

Serverless and no-ops

- GCP automatically assigns resources needed to run job

Autoscaling

- GCP automatically adds or removes capacity based on workload

Parallelized

- Operations in code are executed in parallel by platform

Resource Management

Dataflow spins up and shuts down worker virtual machines as needed

Cloud Storage buckets are used for I/O as well as temporary file staging

Pipelines can integrate with and access other GCP services such as Pub/Sub and BigQuery

Resource Management

Limit of 25 concurrent Dataflow jobs in a project

Can be increased on request

Maximum of 1000 compute engines per job

Maximum of 15 persistent disks per worker instance

Streaming Engine

The Dataflow pipeline runner executes the pipeline on worker VMs

The streaming engine moves pipeline execution into the Dataflow service backend

Streaming Engine

Reduced resource usage in terms of consumed CPU, memory, and disk

More responsive autoscaling based on incoming data volume

No pipeline redeployment to apply service updates

Streaming Engine

Uses smaller worker machine types

Workers only require a small boot disk

Available only in some GCP regions

Cost of pipeline execution roughly the same as the regular Dataflow runner

Demo

Enabling Google Cloud Platform APIs

Creating service account keys

Demo

Creating a Maven project and running the example word count program using the direct runner

Demo

Creating Cloud Storage buckets

Uploading data to buckets

Demo

Implementing and executing a Dataflow job

Dataflow Pricing

Using Dataflow

Write code for pipeline

Submit job for execution

Dataflow assigns workers to execute

Pipeline parallelized and executed

Using Dataflow

Write code for pipeline

Submit job for execution

Dataflow assigns workers to execute

Pipeline parallelized and executed

Total Price = Compute Cost + Storage Cost + Data Processing Cost

Cloud Dataflow Pricing

Three components of pricing - based on workers used in batch or stream processing

Total Price = **Compute Cost** + Storage Cost + Data Processing Cost

Compute Cost

Per-second billing of vCPUs and memory used in batch or streaming workers

Total Price = **Compute Cost** + Storage Cost + Data Processing Cost

Compute Cost

Standard worker configurations for batch: 1 vCPU, 3.75GB memory and 250 GB Persistent Disk

Total Price = **Compute Cost** + Storage Cost + Data Processing Cost

Compute Cost

Standard worker configurations for streaming: 4 vCPU, 15GB memory and 420 GB Persistent Disk

Total Price = **Compute Cost** + Storage Cost + Data Processing Cost

Compute Cost: vCPU

Batch: $0.056 /vCPU/hour; Streaming: $0.069 /vCPU/hour

Total Price = **Compute Cost** + Storage Cost + Data Processing Cost

Compute Cost: Memory

Batch: $0.003557 /GB/hour; Streaming: $0.003557 /GB/hour

Total Price = Compute Cost + **Storage Cost** + Data Processing Cost

Storage Cost: Standard Persistent Disk

Batch: $0.000054 /GB/hour; Streaming: $0.000054 /GB/hour

Total Price = Compute Cost + **Storage Cost** + Data Processing Cost

Storage Cost: SSD Persistent Disk

Batch: $0.000298 /GB/hour; Streaming: $0.000298 /GB/hour

Using Dataflow

Write code for pipeline

Submit job for execution

Dataflow assigns workers to execute

Pipeline parallelized and executed

Using Dataflow

Write code for pipeline

Submit job for execution

Dataflow assigns workers to execute

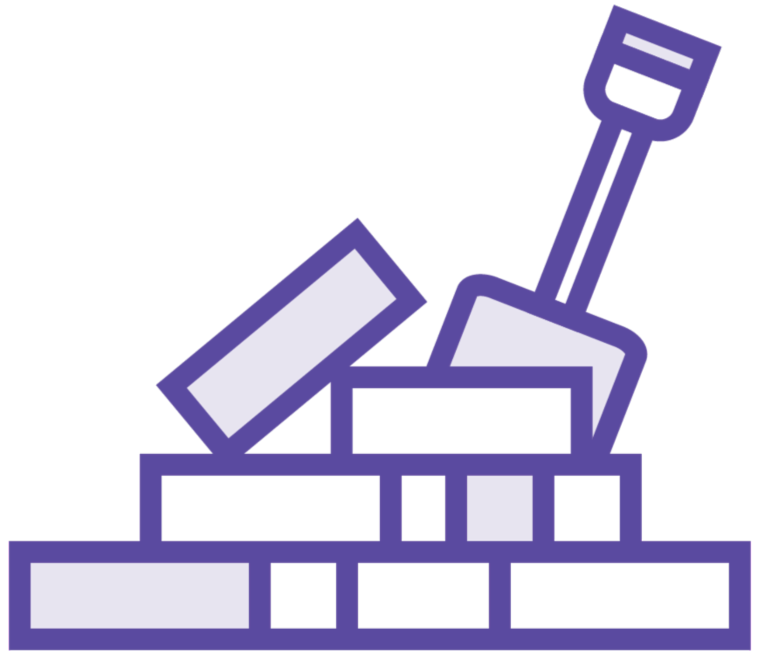
Pipeline parallelized and executed

Total Price = Compute Cost + Storage Cost + **Data Processing Cost**

Data Processing Cost

Batch: $0.011 /GB; Streaming: $0.018 /GB

Additional Resources



Billed at its own pricing:

- Cloud Storage
- Pub/Sub
- Bigtable
- BigQuery

Summary

Understanding the Cloud Dataflow programming model

Executing streaming pipelines in Cloud Dataflow

Building and running a data pipeline using Java and Maven

Understanding Cloud Dataflow pricing

Up Next:

Monitoring Jobs in Cloud Dataflow