GCP Topics

10 February 2023 12:12

Module 1: GCP Introduction

- Why we need Cloud
- Overview of Google Cloud Platform (GCP)
- Key GCP Services and Products
- Understanding Cloud Computing and its Benefits

Module 2: GCP Interfaces

- Console
  - Navigating the GCP Console
  - Configuring the GCP Console for Efficiency
  - Using the GCP Console for Service Management
- Shell
  - Introduction to GCP Shell
  - Command-line Interface (CLI) Basics
  - GCP Shell Commands for Service Deployment and Management
- SDK
  - Overview of GCP Software Development Kits (SDKs)
  - Installing and Configuring SDKs
  - Writing and Executing GCP SDK Commands

Module 3: GCP Locations

- Regions
  - Understanding GCP Regions
  - Selecting Regions for Service Deployment
  - Impact of Region on Service Performance
- Zones
  - Exploring GCP Zones
  - Distributing Resources Across Zones
  - High Availability and Disaster Recovery Considerations
- Importance
  - Significance of Choosing the Right Location
  - Global vs. Regional Resources
  - Factors Influencing Location Decisions

Module 4: GCP IAM & Admin

- Identities
  - Introduction to Identity and Access Management (IAM)
  - Users, Groups, and Service Accounts
  - Best Practices for Identity Management
- Roles
  - GCP IAM Roles Overview
  - Defining Custom Roles

        - Role-Based Access Control (RBAC) Implementation
    - Policy
        - Resource-based Policies
        - Understanding and Implementing Organization Policies
        - Auditing and Monitoring Policies
    - Resource Hierarchy
        - GCP Resource Hierarchy Structure
        - Managing Resources in a Hierarchy
        - Organizational Structure Best Practices

Module 5: GCP Networking

- VPC (Virtual Private Cloud)
- Load Balancer
- Firewalls

Module 6: Compute Options

- Google Compute Engine (GCE)
    - Introduction to GCE and Virtual Machines (VMs)
- Google Kubernetes Engine (GKE)
    - Overview of Kubernetes and Container Orchestration
- Google App Engine (GAE)
    - Understanding the App Engine Platform
- Cloud Run
    - Serverless Container Execution with Cloud Run
- Cloud Functions
    - Serverless Computing with Cloud Functions

GCP Data Engineer Topics

Thursday, November 30, 2023 11:26 AM

https://drive.google.com/file/d/1TA20hDsALX3TfXH4PHA2FRdbr2xvqy96/view?usp=drive_link

Google Cloud Storage

- Introduction to Cloud Storage
  - Overview of Cloud Storage as a scalable and durable object storage service.
  - Understanding buckets and objects in Cloud Storage.
  - Use cases for Cloud Storage, such as data backup, multimedia storage, and website content delivery.
- Cloud Storage Operations
  - Creating and managing Cloud Storage buckets.
  - Uploading and downloading objects to and from Cloud Storage.
  - Setting access controls and permissions for buckets and objects.
- Data Transfer and Lifecycle Management
  - Strategies for efficient data transfer to and from Cloud Storage.
  - Implementing data lifecycle policies for automatic object deletion or archival.
  - Utilizing Transfer Service for large-scale data transfers.
- Versioning and Object Versioning
  - Enabling and managing versioning for Cloud Storage buckets.
  - Understanding how object versioning works.
  - Use cases for object versioning in data resilience and recovery.
- Integration with Other GCP Services
  - Integrating Cloud Storage with BigQuery for data analytics.
  - Using Cloud Storage as a data source for Dataflow and Dataproc.
  - Exploring options for serving static content on websites.
- Best Practices and Security
  - Implementing best practices for optimizing Cloud Storage performance.
  - Securing data in Cloud Storage with encryption and access controls.
  - Monitoring and logging for Cloud Storage operations.

Cloud SQL

- Introduction to Cloud SQL
  - Overview of Cloud SQL as a fully managed relational database service.
  - Supported database engines and use cases for Cloud SQL.
- Creating and Managing Cloud SQL Instances
  - Creating MySQL or PostgreSQL instances.
  - Configuring database settings, users, and access controls.
  - Importing and exporting data in Cloud SQL.
- Backups and High Availability
  - Configuring automated backups and performing manual backups.
  - Implementing high availability with failover replicas.
  - Strategies for restoring data from backups.
- Scaling and Performance Optimization
  - Vertical and horizontal scaling options in Cloud SQL.
  - Performance optimization tips for database queries.
  - Monitoring and troubleshooting database performance.
- Integration with Other GCP Services
  - Connecting Cloud SQL with App Engine, Compute Engine, and Kubernetes Engine.
  - Using Cloud SQL as a backend database for applications.
  - Data synchronization with Cloud Storage and BigQuery.

Bigtable

- Introduction to Bigtable
  - Overview of Bigtable as a fully managed NoSQL wide-column store.
  - Use cases for Bigtable in real-time analytics and IoT applications.

BigQuery

- Introduction to BigQuery
  - Overview of BigQuery as a fully managed, serverless data warehouse.
  - Use cases for BigQuery in business intelligence and analytics.
- SQL Queries and Performance Optimization
  - Writing and optimizing SQL queries in BigQuery.
  - Understanding query execution plans and best practices.
  - Partitioning and clustering tables for performance.
- Data Integration and Export
  - Loading data into BigQuery from Cloud Storage, Cloud SQL, and other sources.
  - Exporting data from BigQuery to various formats.
  - Real-time data streaming into BigQuery.
- Access Controls and Security

- Configuring access controls and permissions in BigQuery.
  - Implementing encryption for data in BigQuery.
  - Auditing and monitoring for security compliance.
- Integration with Other GCP Services
  - Integrating BigQuery with Dataflow for ETL processes.
  - Using BigQuery in conjunction with Data Studio for visualization.
  - Building data pipelines with BigQuery and Composer.

DataProc

- Introduction to DataProc
  - Overview of DataProc as a fully managed Apache Spark and Hadoop service.
  - Use cases for DataProc in data processing and analytics.
- Cluster Creation and Configuration
  - Creating and managing DataProc clusters.
  - Configuring cluster properties for performance and scalability.
  - Preemptible instances and cost optimization.
- Running Jobs on DataProc
  - Submitting and monitoring Spark and Hadoop jobs on DataProc.
  - Use of initialization actions and custom scripts.
  - Job debugging and troubleshooting.
- Integration with Storage and BigQuery
  - Reading and writing data from/to Cloud Storage and BigQuery.
  - Integrating DataProc with other storage solutions.
  - Performance optimization for data access.
- Security and Access Controls
  - Configuring access controls for DataProc clusters.
  - Implementing encryption for data at rest and in transit.
  - Managing security configurations for DataProc.
- Scaling and Automation
  - Autoscaling DataProc clusters based on workload.
  - Using Dataprep or other tools for data preparation before processing.
  - Automation and scheduling of recurring jobs.

DataFlow

- Introduction to DataFlow
  - Overview of DataFlow as a fully managed stream and batch processing service.
  - Use cases for DataFlow in real-time analytics and ETL.
- Building Data Pipelines with Apache Beam
  - Writing Apache Beam pipelines for batch and stream processing.
  - Transformations and windowing concepts.
  - Error handling and testing of DataFlow pipelines.
- Monitoring and Optimization
  - Monitoring and troubleshooting DataFlow pipelines.
  - Optimizing pipeline performance and resource utilization.
  - Utilizing DataFlow templates for reusable pipelines.
- Integration with Other GCP Services
  - Integrating DataFlow with BigQuery, Pub/Sub, and other GCP services.
  - Real-time analytics and visualization using DataFlow and BigQuery.
  - Workflow orchestration with Composer.

Cloud Pub/Sub

- Introduction to Pub/Sub
  - Understanding the role of Pub/Sub in event-driven architectures.
  - Key Pub/Sub concepts: topics, subscriptions, messages, and acknowledgments.
- Creating and Managing Topics and Subscriptions
  - Using the GCP Console to create Pub/Sub topics and subscriptions.
  - Configuring message retention policies and acknowledgment settings.
- Publishing and Consuming Messages
  - Writing and deploying code to publish messages to a topic.
  - Implementing subscribers to consume and process messages from subscriptions.
- Integration with Other GCP Services
  - Connecting Pub/Sub with Cloud Functions for serverless event-driven computing.
  - Integrating Pub/Sub with Dataflow for real-time stream processing.
- Monitoring and Logging
  - Setting up monitoring and logging for Pub/Sub.
  - Analyzing metrics and logs to troubleshoot and optimize message processing.

Cloud Composer/Airflow

- Introduction to Composer
  - Overview of Composer as a fully managed workflow orchestration service.
  - Use cases for Composer in managing and scheduling workflows.
- Creating and Managing Workflows
  - Creating and configuring Composer environments.
  - Defining and scheduling workflows using Apache Airflow.

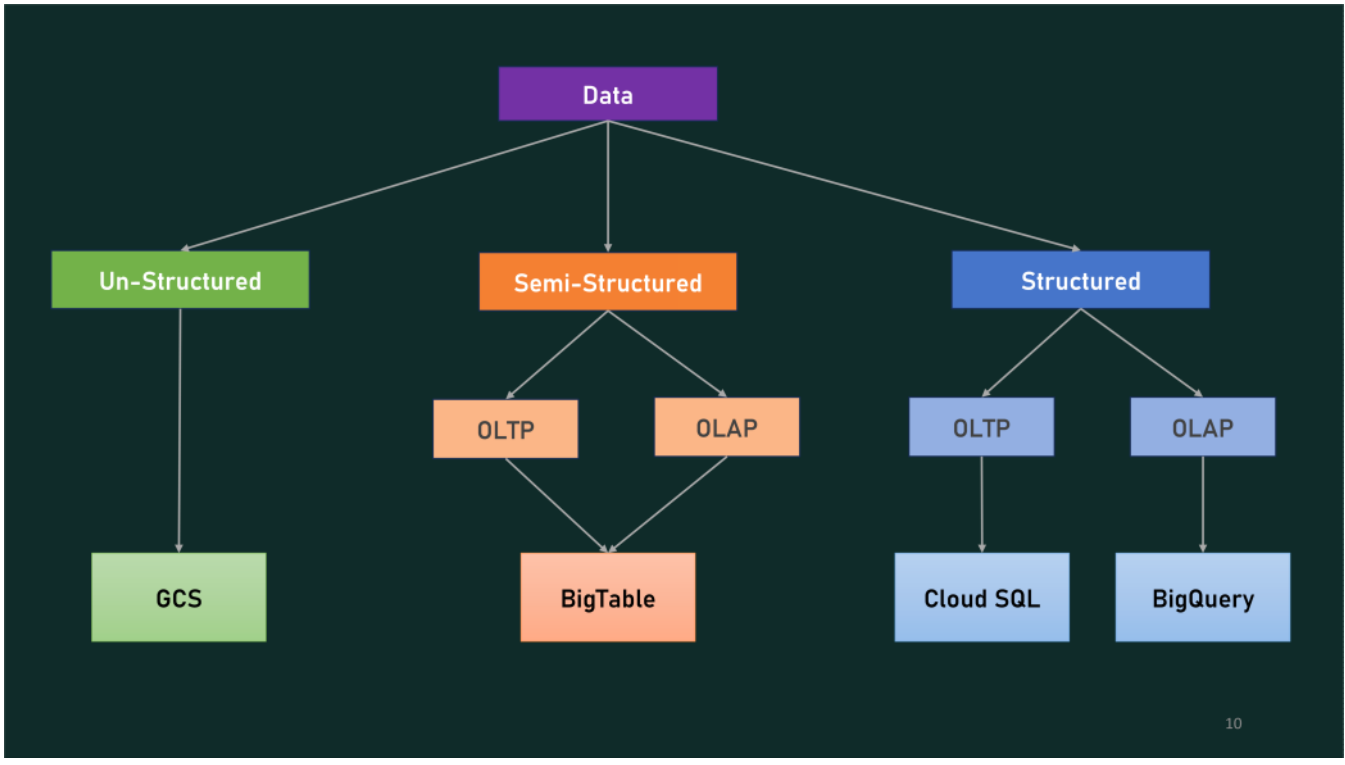
        - Monitoring and managing workflow executions.
    - Integration with Data Engineering Services
        - Orchestrating workflows involving BigQuery, DataFlow, and other services.
        - Coordinating ETL processes with Composer.
        - Integrating with external systems and APIs.
    - Extending and Customizing Composer
        - Extending Apache Airflow with custom operators and sensors.
        - Creating and managing Composer plugins.
        - Versioning and managing workflow code.
    - Security and Access Controls
        - Configuring access controls for Composer environments.
        - Implementing encryption for data and workflow metadata.
        - Best practices for securing Composer workflows.

