BigQuery

BigQuery Overview

Grow Data Skills

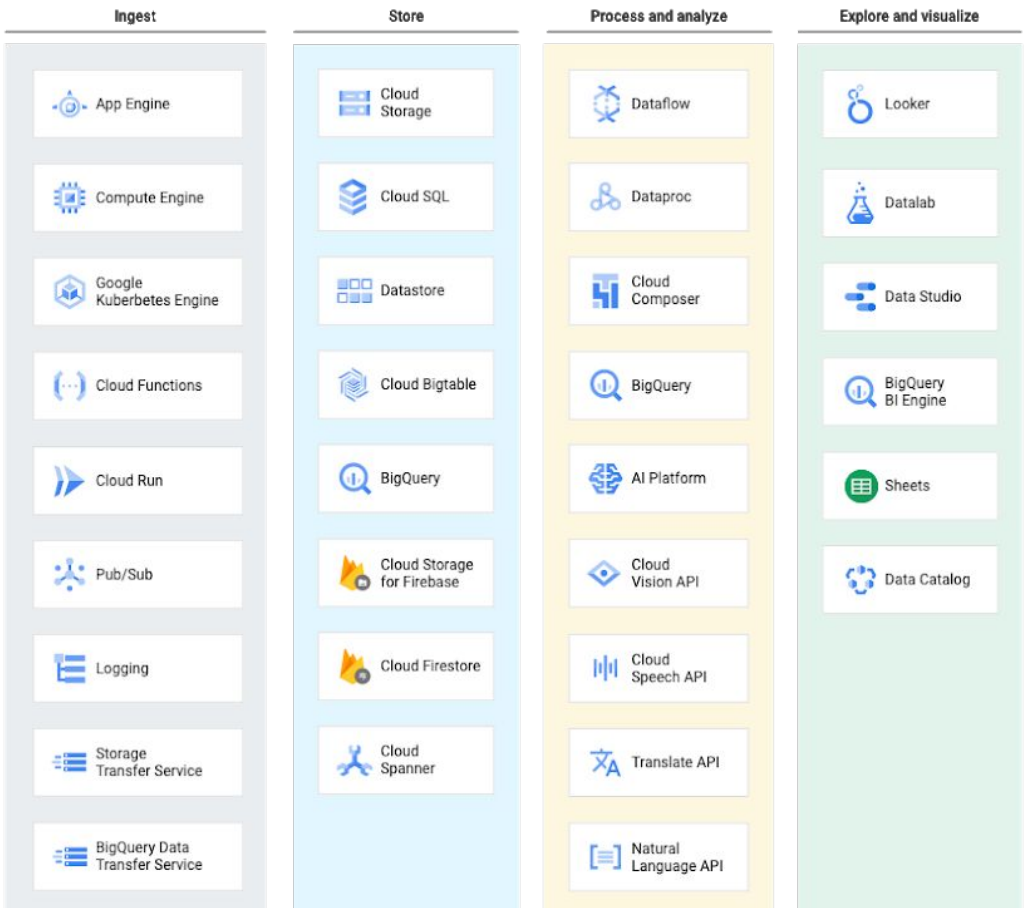
BigQuery is GCP's serverless, highly scalable, and cost effective cloud data warehouse. It allows for super-fast queries at petabyte scale using the processing power of Google's infrastructure. Because there's no infrastructure for customers to manage, they can focus on uncovering meaningful insights using familiar SQL without the need for a database administrator. It's also economical because they pay only for the processing and storage they use.

- **Serverless and Fully Managed:** BigQuery requires no server management or setup, offering a hands-off experience with automatic scaling and performance tuning.
- **Scalability and Performance:** Designed for fast analysis of large datasets, BigQuery provides high-speed SQL query execution and automatic scalability to accommodate data and query demands.
- **Cost-Effectiveness:** Utilizes a pay-as-you-go model based on the data processed and stored, offering a cost-efficient solution for various data workloads.
- **Data Storage and Formats:** Supports massive datasets and a wide range of data formats (like CSV, JSON, Avro), with multiple data loading options including streaming.
- **Integration and Ecosystem:** Seamlessly integrates with other Google Cloud services and supports third-party tools, enhancing its utility in diverse data analytics scenarios.

BigQuery Overview

Grow Data Skills

BigQuery is part of Google Cloud's comprehensive data analytics platform that covers the entire analytics value chain including ingesting, processing, and storing data, followed by advanced analytics and collaboration. BigQuery is deeply integrated with GCP analytical and data processing offerings, allowing customers to set up an enterprise ready cloud-native data warehouse.

