Dataproc - Michaël Bettan

Definition: Managed Apache Spark and Hadoop service to leverage OSS Hadoop ecosystem tools (Pig, Hive, Spark, etc.).	**Availability**: Regional	**SLA**: >=99.5%

Use Cases:

- **Big Data Processing**: Process vast amounts of structured and unstructured data using Hadoop, Spark, etc.
- **Data Warehousing & ETL**: Scalable data extraction, transformation, and loading using Hive, Presto, Spark SQL.
- **Machine Learning**: train & deploy ML models with Spark MLlib
- **Real-time Analytics**: streaming data w/ Flink & Spark Streaming
- **Log/Data Analysis**: Process and analyze logs or clickstreams
- **Hadoop/Spark Migration**: Migrate existing clusters from on-premises and other clouds environment
- **Interactive Big Data Analysis**: Querying/Scripting w/ Hive/Pig
- **Data Lakehouse**
- **Interactive data science** and **notebooks**

Billing: billed by the second (1-minute minimum)

- Underlying infrastructure costs: You pay for the standard resources used, including:
  - Compute Engine VMs (cost varies by machine type)
  - Persistent Disk storage
  - Cloud Storage (operations, storage size, and class)
  - Cloud Monitoring
  - Global Networking
- Dataproc managed service cost: $0.01 per vCPU per hour:
  - Aggregate # of vCPUs across the entire cluster
  - Duration of running cluster

Core Components

- *Data can no longer fit in memory on one machine (monolithic), so a new way of computing was devised using many computers to process the data (distributed). Such a group is called a cluster, which makes up server farms. All of these servers have to be coordinated in the following ways: partition data, coordinate computing tasks, handle fault tolerance/recovery, and allocate capacity to process.*
- ***Hadoop** is an open source **distributed processing framework** that manages data processing and storage for big data applications running in clustered systems.*

HDFS (*Hadoop Distributed File System*)

Distributed file system that provides high-throughput access to application data by partitioning data across many machines

- *Each disk on a different machine in a cluster consists of 1 master node; the rest are data nodes. The master node manages the overall file system by storing the directory structure and metadata of the files. The data nodes physically store the data. Large files are broken up/distributed across multiple machines, which are replicated across 3 machines to provide fault tolerance.*

YARN (Yet Another Resource Negotiator)

Framework for job scheduling and cluster resource management (task coordination)

- *Coordinates tasks running on the cluster and assigns new nodes in case of failure:*
  - ***The resource manager** runs on a single master node and schedules tasks across nodes.*
  - ***The node manager** runs on all other nodes and manages tasks on the individual node.*

MapReduce

YARN-based system for parallel processing of large data sets on multiple machines

- *Parallel programming paradigm which allows for processing of huge amounts of data by running processes on mul- tiple machines. Defining a MapReduce job requires two stages: map and reduce.*

<table>
<tr><td rowspan="3">Sqoop
Import and export of relational data</td><td rowspan="4">HBase:
NoSQL datastore</td><td>Hive: SQL DW</td><td>MLLib & Spark ML: Machine Learning</td><td rowspan="2">Zookeeper:
Coordination</td></tr>
<tr><td>Pig: Scripting</td><td>Flink/Kafka: Streams</td></tr>
<tr><td>Spark: Cluster data processing</td><td>Presto: Distributed SQL Query</td><td rowspan="2">Oozie:
Workflow automation</td></tr>
<tr><td colspan="2">MapReduce: Cluster data processing</td></tr>
<tr><td colspan="4">Yarn: Job scheduling & cluster resource management</td></tr>
<tr><td colspan="4">HDFS: Hadoop Distributed File System</td></tr>
</table>