Data Fusion

- Introduction to Data Fusion
    - Overview of Data Fusion as a fully managed data integration service.
    - Use cases for Data Fusion in ETL and data migration.
- Building Data Integration Pipelines
    - Creating ETL pipelines using the visual interface.
    - Configuring data sources, transformations, and sinks.
    - Using pre-built templates for common integration scenarios.

Terraform

- Introduction to Infrastructure as Code (IaC) and Terraform
    - Understanding IaC principles and benefits.
    - Overview of Terraform as a declarative infrastructure provisioning tool.
- Terraform Basics
    - Installing and configuring Terraform.
    - Writing Terraform configurations using HashiCorp Configuration Language (HCL).
    - Initializing and applying Terraform configurations.
- Infrastructure Provisioning
    - Creating and managing infrastructure resources using Terraform.
    - Terraform state and remote backends.
    - Importing existing infrastructure into Terraform.
- Module and Provider Usage
    - Organizing Terraform configurations using modules.
    - Utilizing different providers for various cloud services.
    - Best practices for reusable and modular Terraform code.
- Variables, Outputs, and Functions
    - Defining and using variables in Terraform.
    - Outputting values from Terraform configurations.
    - Utilizing functions for dynamic configurations.
- Terraform Workflow and Best Practices
    - Terraform workflows: plan, apply, and destroy.
    - Managing Terraform environments and workspaces.
    - Best practices for writing maintainable and scalable Terraform code.


Data
Un-Structured
Semi-Structured
Structured
OLTP
OLAP
OLTP
OLAP
GCS
BigTable
Cloud SQL
BigQuery
10

Pre-requisites

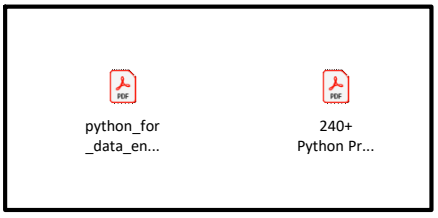
Thursday, November 30, 2023 11:47 AM

SQL Concepts :

1. data types (int, float, bool, datetime, string)
2. create/insert/update/drop/delete/alter/select/truncate commands
3. from/where/having clause
4. order by/group by
5. union and union all
6. Aggregate functions: count, min, max, avg, sum
7. ranking functions: row number, rank, dense rank
8. Joins : inner, left, right, full
9. top, limit, like, and, or, null
10. sub queries

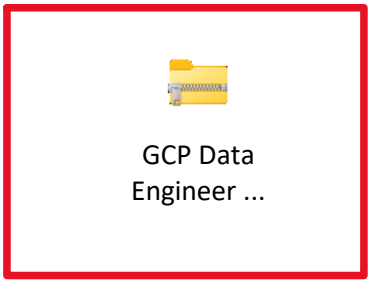
Python Concepts :

1. datatypes (int, string, float, Boolean, array, list, tuple, dict, range)
2. variables, operators, string formatting, casting
3. conditions(if, if else)
4. functions
5. loops(for, while)



Resume Templates

Wednesday, December 13, 2023 8:47 AM



Interview Introduction

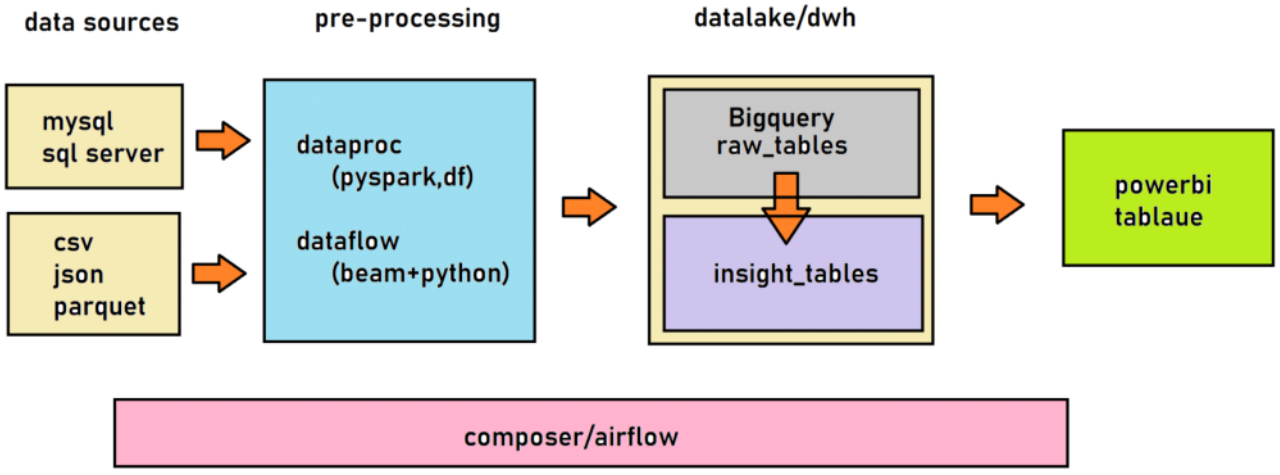
Tuesday, December 19, 2023 8:27 AM

Introduction :

- Hi name, good morning. thanks for connecting with me.
- My name is "name",
- i have completed my B.Tech/MTech from "college name" in year
- i have total "2/3/4/8" years of experience as data engineer,
- from last "2/3/" years i have been working in google cloud projects

Project - 1 :

- I am working a BigQuery lake project, where we get the data from multiple data sources such as SQL server, mysql and sometimes csv, Json , parquet in gcs.
- we extract the data from source using DataProc(pyspark)/dataflow(Apache beam python), on top this data are also performing transformations(such as duplicate removal, null handling, column naming, aggregations, joining, merging) and finally loading it into BigQuery as raw tables.
- as per the business use case requirements, we also perform analysis and create as insight table in BigQuery, these insight tables are very useful for downstream for creating dashboards and reports in BI tools.
- for scheduling and managing all these pipelines and refreshing the tables(BigQuery) we are using cloud composer as a schedular.
- Coming to languages I have good experience in SQL and pyspark



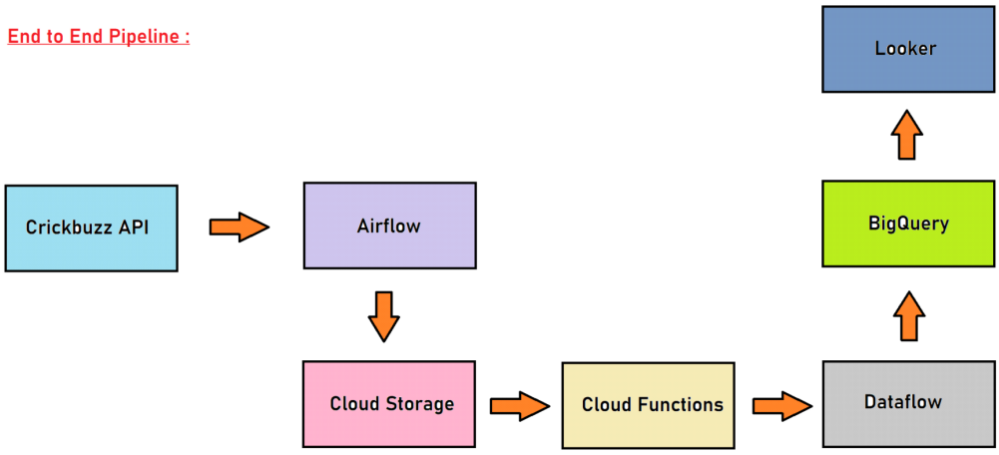
Project -2 :

- I am working on google cloud projects, In this project, we have implemented a data pipeline to ingest data from an Application API using a Python script. This script is scheduled to run as a part of DAG in Cloud Composer. Once the Python script retrieves the data, it is stored in Google Cloud Storage as a CSV file.

- after creation of the CSV file in the Cloud Storage bucket, a Cloud Function is triggered. This Cloud Function automatically submits a beam job to Dataflow, which is configured to load the data from Cloud Storage into BigQuery.

- on top of the BigQuery table, we have created dashboards using Looker, These dashboards provide stakeholders with interactive visualizations, enabling them to derive valuable insights and make informed decisions based on the processed data.

End to End Pipeline :



Data Engineering Questions

Wednesday, November 29, 2023 3:46 PM

What are the services that you have worked in GCP ?

- GCS, BigQuery, DataProc, dataflow, composer, cloud SQL, pub sub, Cloud Function

**** Google Cloud Storage ****

What is Cloud Storage(GCS)

- Cloud Storage (GCS), also known as Google Cloud Storage, is a scalable object storage service
- It allows users to store and retrieve data in a highly available and globally distributed manner.
- GCS is designed to handle large amounts of data and is suitable for various use cases, including data storage, backup, and serving static content for websites.

What are the storage classes in GCS

- GCS provides 4storage classes to cater to different performance and cost requirements
  - Standard Storage: Suitable for frequently accessed data with low-latency requirements.
  - Nearline Storage: Intended for data that is accessed less frequently, with a minimum storage duration.
  - Cold line Storage: Designed for archival data that is accessed very infrequently, with a minimum storage duration.
  - Archive Storage: Ideal for long-term archival data with the lowest storage cost but the highest access cost.

How to control the access to buckets and objects in GCS

- Access to GCS resources (buckets and objects) is controlled through a combination of Identity and Access Management (IAM) policies, access control lists (ACLs), and signed URLs. IAM policies allow users to grant specific permissions to users or groups, while ACLs provide fine-grained control over individual objects.
- Users can define who can access their GCS resources and what actions they are allowed to perform, such as reading, writing, or deleting objects.

What are life cycle rules in GCS

- Lifecycle management in GCS allows users to automate the process of managing objects over time. This includes transitioning objects between storage classes or deleting them based on specified conditions. Lifecycle rules are defined at the bucket level and can be used to:
  - Automatically transition objects to a different storage class after a specified period.
  - Delete objects that meet certain criteria, such as objects older than a certain age.
- Lifecycle rules help optimize storage costs by automating the management of data based on its lifecycle.