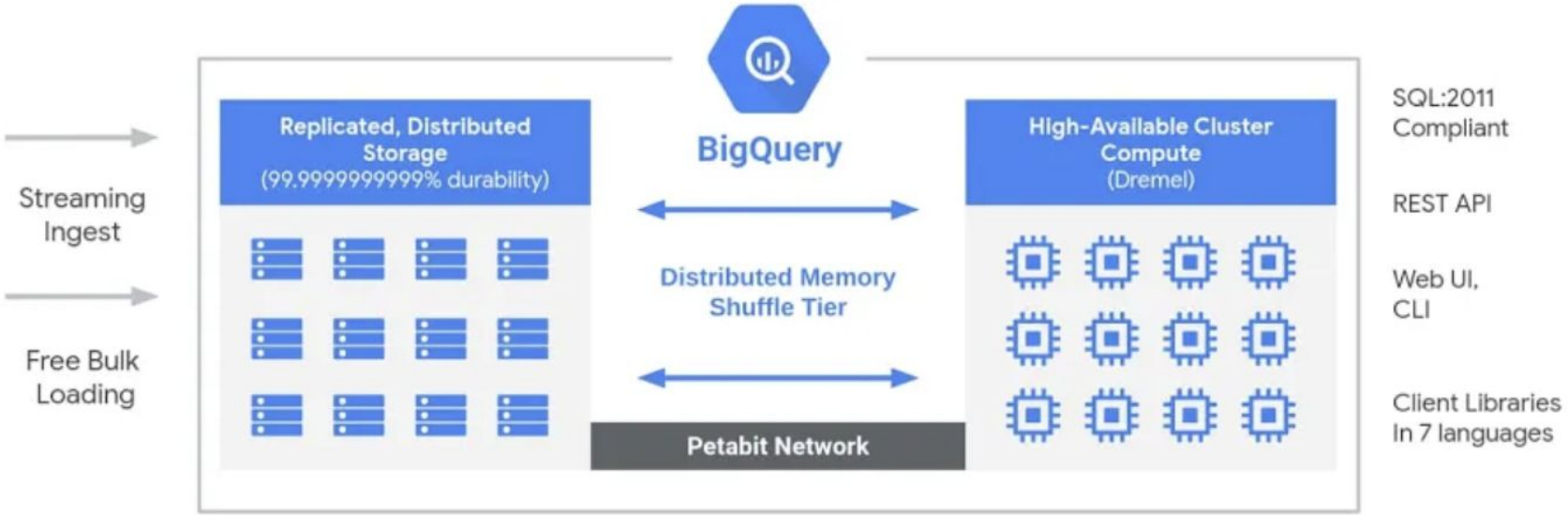


BigQuery Architecture

Grow Data Skills

BigQuery's serverless architecture decouples storage and computing and allows them to scale independently on demand, which offers both immense flexibility and cost controls for customers.

Under the hood, BigQuery employs a vast set of multi-tenant services like **Dremel**, **Colossus**, **Jupiter** and **Borg**.



BigQuery Architecture

Grow Data Skills

- **Capacitor — Columnar format**
  - BigQuery stores data in a columnar storage format called **Capacitor**, which was introduced in 2016. The capacitor allows BQ to store and efficiently query **semi-structured data** with nested and repeated fields.
  - Each column in a BQ table is stored in a separate Capacitor file, which enables **high compression** ratios and fast scan throughput. In contrast to its predecessor, ColumnIO, Capacitor allows BQ to directly operate on compressed data without the need to decompress it on the fly.
  - The capacitor encodes each column by collecting statistics about the data and storing two numbers, called the **definition** and **repetition levels**, in addition to the column's value. These statistics are used during query execution to optimize the query plan and choose the most efficient runtime algorithms. Once all of the column data has been encoded, it is **encrypted** and written to **Colossus**.

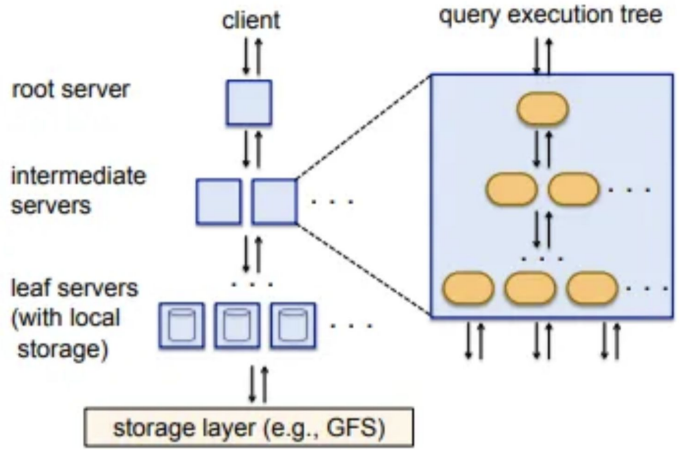
BigQuery Architecture

Grow Data Skills

- **Colossus — Storage**
  - Colossus, Google's latest-generation distributed file system, is designed to be reliable and fault-tolerant.
  - It replaced the previous Google File System (GFS) and is capable of handling cluster-wide replication, recovery from disk crashes, and distributed management.
  - BigQuery uses a model to **evaluate the input data** and determine the **optimal number of shards**, or **partitions**, for the data. Once data is written to Colossus, BigQuery initiates the **geo-replication process**, which mirrors the data across different data centres within a specified jurisdiction to ensure high data availability and query load balancing.