Data Processing

- ***Spark** is a fast, general-purpose distributed computing framework for data processing. It improves upon Hadoop MapReduce by leveraging fast in-memory computation. Unified engine for SQL, streaming, machine learning, and graph processing, and can run with or without Hadoop, integrating seamlessly when used together. Its core data abstraction is the **Resilient Distributed Dataset (RDD)**, which is immutable, lazily evaluated, and fault-tolerant through lineage tracking.*
- ***Hive** is a data warehousing and SQL-like query tool built on Hadoop, enabling users to manage and query large datasets stored in HDFS using the SQL-based HiveQL language. It translates HiveQL queries into MapReduce jobs, eliminating the need for users to write Java MapReduce code.*
- ***Pig** is a high-level scripting language (Pig Latin) designed for data transformation (ETL), commonly used to prepare raw data for analysis in data warehouses. It complements tools like Hive by enabling complex data processing before querying. Pig supports **checkpointing**—allowing you to store intermediate data during processing, which is beneficial for complex pipelines that need to resume from specific points in case of failure. Allows you to **split pipelines** to process data in parallel or route different parts of the dataset through different transformations.*

Data Analysis

- ***Spark SQL*** *allows users to process structured and semi-structured data using SQL or a DataFrame API. It provides a programming abstraction called DataFrames and can act as a distributed SQL query engine*
- ***Presto****: Distributed SQL query engine for interactive data analysis. Supports various data sources.* ***Trino*** *is an improved version with enhanced performance*

Data Ingestion and Storage

- ***HBase*** *is a non-relational, NoSQL, column-oriented database that runs on HDFS. Ideal for handling sparse datasets*
- ***Sqoop****: transferring framework to transfer large amounts of data into HDFS from relational databases (MySQL)*
- ***Flume****: A distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.*

Data Streaming

- ***Kafka****: Distributed streaming platform for building real-time data pipelines.*
- ***Kafka Connect*** *is a tool to connect Kafka with other systems so that the data can be easily streamed via Kafka. Connectors are built for systems like MySQL and Postgres and turn data from their CDC systems into Kafka topics*
- ***Flink****: Stream processing framework for stateful computations over unbounded and bounded data streams.*
- ***Debezium*** *is a distributed, OSS platform designed for real-time change data capture (CDC). It captures row-level changes in databases and streams them as events, typically to Kafka, enabling other systems to consume these changes efficiently and in near real-time.*

Workflow and Coordination

- ***Zookeeper*** *manages the distributed environment and handles configuration across each service*
- ***Oozie****: workflow scheduler system to manage Hadoop jobs. Complex workflows with dependencies and schedules.*

Data Serialization and Management

- ***Avro****: data serialization system that provides a compact, fast, binary data format. It's often used with other Hadoop components for data exchange.*
- ***Parquet****: A columnar storage format for Hadoop designed for efficient querying and analysis.*

Dataproc Architecture & Components

- ***On-Premise limitations****: not elastic, scale slowly, capacity, no separation between storage and compute resources*
- *On-demand, managed Hadoop and Spark (ephemeral) clusters*
- *Reduce operation tasks but requires cluster creation, deletion, configuration and scaling policies*
- ***Master and Worker nodes:*** *master node is responsible for distributing and managing workload distribution.*
    - ***Primary*** *workers vs* ***Secondary*** *workers (Spot VM)*
- ***Cluster type****: Standard, Single Node (1,0), High-Availability*
- ***Dataproc jobs****: submit & manage Hadoop, Hive, Spark, Pig jobs*
- *Preemptible VM's low-cost worker nodes*
- ***Workflow:*** *Directed Acyclic Graph (DAG) of jobs on a cluster.*
- ***Autoscaling*** *dynamically adjusts cluster size based on workload needs. Set policies to define scaling boundaries, metrics (e.g., YARN memory, CPU utilization), and thresholds for scaling up or down. Configure* ***cooldown periods*** *to avoid rapid, oscillating changes. This provides optimal resource use and cost efficiency throughout the cluster's lifecycle.*
- ***Component Gateway:*** *secure access to web endpoints for Dataproc default and optional components.*
- ***Dataproc Metastore*** *is a fully managed and highly available service that provides a* ***centralized Hive metastore*** *for your data lake. Allows diverse data processing tools and engines, like Spark, Hive, and Presto, to seamlessly share and access metadata, simplifying data discovery, querying, governance, and overall data management. Think of it as a central repository for information about your data: table schemas, locations, partitions, data types, etc.*
- ***Spark History Server*** *offers a web interface to view details about completed Spark jobs, including stages, tasks, executors, RDDs, and environment settings. It's essential for performance tuning and debugging. Dataproc clusters can be set up to automatically upload application history to GCS, allowing you to access and analyze this data even after the cluster has ended.*