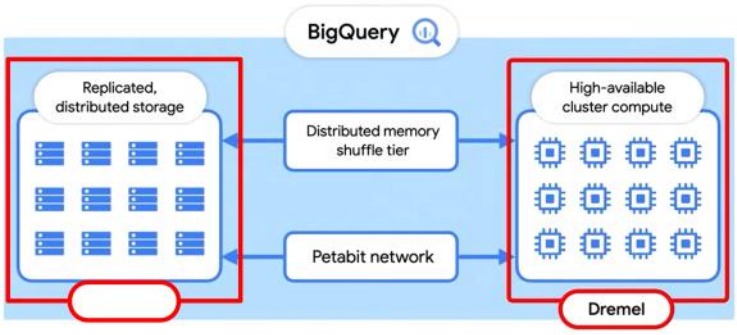
**** BigQuery ****

What is BigQuery ?

BigQuery is a fully-managed, serverless data warehouse to store, transform, process and analyze the data.

What is the Architecture of BigQuery?

- BigQuery has 3 main components : CDFS, Jupiter and Dremel
- BigQuery has a multi-layered architecture with a storage layer that holds the data in a columnar format



What are the different types of file formats supported in BigQuery?

- BigQuery supports various file formats, including CSV, JSON, Avro, Parquet, and ORC.

What is Materialized and Generic views and Authorized Views in BigQuery?

- Materialized views in BigQuery store the results of a query, improving query performance.
- Generic views are virtual and don't store data, offering dynamic querying capabilities.
- authorized view in BigQuery is a view with defined row-level access control policies. It allows you to restrict data access based on user or group identity.

What are the Optimization techniques in BigQuery?

BigQuery optimization involves partitioning tables, clustering tables based on columns, using appropriate data types, and optimizing SQL queries for better performance.

What is Partitioning and Clustering and benefits ?

In BigQuery, partitioning and clustering are strategies used to organize and optimize the storage and query performance of large datasets.

Partitioning:

- Partitioning involves dividing a large table into smaller, more manageable and easily query able segments based on a chosen column or set of columns.

Benefits:

- Improved Query Performance
- Reduced Costs
- Reduce time taken for querying

Limitations:

- Only one column can partition column per table
- Can be : integer range, data time and ingestion only
- Cannot be : done one string and float

Clustering:

Clustering involves organizing the data within each partition based on the values of one or more columns. The data is physically sorted and stored together on disk, improving the efficiency of query processing.

Benefits:

- Reduced Data Skew
- Improved Query Performance

Limitations:

- Overhead during Writes
- Max 4 columns
- Cannot be done on float column

What are the different ways of loading data in BigQuery Table?

- Data can be loaded into BigQuery through batch loading using tools like bq command-line tool (shell, sdk, console) and streaming for real-time data, or via various data transfer services.

What is the difference between Row level and columnar base Datawarehouse?

- BigQuery is columnar-based, providing advantages in terms of query speed and cost efficiency.
- It differs from traditional row-level data warehouses in its storage and processing model.

What is the Difference between Redshift and BigQuery?

- While both are cloud-based data warehouses, Redshift is part of AWS, and BigQuery is part of GCP. Redshift uses a traditional cluster-based model, while BigQuery is serverless and fully managed.

What is the Difference between Hive and BigQuery?

- Hive is a Hadoop-based data warehousing solution, whereas BigQuery is a fully managed, serverless data warehouse. BigQuery offers better performance and scalability with its architecture.

What are the different data types supported by BigQuery?

- BigQuery supports various data types, including INTEGER, FLOAT, STRING, BOOLEAN, TIMESTAMP, DATE, ARRAY, STRUCT, and more.

How can you manage accesses and controls in BigQuery tables?

- Access controls in BigQuery are managed through Identity and Access Management (IAM). You can assign roles like OWNER, READER, or WRITER to control access at the dataset and table levels.

How does BigQuery stores the data in the backend?

- BigQuery stores data in Capacitor, a highly compressed columnar storage format. This enables fast and cost-effective querying by reading only the required columns.

What is the concept of federated queries in BigQuery?

- Federated queries in BigQuery allow you to query data across different data sources, such as BigQuery tables, Cloud SQL, or external data like CSV files in GCS.

How to retrieve deleted Table in BigQuery?

- If the table was deleted within the last seven days, you can use BigQuery's Time Travel feature to access and query past versions of the table.

****** Dataproc *****

What have you done In Dataproc ?

- Dataproc is a cluster based tool,
- we have created pyspark jobs to read the data from google cloud storage(csv, Json) and create data frame and do the transformations such as aggregation, joins, duplicate removal, ranking functions and finally loaded the data to BigQuery

What is Google Cloud Dataproc?

Answer: Google Cloud Dataproc is a fast, easy-to-use, fully managed cloud service for running Apache Spark and Apache Hadoop clusters. It allows you to process large datasets quickly and cost-effectively.

How is Dataproc different from other Hadoop/Spark solutions?

Answer: Dataproc provides a fully managed environment, which means you don't need to worry about cluster management. It is designed to be quick to set up and scale, providing a cost-effective solution for processing big data.

What are some use cases for Google Cloud Dataproc?

Answer: Dataproc is suitable for various big data processing tasks, including batch processing, machine learning, graph processing, and interactive querying.

How can you create a Dataproc cluster using the Google Cloud Console?

Answer: To create a Dataproc cluster, you can navigate to the Google Cloud Console, go to the Dataproc section, and click "Create Cluster." You need to provide details such as cluster name, region, number of worker nodes, and the type of machine to be used.

What is the significance of initialization actions in Dataproc?

Answer: Initialization actions are scripts that run on all nodes in the cluster when it starts. They allow you to customize the environment by installing additional software, setting configuration files, or executing other setup tasks.

How can you submit a Spark job to a Dataproc cluster?
Answer: You can submit a Spark job to a Dataproc cluster using the gcloud command-line tool or the Dataproc API. For example, you can use the following command:

```
gcloud Dataproc jobs submit spark --cluster my-cluster --class com.example.MySparkJob --jars my-job.jar --args arg1,arg2
```

How does autoscaling work in Dataproc?
Answer: Dataproc supports automatic scaling, which adjusts the number of worker nodes in a cluster based on the load. You can enable autoscaling when creating a cluster, and Dataproc will add or remove worker nodes dynamically.

How does Dataproc handle data storage?
Answer: Dataproc provides options for data storage, including Cloud Storage and HDFS. You can store input and output data in Cloud Storage, and Dataproc clusters can also use HDFS for temporary data storage.

What is the role of a master node in a Dataproc cluster?
Answer: The master node in a Dataproc cluster is responsible for managing the overall execution of jobs. It coordinates with worker nodes and ensures that tasks are distributed and executed properly.

**** Dataflow ****

What have you done In dataflow ?

- Dataflow is serverless
- we've created beam pipelines using Apache Beam with python
- Our tasks included reading data from different sources, such as Pub/Sub, Cloud Storage performing various transformations like mapping, filtering, aggregating, and enriching the data, and finally writing the processed data to bigquery

Dataflow and transformations :

Google Cloud Dataflow is a powerful stream and batch data processing service that allows you to perform transformations on large datasets using parallel processing. Dataflow provides a programming model based on transformations and collections of data.

Here are some common transformations you can perform in Google Cloud Dataflow:

- ParDo Transformation: This transformation allows you to apply a custom user-defined function (DoFn) to each element in the input collection and produce zero or more output elements. This is useful for tasks like filtering, extracting, or transforming data.
- Map Transformation: Similar to ParDo, the Map transformation applies a function to each element in the input collection and produces a new collection of transformed elements.
- Filter Transformation: This transformation allows you to filter the elements of a collection based on a condition. It produces a new collection containing only the elements that satisfy the condition.
- GroupByKey Transformation: This transformation groups the input collection's elements by a key and produces a new collection of key-value pairs where each key is associated with a collection of values.
- CombineByKey Transformation: Similar to GroupByKey, this transformation groups elements by key but allows you to apply a user-defined function to combine the values associated with each key. It's used for aggregation operations.
- Flatten Transformation: This transformation combines multiple input collections into a single output collection by concatenating the elements from all input collections.
- Windowing: Dataflow supports windowed processing, where data is divided into time-based windows. You can perform transformations on data within these windows, allowing you to analyze time-based patterns.
- CoGroupByKey Transformation: This transformation is used to combine two or more input collections based on a common key, producing a collection of key-value pairs where each key is associated with a collection of values from all

input collections.

- Side Inputs: Side inputs allow you to provide additional data to a transformation. For example, you can use side inputs to enrich data during processing.

- Partitioning and Shuffling: Dataflow automatically handles partitioning and shuffling of data across workers to distribute processing and ensure parallelism.

**** Cloud Composer ****

What is Cloud Composer?

Cloud Composer is a fully managed workflow orchestration service on Google Cloud Platform (GCP) that allows you to author, schedule, and monitor workflows using Apache Airflow.

2**Explain the key components of Cloud Composer.**

The key components include:

- DAGs (Directed Acyclic Graphs): Workflows defined as directed acyclic graphs.
- Composer Environment: A GCP environment where Airflow services are deployed.
- Airflow Web Server: Provides a web-based UI for Airflow administration.
- Airflow Scheduler: Manages the scheduling and triggering of DAGs.
- Airflow Workers: Execute the tasks defined in DAGs.

3**What is the role of an Operator in Apache Airflow?**

- An Operator defines a single task in a workflow, and it determines what gets executed. Operators represent a unit of work.

How does Cloud Composer handle scalability and reliability?

- Cloud Composer is fully managed, so it automatically scales based on the demand. It uses GCP infrastructure, ensuring reliability, and it supports features like auto healing.

Explain the purpose of a Composer Environment.

- The Composer Environment is a GCP environment where Cloud Composer deploys Airflow components. It includes Cloud Storage and Cloud SQL for storing metadata and DAGs.

What is the significance of a DAG in Cloud Composer?

- A DAG (Directed Acyclic Graph) defines the workflow in Cloud Composer. It specifies the tasks to be executed, their order, and any dependencies between them.

How can you trigger a DAG in Cloud Composer?

- DAGs in Cloud Composer can be triggered manually, based on a schedule, or using external triggers.

Explain the use of XComs in Apache Airflow.

- XComs (Cross-Communication) are a way for tasks to exchange small amounts of metadata.
- They allow tasks to communicate and share information.

What is the Airflow Web Server used for in Cloud Composer?

- The Airflow Web Server provides a web-based UI for interacting with and monitoring Airflow DAGs. Users can view DAG runs, check logs, and trigger DAGs.

How do you monitor and troubleshoot workflows in Cloud Composer?

- Cloud Composer integrates with Stackdriver Logging and Monitoring for monitoring and troubleshooting. You can view logs and metrics in the GCP Console.

C**ron Job format for scheduling in Airflow?**

**** Cloud PubSub ****

What is Google Cloud Pub/Sub?

- Google Cloud Pub/Sub is a messaging service that allows the sending and receiving of messages between independent applications. It provides a scalable and reliable platform for building event-driven systems.

Explain the key components of Google Cloud Pub/Sub.

- Key components include:
  - Topics: Channels for publishing messages.
  - Subscriptions: Named resources representing the stream of messages from a single, specific topic, to be delivered to the subscribing application.
  - Messages: Payloads of data sent from publishers to subscribers.

How does Pub/Sub ensure reliability and scalability?

- Pub/Sub is designed to be globally distributed and highly available.
- It scales automatically to handle large amounts of data and provides at-least-once delivery guarantees.

What are the two modes of message delivery in Pub/Sub?

- Pub/Sub supports both pull and push delivery modes.
- In pull mode, subscribers actively request messages.
- In push mode, messages are automatically pushed to the subscribers.