- **Dremel — Execution Engine**
  - Dremel is a scalable, Interactive **ad-hoc query system** for analysis of largescale read-only nested data, developed by Google.
  - It implements a **multi-level serving tree** to scale query execution, which includes a **root node**, **intermediate** and **leaf nodes**.
  - Once a user submits a query, it is received by the **Root node**.

BigQuery Architecture



- **Root Node**
  - Reads metadata from tables.
  - The root server is responsible for communication between the client and mixers.
  - The root node then routes the query to the intermediate nodes.

- **Intermediate Nodes (Mixers)**
  - Performs query **optimization** and **rewrites*** the query to include horizontal partitions of the table (shards), and partial aggregation and filtering.
  - These partitions (shards) are used to parallelize the query execution across **multiple leaf nodes**.
  - Query optimizer, working in conjunction with the mixers, analyzes the query to select the best execution plan, taking into account available resources and opportunities for parallelization and data pruning.
  - Once the mixer has performed its tasks, it routes the optimized and rewritten query to the appropriate leaf nodes for execution.

BigQuery Architecture

- **Leaf Nodes**
  - Perform the heavy lifting of reading the data from Colossus and performing filters and final aggregation.
  - In a typical Dremel tree, there are hundreds or thousands of leaf nodes.
  - Each leaf node provides **execution threads (slots)**, which BQ automatically calculates for each query based on its size and complexity.
  - **Slots**
    - Each leaf node provides an execution thread or the number of processing units often called slots, which BQ automatically calculates for each query based on query size and complexity.
    - For example, a system of 3,000 leaf servers each using 8 threads has 24,000 slots.
    - For the on-demand pricing model, a maximum of 2000 concurrent slots are allowed per BigQuery project.
  - Leaf nodes return results to **Mixers or intermediate nodes**.

BigQuery Architecture

Grow Data Skills

Let’s understand the query execution with a simple example:

- A user submits a query to BigQuery, such as SELECT COUNT(*) FROM mytable WHERE timestamp > ‘2021–01–01’.
- The query is received by the root node of the Dremel serving tree, which is the starting point for query execution.
- The root node routes the query to the intermediate nodes (or mixers) of the serving tree. These nodes perform query optimization and rewrite the query to include horizontal partitions of the table and partial aggregation and filtering.
- In this example, the query optimizer might decide to partition the table based on the timestamp column and only include the partitions that have timestamps greater than ‘2021–01–01’.
- The intermediate nodes then send the rewritten query to the leaf nodes for execution. The leaf nodes read the relevant partitions of the table from Colossus and perform the filters and final aggregation specified in the query.
- In this example, the leaf nodes would count the number of rows in the partitions that have timestamps greater than ‘2021–01–01’ and return the result to the intermediate nodes.
- The intermediate nodes then pass the results back up the serving tree to the root node, which returns the final result of the query, such as “COUNT(*) = 1000000”, to the user.
- The query optimizer ensures that the query can be executed within the constraints of the system and also that it’s the most efficient way to execute the query.

- **Query Rewrite**
  - To parallelize the query, each serving level performs (Root and Mixers) query rewrite, and ultimately modified and partitioned queries reach the leaf nodes for execution.
  - During query rewrite, a few things happen.
    - The query is modified to include horizontal partitions of the table, i.e. shards (in the original Dremel paper shards were referred to as tablets).
    - Certain SQL clauses can be stripped out before sending to leaf nodes.

- **Query Dispatcher**
  - The query dispatcher plays an important role by balancing the load and ensuring that queries are executed even in the event of a failure or a node outage (fault tolerance). It monitors the status of the nodes and can quickly redirect queries to other nodes if a problem is detected.

- **Borg — Compute**
  - Borg is Google's large-scale **cluster management** system. Borg simultaneously runs thousands of Dremel jobs across one or more clusters made up of tens of thousands of machines. In addition to assigning compute capacity for Dremel jobs, Borg handles fault tolerance.

- **Jupiter — Network**
  - BigQuery requires an ultra-fast network which can deliver terabytes of data in seconds directly from storage into compute for running Dremel jobs. Google's Jupiter network enables BigQuery service to utilize **1 Petabit/sec of total bisection bandwidth**.

BigQuery Architecture

Grow Data Skills