Foundation

- ***Persistent Clusters:*** *Long-lived, ideal for ongoing batch jobs, retaining state between jobs. Higher ongoing cost but efficient for frequent use.*
- ***Transient Clusters:*** *Short-lived, created for specific jobs and terminated afterward. Cost-effective for one-off or sporadic batch processing.*
- ***Customize clusters:*** *optional components and initialization actions*
- ***Initialization actions*** *to install additional component on the cluster (e.g., kafka installed at worker creation time)*
- ***Optional Components:*** *Anaconda, Hive WebHCat, Jupyter Notebook, Zeppelin Notebook, Druid, Presto and Zookeeper, Ranger, HBase, Flink, Docker, Solr*
- *Design for* ***short-lived clusters***
- *Jobs can be submitted through Console, gcloud, REST API, orchestration services (Dataproc Workflow and Composer)*

Dataproc integration in GCP Ecosystem

- ***GCS*** *for staging and storing data, providing scalable and durable storage for your Dataproc jobs.*
- ***BigQuery*** *acts as a data warehouse for querying large datasets efficiently, often used in conjunction with Dataproc for analytics.*
- ***Dataflow*** *manages data pipelines for ETL processes, integrating with Dataproc for seamless data processing and transformation.*
- ***Dataproc vs Dataflow?***
    - *Lead with Dataflow for streaming workloads*
    - *Lead with Dataproc if Spark and/or for lift-and-shift*

Security Access Control

- *Project-wide access only*
- *Dataproc Editor: create/delete/edit clusters, jobs, workflows*
- *Dataproc Viewer*
- ***Dataproc Worker****: execute jobs ; assigned to service account*

Best Practices

- ***Cost Optimization:***
  - ***Utilize Spot VMs*** *for non-critical, ephemeral clusters*
    - *Limitations: availability, preemption, no SLA*
  - ***Autoscaling:*** *Dynamically right-size clusters based on workload demands for optimal resource utilization.*
  - ***Larger persistent disks*** *over adding more VMs for high disk throughput & cost-effective scaling*
  - ***Larger persistent disks*** *over adding more VMs for improved disk throughput and cost-effective scaling.*
- ***Performance tuning:***
  - ***Co-locate clusters and buckets in the same region****. This minimizes latency and egress costs.*
  - ***Use SSDs*** *when many shuffling operations or partitioned write workloads to boost performance*
  - *Storing intermediary data on PD with* ***HDFS*** *will offer* ***better I/O performance*** *than using GCS*
  - ***Shuffle-heavy workloads*** *→ increase boot disk size on spot workers*
  - ***Shuffle:*** *redistributes data across executors for operations like sorting, joining, or grouping.*
  - ***spark.sql.shuffle.partitions:*** *sets the number of partitions after a shuffle in Spark SQL*
  - ***spark.default.parallelism:*** *determines the default parallelism level, including initial data partitioning*
  - ***Choose right*** *machine, disk types (SSD vs. standard), and cluster sizing for specific workloads*
  - *Place the* ***workers in the same zon****e to ensure low latency between nodes*
- ***Storage Optimization:***
  - ***Leverage GCS over HDFS*** *(3x data replication overhead) → separating compute and storage*
  - ***Optimize Columnar formats****: Prefer Parquet or ORC*
  - ***I/O optimization:*** *avoid* ***small reads*** *and use* ***large block sizes****. Avoid excessive nested directory traversal*
- ***Use distcp*** *for efficient on-premises to cloud data transfers.*
- ***Monitor & Optimize****: Use Cloud Monitoring for performance tuning and resource optimization.*
- ***Secure*** *with VPC Service Controls, CMEK, and Least Privilege*
- ***Advanced optimizations:*** *aggressive query optimizations (AQE, code, shuffling) potentially leading to significant performance gains. May introduce complexity and instability.*
- ***Advanced execution layer:*** *Spark 3's improved execution engine in Cluster mode resulting in better performance and resource utilization.*
- ***GCS caching:*** *caches data from GCS to accelerate data access. Highly effective for workloads accessing the same data repeatedly, significantly reducing I/O overhead.*