How does Pub/Sub handle acknowledgment of messages?

- Subscribers acknowledge the messages they receive.
- Pub/Sub ensures that a message is delivered at least once to a subscriber before acknowledging it.

How can you control access to Pub/Sub resources?

- Access to Pub/Sub resources is controlled through Identity and Access Management (IAM) policies, allowing fine-grained control over who can publish or subscribe to topics.

What is the use of Dead Letter Queues in Pub/Sub?

- Dead Letter Queues (DLQ) allow you to capture messages that cannot be successfully processed after a certain number of delivery attempts. They help in handling undeliverable messages.

How does Pub/Sub integrate with other Google Cloud services?

- Pub/Sub integrates with other Google Cloud services like Cloud Functions, Cloud Run, and Dataflow, allowing you to build event-driven and real-time data processing workflows.

Q: How do you handle data ingestion in Google Cloud Platform?
A: I leverage Google Cloud Dataflow for streaming ingestion and Cloud Storage for batch ingestion, ensuring reliable and scalable data processing.

Q: How do you ensure data quality and integrity in a GCP data pipeline?
A: I implement data validation techniques within Dataflow or Cloud Dataprep to detect anomalies, coupled with monitoring using Stackdriver for real-time alerts on pipeline health.

Q: Can you explain your approach to optimizing BigQuery performance?
A: I utilize partitioning and clustering strategies to minimize query costs and enhance query speed, while also optimizing schema design to reduce unnecessary data shuffling.

Q: How do you manage sensitive data in GCP?
A: I implement Cloud Key Management Service (KMS) for encryption at rest and in transit, along with IAM policies to restrict access, ensuring compliance with security standards like GDPR and HIPAA.

Q: What is your experience with orchestrating data pipelines in GCP?
A: I utilize Apache Airflow on Google Cloud Composer for workflow orchestration, enabling automation, monitoring, and dependency management across complex data pipelines.

Q: How do you handle schema evolution in your data pipelines?
A: I employ Avro or Protocol Buffers for schema evolution, ensuring backward compatibility, and utilize schema registry tools like Google Cloud Schema Registry for centralized schema management.

Q: Can you explain how you would design a real-time analytics solution on GCP?
A: I would use Google Cloud Pub/Sub for data ingestion, Cloud Dataflow for real-time processing, and BigQuery for analytics, enabling scalable and low-latency insights.

Q: How would you approach optimizing costs in a GCP data engineering project?
A: I would use tools like Google Cloud Cost Management to monitor resource usage, leverage serverless offerings like BigQuery and Dataflow to scale dynamically, and implement cost-saving strategies such as data lifecycle management to reduce storage costs.

Q: What methods do you employ for data transformation and cleansing in GCP?
A: I use Cloud Dataprep for visual data preparation, transformation, and cleaning tasks, and leverage Dataflow for complex transformations, ensuring data consistency and quality.

Q: How do you ensure high availability and fault tolerance in your data pipelines?
A: I design data pipelines with redundancy using Google Cloud's regional or multi-regional configurations, implement retries and error handling within Dataflow, and leverage features like Cloud Pub/Sub's message durability for fault tolerance.

Q: How do you handle data lineage and metadata management in GCP?
A: I utilize Google Cloud Data Catalog to capture metadata and lineage information, providing a comprehensive view of data assets and their relationships within the ecosystem.

Q: Can you describe a scenario where you optimized a GCP data pipeline for performance?
A: I improved pipeline performance by optimizing SQL queries in BigQuery, implementing streaming updates instead of batch processing where applicable, and scaling resources dynamically based on workload demands.

Q: How do you ensure data security and compliance in GCP?
A: I enforce encryption at rest and in transit using GCP Key Management Service (KMS) and ensure compliance with regulations such as GDPR and CCPA by implementing fine-grained access controls and audit logging.

Q: What strategies do you employ for data versioning and rollback in GCP?
A: I utilize version control systems like Git for code versioning and store datasets in versioned Cloud Storage buckets or BigQuery tables, enabling easy rollback to previous states if necessary.

Q: Can you explain how you would design a data lake architecture in GCP?
A: I would use Cloud Storage as a scalable and cost-effective storage layer, ingest data using tools like Cloud Dataflow or Apache Beam, catalog and manage metadata with Data Catalog, and enable analytics with services like BigQuery and Dataproc for batch processing or Dataflow for stream processing.

Q: How do you ensure data reliability and consistency in distributed data processing environments?
A: I implement idempotent processing and transactional semantics where applicable, use Apache Beam's windowing and triggering mechanisms for event-time processing, and employ Apache Kafka or Google Cloud Pub/Sub for message replayability and fault tolerance.

Q: What monitoring and alerting tools do you use to ensure the health and performance of your GCP data pipelines?
A: I utilize Stackdriver Monitoring for real-time monitoring of pipeline metrics, Stackdriver Logging for centralized log management, and create custom dashboards and alerts to proactively identify and address performance bottlenecks.

Q: How do you handle data skew and hotspots in your data processing workflows?
A: I use techniques such as key sharding and bucketing to evenly distribute data across processing nodes, implement dynamic scaling in Dataflow to adjust resources based on workload patterns, and optimize windowing strategies to mitigate skew.

Q: How do you handle schema evolution in streaming data pipelines?
A: I design schemas to be backward compatible, using techniques such as schema versioning and evolution rules, and employ tools like Apache Avro or Protobuf to serialize data with schema information, enabling seamless updates without disrupting downstream consumers.

SQL Questions

Wednesday, November 29, 2023 3:47 PM

Joins question : table-A and table-B find the count of each join and explain

Table A:

c1
1
2
null
null
1

Table B:

c1
1
1
1
3
null
null

Inner Join : 6

```
a.c1 | b.c1
-----|-----
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
```

Left join : 9

```
a.c1 | b.c1
-----|-----
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
2    | null
null | null
null | null
```

Right join : 9

```
a.c1 | b.c1
-----|-----
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
null| null
null | 3
null | null
```

Full join : 12

```
a.c1 | b.c1
-----|-----
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
1    | 1
2    | null
null | 3
null | null
null | null
null | null
null | null
```

5th highest salary in each deprtment :

```
SELECT department, salary as fifth_highest_salary
FROM (
 SELECT
     department,
     salary,
     ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) as salary_rank
  FROM employees
) ranked_employees
WHERE salary_rank = 5
```

To retrieve employees whose salary is greater than their manager's salary :

```
SELECT e.empid, e.salary
FROM employees e
JOIN employees m ON e.mng_eid = m.empid
WHERE e.salary > m.salary;
```

To get the elements that are in list a but not in list b :

```
SELECT value
FROM table_a
WHERE value NOT IN (SELECT value FROM table_b);
```

query to find who have top2 salries in dept :

```
EmpId EmpName Dept     Salary
1     Jack    HR     10000
2.    Ryan    Finance 8000
3.    Smith   HR     12000
4.    Michael HR     9000
5.    Todd    Finance 9000
6.    John    Finance 13000
```

```
WITH RankedSalaries AS (
    SELECT
        EmpId,
        EmpName,
        Dept,
        Salary,
        RANK() OVER (PARTITION BY Dept ORDER BY Salary DESC) AS SalaryRank
    FROM employees
)
SELECT EmpId, EmpName, Dept, Salary
```

```
FROM RankedSalaries
WHERE SalaryRank <= 2;
```

To retrieve a list of customers who have bought all products:

```
SELECT c.customer_id, c.customer_name
FROM customer c
WHERE NOT EXISTS (
    SELECT p.product_id
    FROM product p
    WHERE NOT EXISTS (
        SELECT s.customer_id
        FROM sales s
        WHERE s.customer_id = c.customer_id AND s.product_id = p.product_id
    )
);
```

SCD (Slowly Changing Dimension) types refer to strategies used in data warehousing to manage changes to dimension tables over time. These changes can involve updating, inserting, or maintaining historical records in a data warehouse's dimension tables. SCDs are particularly useful when dealing with data that changes gradually or irregularly.

There are several types of SCDs:

Type 1 - Overwrite:

- In this approach, the old data is simply overwritten with new data. Historical data is lost, and the dimension table always reflects the most current values.
- This type is suitable when historical data is not important, and you only need the current state of dimensions.
- Use cases: Basic customer information where historical changes aren't necessary.

Type 2 - Add New Row:

- Type 2 maintains historical changes by adding a new row for each change in the dimension.
- It includes attributes like start date and end date to represent the period during which a particular version of the record was valid.
- This approach provides a full history of changes but can increase storage requirements.
- Use cases: Employee records, product details, customer demographics.

Type 3 - Add New Column:

- Type 3 maintains some historical information by adding new columns to the dimension table to capture changes. However, it doesn't provide a complete history.
- It's less storage-intensive than Type 2, but it only retains a limited history.
- Use cases: Tracking a limited number of changes in dimensions without maintaining a full history.

Type 4 - Hybrid Approach:

- Type 4 combines elements of both Type 1 and Type 2.
- It maintains the current version of the record directly in the main dimension table while keeping a separate history table to store changes.
- This approach balances historical tracking and current data efficiency.
- Use cases: Complex scenarios where both current data and historical tracking are important.

Type 6 - Hybrid Slowly Changing Dimension:

- Type 6 combines elements of Type 1 and Type 2, along with additional attributes.
- It introduces a surrogate key for each version of the record to maintain a complete history while also allowing the current version to be directly accessible from the main dimension table.
- This approach offers both current data access and comprehensive historical data.
- Use cases: Complex requirements where current data and historical tracking are critical.

Each SCD type has its own benefits and trade-offs, and the choice depends on your specific business requirements, data volume, and performance considerations.

Rank and dense rank difference :

- RANK(): leaves gaps in the ranking sequence for tied rows. If there are ties, the next rank is increased by the number of tied rows. For example, if two rows have the same value and they are ranked 1st, the next rank will be 3rd.

- DENSE_RANK() : does not leave gaps in the ranking sequence for tied rows. If there are ties, the next rank is increased by 1, regardless of the number of tied rows. For example, if two rows have the same value and they are ranked 1st, the next rank will be 2nd.

Difference between having and where :

In SQL, both the HAVING and WHERE clauses are used to filter rows from a result set, but they are applied at different stages of query processing and serve different purposes:

WHERE Clause:

- The WHERE clause is used to filter rows from the source tables before any grouping or aggregation is performed. It operates on individual rows.
- It is typically used to filter rows based on conditions involving columns in the source tables, such as filtering rows with specific values, date ranges, or other criteria.
- The WHERE clause is applied before the GROUP BY clause in a SQL query.

Example:

```
SELECT department, AVG(salary) AS avg_salary
FROM employees
WHERE salary > 50000
GROUP BY department;
```

HAVING Clause:

- The HAVING clause is used to filter rows from the result set after the GROUP BY clause has grouped the rows into sets or aggregates.
- It is typically used to filter groups or aggregates based on conditions involving the result of aggregate functions like SUM, AVG, COUNT, etc.
- The HAVING clause allows you to filter groups based on their aggregated values.

Example:

```
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

In summary:

- Use the WHERE clause for filtering individual rows from the source tables before grouping or aggregation.
- Use the HAVING clause for filtering groups or aggregates based on conditions involving aggregate functions after the grouping has been applied.

Product table : find the net profit loss for each product :

```
TxnID,ProductID,Cost,Units,      Transaction
1     , 100 ,      1001 ,  5    ,      Buy
2     , 101 ,      20     , 50   ,      Sold
3     , 102 ,      25     , 70   ,      Buy
4     , 100 ,      1200 ,  5    ,      Sold
```

```
SELECT
  ProductID,
  SUM(CASE WHEN Transaction = 'Buy' THEN Cost * Units
```

```
            WHEN Transaction = 'Sold' THEN -Cost * Units
            ELSE 0
        END) AS NetProfitLoss
FROM
    YourTableName
GROUP BY
    ProductID;
```

Python Questions

Wednesday, November 29, 2023 3:47 PM

list :

1. ordered collection of items
2. comma separated
3. []
4. any type
5. duplicates allowed
6. my_list = ["rice", "dal", 1, 2, 2]

set :

1. un-ordered collection of items
2. comma separated
3. {} or set()
4. any type
5. duplicates not allowed
6. my_set = {"rice", "dal", 1, 2]

dictionary :

1. collection of key value pairs
2. comma separated
3. {}
4. any type
5. duplicates allowed
6. my_dict = {"mango":1, "orange":2, "orange":2}

==> item count in list

```
def item_count(my_list):
  dict = {}
  for item in my_list:
    if item not in dict:
      dict[item] = 1
    else:
      dict[item] += 1
  return dict

item_count(['apple', 'banana', 'apple'])
```

==> word count in string

```
def word_count(text):
  dict = {}
  for letter in text:
    if letter not in dict:
      dict[letter] = 1
    else:
      dict[letter] += 1
  return dict

word_count("adsajhgcas")
```

To find the count of each element in the list :

```
from collections import Counter
l = [1, 2, 2, 2, 3, 3, 4, 5]
# Count the occurrences of each element
element_count = Counter(l)
# Print the result
for element, count in element_count.items():
    print(f"Element {element} appears {count} times.")
```

To reverse the words in the string s :

```
s = 'This is Kushal'
# Split the string into words
words = s.split()
# Reverse the words and convert them to lowercase
reversed_words = reversed(words)
# Join the reversed words back into a string
output = ' '.join(reversed_words)
print(output)
```

If you want to filter out elements from the list a that don't contain the letter "a":

```
a = ['apple', 'kiwi', 'guava', 'mango']
result = [item for item in a if 'a' in item]
print(result)

a = ['apple', 'kiwi', 'guava', 'mango']
y = [x for x in a if 'a' not in x]
print(y)
```

To get the elements that are in list a but not in list b :

```
a = [1, 2, 3]
b = [3, 4, 5]
result = [x for x in a if x not in b]
print(result)
```

To count the number of vowels in the string "welcome python" :

```
# Input string
input_string = "welcome python"
# List of vowels
vowels = "aeiouAEIOU"
# Initialize a variable to count vowels
vowel_count = 0
# Iterate through the characters of the string
for char in input_string:
   if char in vowels:
      vowel_count += 1
# Print the vowel count
print("Number of vowels:", vowel_count)
```

list vs tuple difference :

- # Lists are defined using square brackets: [].
- Lists are mutable, which means you can modify their contents (add, remove, or change elements) after they're created.
  - my_list = [1, 'hello', 3.14, [4, 5]]
- # Tuples are defined using parentheses: ().

- # Tuples are immutable, which means you cannot change their contents once they're created. However, you can create new tuples with modified contents.
  - my_tuple = (1, 'hello', 3.14, (4, 5))

reverse the order of words in a sentence or a string in Python :

```
def reverse_words(sentence):
    words = sentence.split()
    reversed_words = ' '.join(reversed(words))
    return reversed_words

input_sentence = "Hello world! This is a sentence."
reversed_sentence = reverse_words(input_sentence)
print(reversed_sentence)
```

create a lambda function to calculate the square of a number in Python :

```
square = lambda x: x ** 2
num = 5
result = square(num)
print(result)
```

Python code to check a number is prime or not :

```
def is_prime(number):
    if number <= 1:
        return False
    if number <= 3:
        return True
    if number % 2 == 0 or number % 3 == 0:
        return False
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6
    return True

user_input = int(input("Enter a number: "))

if is_prime(user_input):
    print(f"{user_input} is a prime number.")
else:
    print(f"{user_input} is not a prime number.")
```

Args vs kwargs :

In Python, args and kwargs are special terms used to pass a variable number of arguments to functions. They are often used when you want to create functions that can accept a flexible number of input arguments.

args (Arbitrary Arguments):

- args stands for "arguments" and is used to pass a variable number of non-keyword arguments to a function.
- When you use *args in the function parameter list, it collects any number of positional arguments as a tuple.
- You can access the arguments inside the function using the tuple indexing.
- Example:

```
def print_arguments(*args):
    for arg in args:
        print(arg)
print_arguments('apple', 'banana', 'cherry')
```

kwargs (Keyword Arguments):

- kwargs stands for "keyword arguments" and is used to pass a variable number of keyword arguments to a function.
- When you use **kwargs in the function parameter list, it collects any number of keyword arguments as a dictionary.
- You can access the arguments inside the function using dictionary keys.
- Example:

```
def print_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")
print_kwargs(fruit='apple', price=0.99, color='red')
```

Combining args and kwargs:
You can also use both args and kwargs together in a function definition, allowing you to accept both positional and keyword arguments in a flexible manner.

```
def example_function(arg1, *args, **kwargs):
    print("arg1:", arg1)
    print("args:", args)
    print("kwargs:", kwargs)

example_function("value1", "value2", kwarg1="value3", kwarg2="value4")
```

In this example, arg1 is a regular positional argument, *args collects additional positional arguments as a tuple, and **kwargs collects keyword arguments as a dictionary.

Remember that the terms args and kwargs are conventions, and you can use any valid variable names. The asterisk (*) in front of args and ** in front of kwargs are what make them special in terms of how arguments are passed to the function.

data skewness in pyspark :

1. uneven distribution of data across partitions
2. some partitions containing significantly more data than others
3. This imbalance can have a significant impact on the performance of your PySpark jobs

how to fix :

1. Data Preprocessing: Cleanse and preprocess your data to handle duplicates, missing values
2. Repartitioning: Repartition your data into a more balanced number of partitions using the repartition() method in PySpark.
3. Custom Partitioning: Implement custom partitioning logic if necessary to ensure a balanced distribution.
4. Key Diversification: If you're partitioning or distributing data based on keys, consider diversifying the keys or using a different partitioning strategy to achieve more balanced partitions.

Addressing data skewness in PySpark is crucial for optimizing the performance of your data processing jobs, particularly when working with large and complex datasets. It often requires a combination of data preprocessing, thoughtful partitioning strategies, and careful monitoring of job execution.

difference between repartition and coalesce:

1. Repartition:

Use Case: Repartition is primarily used to increase or decrease the number of partitions in a DataFrame. It's often used when you want to change the partitioning of data, either to increase parallelism or to redistribute data based on a specific column.

Number of Partitions: You explicitly specify the desired number of partitions when using repartition.

Data Shuffling: Repartitioning may involve data shuffling, which means that data is redistributed across partitions. This can be a resource-intensive operation.

Example:

```
# Repartition a DataFrame into 4 partitions
df = df.repartition(4)
```

2. Coalesce:

Use Case: Coalesce is used to reduce the number of partitions in a DataFrame, typically to optimize performance or to reduce the memory footprint. It's useful when you want to combine existing partitions without shuffling data.

Number of Partitions: You specify the target number of partitions when using coalesce.

Data Shuffling: Coalesce does not involve data shuffling. It reduces the number of partitions by merging existing partitions, which can be done efficiently.

Example:

```
# Coalesce a DataFrame to 2 partitions
df = df.coalesce(2)
```

Here are some considerations for choosing between repartition and coalesce:

-If you need to increase or decrease the number of partitions, and data shuffling is acceptable, use repartition.
-If you want to reduce the number of partitions without shuffling (e.g., to optimize performance), use coalesce.
-coalesce is more efficient than repartition when reducing the number of partitions because it avoids the overhead of data shuffling.
-Data shuffling in repartition can be resource-intensive, so use it carefully, especially with large datasets.

broadcasting pyspark :

-The driver program (the PySpark application) broadcasts the small dataset to all worker nodes in the cluster.
-Each worker node stores the broadcast data in memory.
-When performing operations that involve the broadcasted data (e.g., joins), the worker nodes can perform the operation locally without needing to fetch data from other nodes or shuffle data.

Broadcast Variables:

-In PySpark, you can create a broadcast variable using the SparkContext.broadcast() method.
-For example:

```
from pyspark import SparkContext
sc = SparkContext("local", "BroadcastExample")

small_data = [1, 2, 3, 4, 5]
broadcast_var = sc.broadcast(small_data)
```

Broadcast Joins:

-In the context of joins, you can use the broadcast hint to instruct PySpark to perform a broadcast join when appropriate.

-For example:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("BroadcastJoinExample").getOrCreate()

large_df = ...
small_df = ...

result_df = large_df.join(broadcast(small_df), "join_column", "inner")
```

-Using broadcast() on a DataFrame before joining hints at a broadcast join.

-PySpark's query optimizer will consider using a broadcast join for performance if it's feasible based on the size of the broadcasted DataFrame and the join operation.

optimization techniques in pyspark :

Use the Appropriate Cluster Configuration:

Choose an appropriate cluster size (number of nodes) based on the volume of data and the complexity of your tasks.
Consider using clusters with more memory and CPU resources for handling large datasets.

Data Partitioning and Repartitioning:

Repartition your data into an optimal number of partitions. Too few or too many partitions can impact performance.
Choose the right partitioning key when performing operations like joins to minimize data shuffling.

Broadcasting:

Use broadcast joins for small tables that fit in memory to avoid expensive shuffling operations.
Adjust the broadcast threshold (spark.sql.autoBroadcastJoinThreshold) as needed.

Memory Management:

Tune memory settings, such as executor memory (spark.executor.memory) and driver memory (spark.driver.memory), based on your workload.
Enable memory caching of frequently accessed data using .cache() or .persist().

Data Skew Handling:

Address data skew issues using techniques like key diversification, data preprocessing, or custom partitioning strategies.

Query Optimization:

Use Spark's built-in query optimizer to optimize SQL queries (spark.sql.cbo.enabled).
Analyze query plans using EXPLAIN to identify potential bottlenecks and optimization opportunities.

Data Pruning:

When reading data, use predicate pushdown to limit the amount of data read from storage.

Parallelism:

Adjust the level of parallelism for operations like repartition, coalesce, and groupBy to match cluster resources.

Code Optimization:

Optimize code for readability and performance. Avoid using expensive operations unnecessarily. Use the DataFrame API whenever possible, as it often provides better optimization opportunities compared to RDDs.

persist() vs cache() :

=> persist() allows you to specify a custom storage level, giving you more control over how the data is cached. Common storage levels include MEMORY_ONLY, DISK_ONLY, MEMORY_AND_DISK, etc.

```
from pyspark.storagelevel import StorageLevel
custom_storage_level = StorageLevel.MEMORY_ONLY_SER_2
df.persist(custom_storage_level)
```

To release the memory used by a cached or persisted DataFrame

```
df.unpersist()
```

=> cache() is a shorthand method for persisting a DataFrame or RDD with the default storage level, which is usually MEMORY_ONLY.

```
df.cache()
```

=> Both cache() and persist() store the DataFrame in memory, so subsequent operations on the DataFrame are faster because they can access the data from memory rather than recomputing it.

Data Pruning :

Data pruning is a data optimization technique that involves reducing the amount of data read and processed during data retrieval or data processing operations.