Ephemeral vs Persistent Clusters

Using ***ephemeral clusters*** *for single-job tasks avoids issues with persistent clusters, such as:*

- ***Single Points of Failure****: Errors in a shared persistent cluster can disrupt all jobs and delay recovery. Ephemeral clusters, which are recreated quickly, minimize this risk.*
- ***State Management****: Persistent clusters face challenges in maintaining and migrating cluster states across systems like HDFS or MySQL.*
- ***Resource Contention****: Multiple jobs on a persistent cluster can lead to resource conflicts, impacting performance.*
- ***Unresponsive Daemons****: Memory pressure in persistent clusters can cause service daemons to become unresponsive.*
- ***Disk Capacity Issues****: Persistent clusters can accumulate logs and temporary files, potentially filling up disk space.*
- ***Upscaling Problems****: Persistent clusters may struggle with resource allocation if zones are unavailable.*
- ***Outdated Software****: Managing outdated cluster images can be problematic with persistent setups.*

Migration Considerations

- ***Migrating data*** *to HDFS or GCS (unless heavy I/O.)*
- ***Migrating jobs*** *to Dataproc clusters*
- ***Migrating HBase*** *to Bigtable (good performance)*
- ***Migrating Hive & Impala*** *to BigQuery*
- *Persistent clusters (multiple jobs) to* ***ephemeral clusters***
- ***Separate storage and compute (clusters)***
- *Maturity model for incremental migration stages:*

Monitoring

- ***Job Driver Output****: log level per job. Gathered automatically.*
- ***Logs:*** *default to INFO, can be configured DEBUG*
- ***Cloud Monitoring:*** *HDFS, YARN, Job & Operational metrics*
- ***Cluster Details graphs:*** *CPU, DISK, NETWORK utilizations*

Machine Learning

- ***SparkML*** *is the modern, higher-level library for building scalable ML applications in Spark, offering pipeline support and DataFrame integration.*
- ***MLlib*** *is the original, lower-level library focused on RDDs and is now a legacy option in favor of SparkML. Supports Java, Scala, Python and R*

Web Interfaces (YARN, HDFS, Spark)

- ***Component Gateway***
  - *One-click access to Hadoop, Spark, and other component web UIs through the GCP console*
  - *Requires enabling during cluster creation.*
- ***Google Cloud CLI***
  - *gcloud compute ssh with dynamic port forwarding to create an* ***SSH tunnel*** *and* ***SOCKS proxy.***

Dataproc Languages

Python (PySpark)

- ***Pros****: Easy to learn, large community, rich libraries (Pandas, NumPy), good for data exploration/visualization, prototyping.*
- ***Cons****: Slower than Scala/Java for complex jobs, less type safety can lead to runtime errors.*

Scala (Spark)

- ***Pros****: Fast performance, statically typed (catches errors early), integrates well with Spark, concise syntax.*
- ***Cons****: Steeper learning curve than Python, smaller community.*

Java (Spark)

- ***Pros****: Very fast, mature ecosystem, extensive tooling.*
- ***Cons****: Verbose syntax, can be less developer-friendly*

SQL (Spark SQL)

- ***Pros****: Familiar to many data analysts, declarative (easy to express what you want), good for data warehousing/BI.*
- ***Cons****: Less flexible than general-purpose languages, can be difficult for complex transformations.*

R (SparkR)

- ***Pros****: Powerful statistical computing language, many statistical libraries.*
- ***Cons****: Slower than Scala/Java, less integrated with Spark ecosystem compared to other options.*