- Predicate Pushdown: In databases and query engines, predicate pushdown involves pushing filters and conditions as close to the data source as possible. For example, when querying a database, you can specify filters in the SQL query to reduce the number of rows retrieved from the database. This minimizes data transfer over the network and reduces the amount of data processed.

- Column Pruning: When querying a dataset, you may not always need all columns. By selecting only the columns you need, you can reduce the amount of data transmitted and processed

- Filtering and Transformation: Apply filters and transformations early in data processing pipelines to eliminate irrelevant or redundant data.

Identify Corrupted Records:

Use the **badRecordsPath** option while reading data from a source like CSV, JSON, Parquet, etc. This option enables you to specify a path where Spark writes records that couldn't be parsed.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CorruptedRecords").getOrCreate()

# Read data and save corrupted records
df = spark.read.option("badRecordsPath", "/path/to/bad_records").csv("/path/to/data.csv")
```

Map, reduce, filter :

map, filter, and reduce are three built-in functions in Python that operate on iterables (like lists, tuples, etc.). They are part of the functools module in Python 3 and are used for different purposes:

map Function:

- The map() function applies a given function to each item in an iterable and returns an iterator that yields the results.
- Syntax: map(function, iterable)
- Example:

```
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x ** 2, numbers)
print(list(squared))  # Output: [1, 4, 9, 16, 25]
```

filter Function:

- The filter() function filters out items from an iterable based on a given function's condition. It returns an iterator containing the items that satisfy the condition.
- Syntax: filter(function, iterable)
- Example:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = filter(lambda x: x % 2 == 0, numbers)
print(list(evens))  # Output: [2, 4, 6, 8, 10]
```

reduce Function:

- The reduce() function (available in the functools module) successively applies a given function to elements of an iterable to accumulate a result. It's not built-in in Python 3 and needs to be imported.
- Syntax: functools.reduce(function, iterable, initial=None)
- Example:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
sum_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)  # Output: 15
```

Please note that while map and filter are straightforward and can be used directly, the reduce function might be less common due to the introduction of list comprehensions, generator expressions, and more expressive ways of achieving similar results in modern Python code

Python code to find the event numbers in list :

```
l1 = [1, 2, 3, 4, 5, 6]

even_numbers = filter(lambda x: x % 2 == 0, l1)
even_numbers_list = list(even_numbers)

print(even_numbers_list)  # Output: [2, 4, 6]
```

To count the occurrences of each item in a list in Python :

```
from collections import Counter

my_list = [1, 2, 3, 4, 1, 2, 3, 5, 4, 5]
item_count = Counter(my_list)

print(item_count)
```

Reverse a string :

```
s = 'srujith'
reversed_s = ''

for char in s:
   reversed_s = char + reversed_s
```

```
print(reversed_s)
```

How to remove the duplicates in list :

```
def remove_duplicates(input_list):
    unique_elements = []
    for item in input_list:
        if item not in unique_elements:
            unique_elements.append(item)
    return unique_elements

input_list = [2, 6, 2, 4, 9, 10, 4]
output_list = remove_duplicates(input_list)
print(output_list)
```

Print

```
d = {"name": "hello"}
x = []
for i in range(len("name")):
    for key, value in d.items():
        x.append(f"{key[i]}:{value}")

result = ",".join(x)
print(result)
```

Extract only strings from html tags

input: test_str = 'RandomTrees is a Golden partner of Snowflake company'
output: RandomTrees is a Golden partner of Snowflake

```
import re

def extract_strings_from_html(html_string):
    clean_text = re.sub(r'<[^>]*>', '', html_string)
    return clean_text

test_str = '<b>RandomTrees</b> is a<b> Golden </b> partner of <b> Snowflake </b> company'
output = extract_strings_from_html(test_str)
print("Extracted strings:", output)
```

1. Find the Missing Number
   list_elements= [1,2,3,5,6,7,8]
   expected_output = [4,9]

2. Write a program in python to identify occurence of each string
   Input: 'RandomTrees'
   Output: {'r':1,'a':1,'n':1,'d':1,'o':1,'m':1,'t':1,'r':1,'e':2,'s':1}

3. Find the minimum & maximum words from the given sentence?

   exclude words: (is , a, of,and ,on)

   input: 'RandomTrees is a pioneering Data & AI company with a culture of innovating and driving ROI on data'

minimum word output: AI
maximum word output: RandomTrees

4. Write a program to calculate the LCM of the two numbers entered by the user?
    a=15
    b=25
    output: 75

5. Extract only strings from html tags

    input:
    test_str = 'RandomTrees is a Golden partner of Snowflake company'
    output: RandomTrees is a Golden partner of Snowflake

Certification Dumps

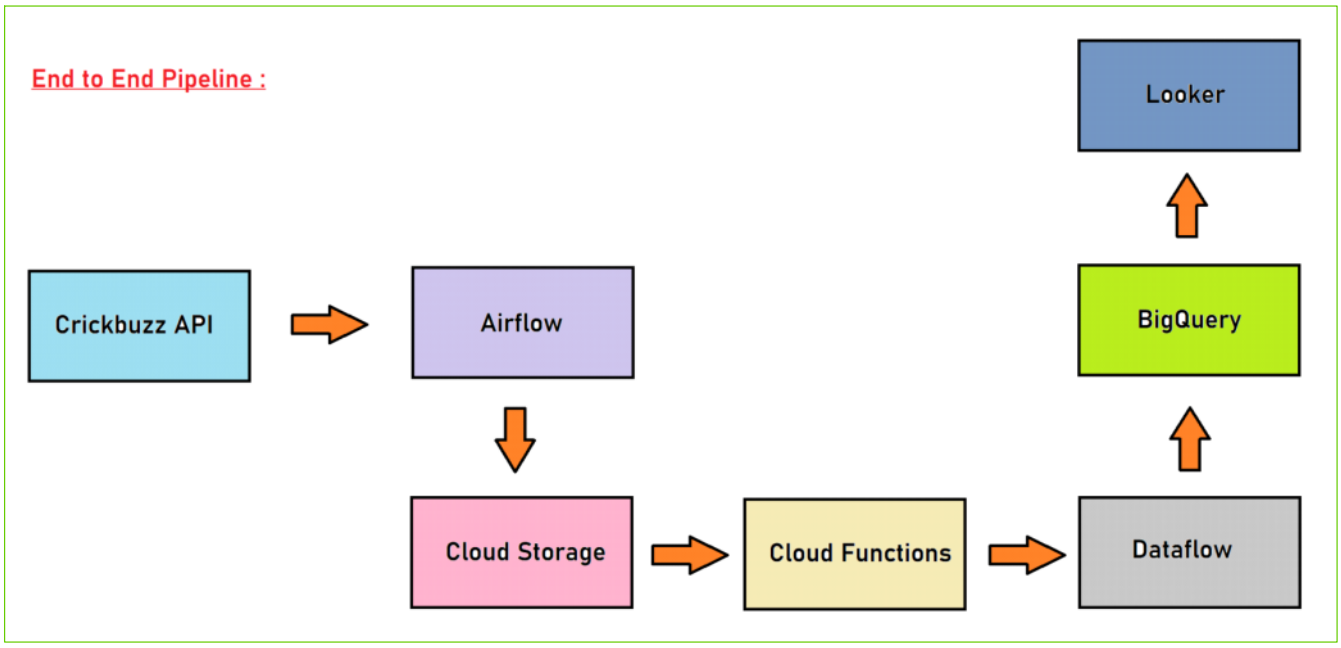
Friday, December 8, 2023 10:04 PM

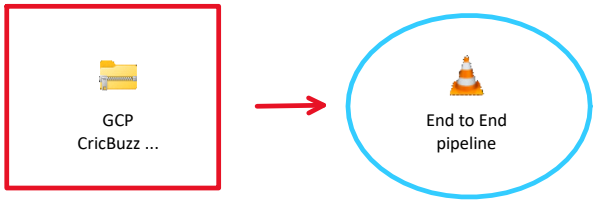
GCP ACE (Associate Cloud Engineer)

GCP PDE (Professional Data Engineer)

End to End Pipeline - project2

Monday, February 12, 2024 5:46 PM





Project Flow :

- I am working on google cloud projects, In this project, we have implemented a data pipeline to ingest data from an Application API using a Python script. This script is scheduled to run as a part of DAG in Cloud Composer. Once the Python script retrieves the data, it is stored in Google Cloud Storage as a CSV file.
- after creation of the CSV file in the Cloud Storage bucket, a Cloud Function is triggered. This Cloud Function automatically submits a beam job to Dataflow, which is configured to load the data from Cloud Storage into BigQuery.
- on top of the BigQuery table, we have created dashboards using Looker, These dashboards provide stakeholders with interactive visualizations, enabling them to derive valuable insights and make informed decisions based on the processed data.

➢ **Setting Up Crickbuzz API :**

- https://rapidapi.com/cricketapilive/api/cricbuzz-cricket
- Create account : username and password
- Click on Pricing => Basic => Subscribe

- Click on Endpoints => stats => stats/get-icc-rankings => test end point => check result
- Change language => python => requests

Building a Cricket Statistics Pipeline with Google Cloud Services

In the world of data engineering, the journey from data retrieval to insightful visualization is an adventure filled with challenges and rewards. In this guide, we'll walk through the intricate steps of constructing a comprehensive cricket statistics pipeline using Google Cloud services. From retrieving data via the Cricbuzz API to crafting a dynamic Looker Studio dashboard,
each phase contributes to the seamless flow of data for analysis and visualization.

Architecture

Data Retrieval with Python and Cricbuzz API

The foundation of our project begins with Python's prowess in interfacing with APIs. We'll delve into the methods of fetching cricket statistics from the Cricbuzz API, harnessing the power of Python to gather the required data efficiently.

Storing Data in Google Cloud Storage (GCS)

Once the data is obtained, our next step involves preserving it securely in the cloud. We'll explore how to store this data in a CSV format within Google Cloud Storage (GCS), ensuring accessibility and scalability for future processing.

Creating a Cloud Function Trigger

With our data safely stored, we proceed to set up a Cloud Function that acts as the catalyst for our pipeline. This function triggers upon file upload to the GCS bucket, serving as the initiator for our subsequent data processing steps.

Execution of the Cloud Function

Within the Cloud Function, intricate code is crafted to precisely trigger a Dataflow job. We'll meticulously handle triggers and pass the requisite parameters to seamlessly initiate the Dataflow job, ensuring a smooth flow of data processing.

Dataflow Job for BigQuery

The core of our pipeline lies in the Dataflow job. Triggered by the Cloud Function, this job orchestrates the transfer of data from the CSV file in GCS to BigQuery. We'll meticulously configure the job settings to ensure optimal performance and accurate data ingestion into BigQuery.

Looker Dashboard Creation

Finally, we'll explore the potential of BigQuery as a data source for Looker Studio. Configuring Looker to connect with BigQuery, we'll create a visually compelling dashboard. This dashboard will serve as the visualization hub, enabling insightful analysis based on the data loaded from our cricket statistics pipeline.

Google Cloud Platform (GCP) is a suite of cloud computing services provided by Google. It offers a wide range of infrastructure and platform services for computing, storage, networking, databases, machine learning, analytics, and more. GCP allows businesses and developers to build, deploy, and scale applications and services using Google's vast global network infrastructure.

Some of the key services and features offered by Google Cloud Platform include:

- **Compute:** GCP provides virtual machines (VMs) known as Google Compute Engine instances, which can be used to run various types of workloads. It also offers managed container services like Google Kubernetes Engine (GKE) for orchestrating containerized applications.
- **Storage:** GCP offers scalable and durable storage options such as Google Cloud Storage for object storage, Google Cloud SQL for managed relational databases, Google Cloud Firestore and Datastore for NoSQL databases, and Google Cloud Bigtable for a fully managed NoSQL database service.
- **Networking:** GCP provides a global network infrastructure that allows you to connect and manage resources across different regions and zones. Services like Google Cloud Virtual Private Cloud (VPC) enable you to create isolated network environments.
- **Big Data and Analytics:** Google Cloud Platform includes services like BigQuery for interactive analysis of large datasets, Dataflow for real-time data processing, and Pub/Sub for event-driven messaging.
- **Machine Learning and AI:** GCP offers a suite of machine learning tools including TensorFlow (an open-source machine learning framework), AI Platform for model development and deployment, Vision AI, Natural Language Processing (NLP) API, and more.
- **Security and Identity Management:** GCP provides features for identity and access management, encryption, security monitoring, and compliance standards to help ensure the security of your applications and data.
- **Serverless Computing:** Services like Google Cloud Functions and Cloud Run allow you to deploy code without managing the underlying infrastructure, enabling you to focus solely on your application logic.
- **IoT**: Google Cloud IoT Core provides tools for connecting, managing, and processing IoT (Internet of Things) data from devices.
- **Developer Tools:** GCP includes tools like Cloud Source Repositories for version control, Cloud Debugger for debugging applications in production, and Cloud Build for continuous integration and continuous delivery (CI/CD).
- **Management and Monitoring:** Google Cloud provides tools like Stackdriver for monitoring and logging, allowing you to gain insights into the performance and health of your applications.

Google Cloud Platform competes with other major cloud service providers like Amazon Web Services (AWS) and Microsoft Azure, offering a wide range of services to cater to various business needs, from startups to enterprises.

GCP Interfaces

Google Cloud Platform (GCP) offers several interfaces and tools that users can use to interact with and manage their cloud resources. These interfaces cater to different levels of technical expertise and use cases.

Here are some of the main interfaces provided by GCP:

- Google Cloud Console (Web UI): The Google Cloud Console is a web-based user interface that provides a graphical way to manage and interact with GCP services. It's suitable for users who prefer a visual interface and may not have extensive technical knowledge. The Console allows you to create, configure, and manage resources through a user-friendly interface.
- Cloud Shell: Cloud Shell is an interactive shell environment that is accessible through the web browser. It comes with pre-installed command-line tools and the Google Cloud SDK, which allows you to manage resources using the command line. It's a convenient way to work with GCP resources without needing to set up local development environments.
- Google Cloud SDK: The Google Cloud SDK is a set of command-line tools that enables developers to interact with GCP services and manage resources from their local development environment. It provides commands for deploying, managing, and troubleshooting GCP resources.
- Cloud APIs: Google Cloud Platform provides a comprehensive set of RESTful APIs that allow developers to programmatically interact with GCP services. These APIs enable you to create, configure, and manage resources using your preferred programming language.
- Integrated Development Environments (IDEs): Some popular integrated development environments, such as JetBrains IntelliJ IDEA and Visual Studio Code, offer extensions and plugins that provide integration with GCP services. These extensions can simplify development and deployment tasks.

- Cloud Client Libraries: Google Cloud offers client libraries for various programming languages that abstract the complexity of using RESTful APIs. These libraries make it easier to interact with GCP services and integrate them into your applications.

- Third-Party Tools: Many third-party tools and integrations are available that provide additional functionality and interfaces for managing and interacting with GCP services. These tools might include development frameworks, monitoring and alerting solutions, and more.

Remember that GCP interfaces cater to different user preferences and levels of technical expertise. You can choose the one that best fits your needs and proficiency in managing cloud resources.

GCP Locations

In Google Cloud Platform (GCP), the concepts of regions and zones are essential for understanding how resources are distributed and deployed across Google's global infrastructure. These concepts help you design and deploy applications for availability, reliability, and performance.

Let's delve into what regions and zones are, their selection criteria, and their importance:

Region:

A region in GCP is a specific geographic area that contains one or more zones. Each region is composed of multiple data centers that are relatively close to each other, but still separated to provide redundancy and fault tolerance. Resources deployed within a region can interact with each other with lower latency compared to resources deployed across different regions. Regions are identified by names like us-central1 or europe-west2.

Zone:

A zone is a more granular unit within a region. Each zone consists of one or more data centers that are physically isolated from each other. Resources deployed in different zones within the same region are designed to be fault-tolerant and isolated from one another, which helps ensure high availability. Zones are identified by a letter appended to the region name, such as us-central1-a or europe-west2-c.

Selection Criteria:

When choosing a region and zone for deploying your GCP resources, consider the following factors:

- Proximity to Users: Choose a region that is geographically close to your users to reduce latency and provide better performance for your applications.

- High Availability: Distribute your resources across multiple zones within a region to achieve high availability. If one zone experiences an outage, your applications can fail over to resources in another zone.

- Data Sovereignty and Compliance: Some regulations and compliance requirements might mandate that data be stored within specific geographic regions. Choose regions that comply with these regulations.

- Resource Availability: Ensure that the required GCP services and resources you intend to use are available in the selected region and zone. Not all services are available in all regions.

- Network Connectivity: Consider the network connectivity between regions and zones. Some regions might have better network connectivity to other regions, which can affect data transfer and communication between resources.

Importance:

Understanding regions and zones is crucial for the following reasons:

- High Availability: Distributing resources across multiple zones within a region ensures that your applications can continue running even if one zone experiences an issue.

- Redundancy: Deploying resources across different regions provides redundancy in case of regional outages or disasters.

- Latency and Performance: Selecting the right region helps minimize latency and improve the overall performance of your applications for end users.

- Compliance and Data Governance: Some industries and regulations require data to be stored within specific geographic boundaries. Choosing the right region helps you comply with these requirements.

- Disaster Recovery: Regions and zones facilitate disaster recovery strategies, allowing you to replicate data and applications in different locations.

In summary, regions and zones in GCP play a critical role in achieving high availability, performance, and compliance for your cloud deployments. Careful consideration of these concepts can significantly impact the reliability and efficiency of your applications and services.

GCP IAM & Admin

Google Cloud Identity and Access Management (IAM) is a fundamental component of Google Cloud Platform (GCP) that governs the authorization and permissions for users, groups, and services to access resources within a GCP project. IAM helps you manage who has access to what resources and what actions they can perform.

Let's explore the key concepts and components of GCP IAM in detail:

Principals:

Users: Individuals who have Google accounts and access to the GCP platform.
Service Accounts: Special accounts used by applications, VM instances, or other services to authenticate and interact with GCP APIs.
Groups: A collection of Google accounts and service accounts, allowing you to manage permissions collectively.
G Suite Domains: Google Workspace (formerly G Suite) domains can be used to grant access to users within your organization's domain.

Roles:

Primitive Roles: These are basic roles with broad permissions, such as Owner, Editor, and Viewer. Primitive roles are project-level roles and are not recommended for fine-grained access control.
Predefined Roles: Google provides a set of predefined roles with specific permissions for various GCP services. These roles are more granular and can be assigned at different levels like project, folder, or resource.
Custom Roles: You can create your own roles with custom permissions tailored to your specific needs. Custom roles are useful when predefined roles don't match your requirements.

Permissions:

Permissions are specific actions that can be performed on resources, such as read, write, create, delete, and more.
Roles consist of one or more permissions. For example, a role might grant permissions to read data from a storage bucket or write data to a database.

Policies:

An IAM policy is a JSON document that defines who has what level of access to which resources.
Policies consist of bindings, where each binding associates a role with one or more members (users, groups, service accounts, etc.).
Policies are applied at different levels: organization, folder, project, and resource.

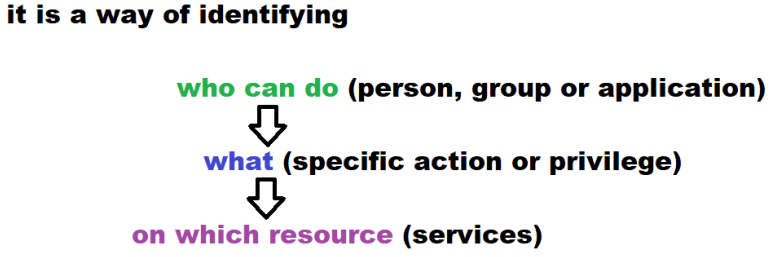
Resource Hierarchy:

GCP resources are organized hierarchically: Organizations contain folders, folders contain projects, and projects contain resources.
IAM policies are inherited downward in the hierarchy. Permissions set at higher levels apply to resources within those levels.

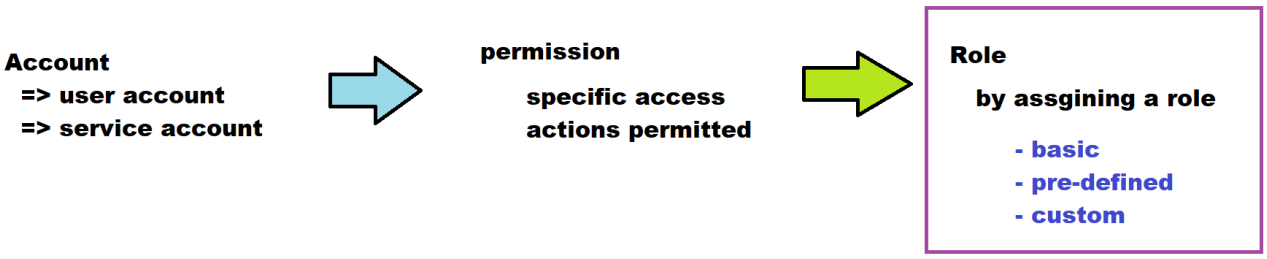
IAM Best Practices:

- Use the principle of least privilege: Assign only the necessary permissions to each user or service account.
- Use predefined roles when possible to avoid potential mistakes and ensure consistent access control.
- Avoid using primitive roles due to their broad permissions.
- Regularly review and audit permissions to ensure they remain appropriate.

IAM is crucial for maintaining security, access control, and compliance in your GCP environment. By properly configuring IAM policies and roles, you can ensure that users and services have the right level of access to resources while preventing unauthorized access or accidental modifications. It's recommended to carefully plan and implement IAM policies to align with your organization's security and operational requirements.

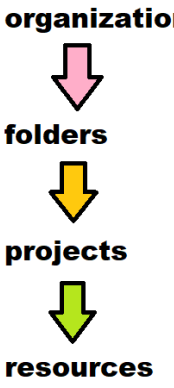


Identity Access Management



role : contains set of permissions that allows you to perform specific actions on GCP services

Resource Hierarchy



you can set IAM rules / policies at any level in this Resource Hierarchy

1. IAM manages the authorization in GCP
2. Central manager which manages who can do what on which resource
   a. Who (a person or group of persons or application)
   b. What (kind of action that can be done)
   c. Which (a service or resource)

How to Create Free Tier Account in GCP

Video : https://drive.google.com/file/d/16zykMZzUHHL6-vxoMBxTaRF07qvuTlQt/view?usp=sharing

pre-requisites :

1. google account (existing / dummy account)
2. debit or credit card details (only 2 rupees)

Steps :

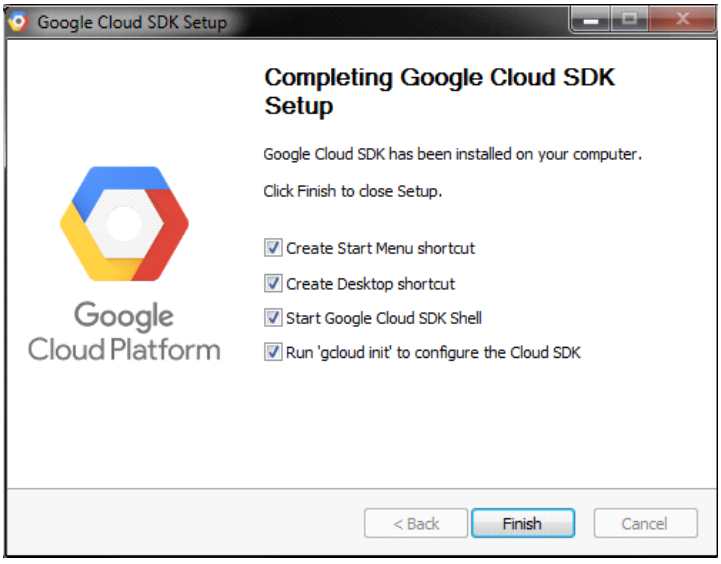
1. go to incognito mode
2. https://console.cloud.google.com/
3. enter your credentials : mail and password click on try for free :
   step-1 : country=india org need : business idea
   step-2 : account type :
   individual
   payment method : card details address
   tax information : un-registered individual
   and complete the card transaction with OTP ===> FREE CREDITS ADDED => check in billing section

Install SDK

17 April 2023 11:13

To install the **Google Cloud SDK** on the **Windows system**, you can follow the below steps.

1. Download the Google Cloud SDK installer.
2. Launch the installer and follow the prompts.
3. After the installation has completed, the installer presents several options as shown below.



- Make sure that the following are selected.

Start Google Cloud SDK :

- gcloud auth login
- Browser opens : login to cloud
- You are now logged in to your account
- Select the project : **gcloud config set project <project-id>**
- Interact with services

Linux Basics

Monday, August 7, 2023 3:10 PM

Linux is an open-source operating system based on the Unix kernel. It is widely used in various devices and systems, including servers, desktops, smartphones, and embedded devices. Linux provides a command-line interface (CLI) where users can interact with the system using commands.

Here are some common Linux commands categorized based on their functionality:

folder = directory
clear = clear the screen
tab = finish off the rest
upper arrow : to go to previous command
down arrow : to switch to next command
cntl+z, cntl+c : to interrupt the shell session

File and Directory Operations:

- **ls (List):** The ls command is used to list files and directories in the current working directory.
  - Example: ls
- **cd (Change Directory):** The cd command is used to change the current working directory.
  - Examples:
    - Change to a specific directory: cd /path/to/directory
    - Move to the home directory : cd
    - Move to the parent/previous directory: cd ..
- **pwd (Print Working Directory):** The pwd command is used to print the current working directory.
  - Example: pwd
- **mkdir (Make Directory):** The mkdir command is used to create a new directory.
  - Example: mkdir new_directory
- **rmdir (Remove Directory):** The rmdir command is used to remove an empty directory.
  - Example: rmdir directory_to_remove
  - rm -rf directory (to force remove directory)
- **cp (Copy):** The cp command is used to copy files or directories from one location to another.
  - Example: cp file.txt /path/to/destination/
- **mv (Move or Rename):** The mv command is used to move files or directories from one location to another or to rename files/directories.
  - Examples:
    - Move a file to a different location: mv file.txt /new/location/
    - Rename a file: mv old_name.txt new_name.txt
- **rm (Remove):** The rm command is used to remove files or directories.
  - Examples:
    - Remove a file: rm file.txt
    - Remove a directory and its contents (recursively): rm -r directory_to_remove
- **touch:** The touch command is used to create an empty file or update the timestamp of an existing file.
  - Examples:
    - Create a new empty file: touch new_file.txt
    - Update the timestamp of an existing file: touch existing_file.txt
- **Nano:** editor
  - To create and add data inside file, save

File Content Manipulation:

- **cat:** display the contents of one or more files.
  - Example:
    - cat file.txt

- **less:** The less command is used to view the content of a file one page at a time. It allows you to navigate through the file using arrow keys.
  - Example:
    - less large_file.txt

- **head:** The head command is used to display the beginning (top) of a file. By default, it displays the first 10 lines, but you can specify a different number of lines.
  - Examples:
    - Display the first 10 lines of a file: head file.txt
    - Display the first 20 lines of a file: head -n 20 file.txt

- **tail:** The tail command is used to display the end (bottom) of a file. By default, it displays the last 10 lines, but you can specif y a different number of lines.
  - Examples:
    - Display the last 10 lines of a file: tail file.txt
    - Display the last 20 lines of a file: tail -n 20 file.txt

- **grep:** The grep command is used to search for a specified pattern in a file or in the output of another command.
  - Examples:
    - Search for the word "apple" in a file: grep "apple" fruits.txt
    - Search for a pattern in multiple files: grep "error" file1.log file2.log
    - Search for a case-insensitive pattern: grep -i "important" notes.txt

Text Processing:

- awk: The awk command is a versatile text processing tool that allows you to extract and manipulate specific fields from text data based on patterns and actions.
  - Example: Let's say we have a file named "sales.txt" containing sales data with columns: Date, Product, and Amount. To extract the total sales amount, you can use awk like this:
    - awk -F ',' '{sum += $3} END {print "Total sales amount:", sum}' sales.txt

- sed: The sed command (stream editor) is used to filter and transform text data using regular expressions and specified comman ds.
  - Example: Replace all occurrences of "old_text" with "new_text" in a file named "data.txt":
    - sed 's/old_text/new_text/g' data.txt

- cut: The cut command is used to remove sections (columns) from each line of a file based on a delimiter.
  - Example: Let's say we have a file named "employees.txt" with columns: Name, Age, and Department. To extract only the names, you can use cut like this:
    - cut -d ',' -f 1 employees.txt

- sort: The sort command is used to sort lines of text files in ascending or descending order.
  - Example: Sort a file named "numbers.txt" containing numbers in ascending/descending order:
    - sort numbers.txt
    - sort -r numbers.txt

- uniq: The uniq command is used to remove duplicate lines from a sorted file. It only considers adjacent duplicate lines.
  - Example: Let's say we have a file named "colors.txt" with duplicate color names. To remove duplicates and get unique colors:
    - sort colors.txt | uniq

- Sample Data for Examples:
  - sales.txt:

```
Date,Product,Amount
2023-01-01,Item1,100
2023-01-02,Item2,150
2023-01-03,Item1,120
2023-01-03,Item2,180
```

  - employees.txt:

```
Name,Age,Department
John,30,HR
Alice,25,IT
Bob,28,Finance
```

  - numbers.txt:

5
3
8
1

- colors.txt:
  Red
  Blue
  Green
  Red
  Yellow
  Blue

User and Permissions:

- whoami: The whoami command is used to print the current username of the user who is currently logged in.
  - Example: whoami
- id: The id command is used to display user and group information, including the user's UID (User ID), GID (Group ID), and associated groups.
  - Example: id
- su: The su command is used to switch to another user account or become the superuser (root) by default.
  - Examples:
    - Switch to the user account "john": su john
    - Switch to the superuser (root) account: su
- sudo: The sudo command is used to execute a command as the superuser (root) or another user with elevated privileges.
  - Example: sudo apt-get update
- chmod: The chmod command is used to change file permissions (read, write, execute) for users, groups, and others.
  - Example: chmod +x script.sh
- chown: The chown command is used to change the owner and group of a file or directory.
  - Example: chown user:group file.txt
- Sample Data:
- Let's consider a file named "data.txt" with the following content:
  - Hello, World!
- To change the owner to "john" and the group to "users" for "data.txt": chown john:users data.txt
- After running this command, the ownership of "data.txt" would be changed to user "john" and group "users."

Process Management:

- ps:The ps command is used to display a list of currently running processes on the system.
  - Example: ps
- top: The top command is used to monitor system processes and resource usage in real-time. It provides an interactive interface to view CPU, memory, and other metrics.
  - Example: top
- kill: The kill command is used to send a signal to terminate a process by its PID (Process ID).
  - Example: kill PID
    - Replace PID with the actual Process ID of the process you want to terminate.
    - For example: kill 1234
- pkill: The pkill command is used to send a signal to terminate a process by its name.
  - Example: pkill process_name

System Information:

- uname:The uname command is used to print system information, including the operating system name, kernel version, and more.
  - Example: uname -a
- df: The df command is used to display disk space usage for all mounted file systems.
  - Example: df -h
  - The -h flag stands for "human-readable," and it formats the sizes in a more understandable format (e.g., KB, MB, GB).

- free: The free command is used to display memory usage information, including total, used, and free memory.
  - Example: free -h
  - The -h flag, similar to the df command, makes the sizes human-readable.

- uptime: The uptime command is used to display how long the system has been running, along with the current time, number of users, and system load averages.
  - Example: uptime

Networking:

- ifconfig: The ifconfig command is used to configure and display network interfaces on Unix-like operating systems. However, on newer systems, the ip command has largely replaced it.
  - Example: ifconfig

- ping: The ping command is used to send ICMP echo requests to a host for testing network connectivity and measuring round-trip time.
  - Example: ping google.com

- netstat: The netstat command is used to display network statistics and active network connections. It provides information ab out open ports, listening services, and more.
  - Example: netstat -tuln
  - The -tuln flags filter the output to display TCP (-t) and UDP (-u) connections along with the corresponding port numbers in numeric format (-n).

- ssh: The ssh command is used to securely connect to a remote server over a network. It provides encrypted communication for remote access and command execution.
  - Example: ssh username@remote_server

Package Management (package-dependent):

- apt (Advanced Package Tool): The apt command is used in Debian-based Linux distributions like Ubuntu to manage software packages. It provides a user-friendly way to install, update, and remove packages from the system.
  - Examples:
    - Install a package: sudo apt install package_name
    - Update package lists and upgrade installed packages: sudo apt update, sudo apt upgrade
    - Remove a package: sudo apt remove package_name

These are just a few examples of the many Linux commands available. The Linux command-line interface provides a powerful and flexible way to interact with the operating system and perform various tasks efficiently.

GOOGLE CLOUD STORAGE

08 February 2023 16:26

What is GCS ?

- ✓ fully managed (no infra headache)
- ✓ a service for storing :
  - any type of data,
  - any amount of data,
  - any number of files
- ✓ Object storage service (Object = file of any format)
- ✓ we need to create bucket : a bucket is a container to store your objects (file)
- ✓ Allows world-wide storage and retrieval
- ✓ Scalable and Secure platform

how to create bucket ? and bucket properties ? : Console

1. bucket name : global unique
2. location :
   region : no replication, latency, less available
   dual-region : replication in two regions, less latency, 99.95% HA
   multi-region : replication among multiple regions, very less latency, 99.99999% HA
3. Storage Classes :
   standard : frequently accessing data
   nearline : access/month
   coldline : access/quarter
   archive : access/year
4. access control :
   uniform : Access @bucket level - it can be achieved by (IAM-Identity Access Management)
   fine-grained : Access @object level + @bucket level - it can be achieved by (IAM + ACL(Access Control List))
5. object versioning : if you enable it, you can able to store the older version data
   retention policy : if you enable it, choose the retention period. during the retention period, no one will be able to delete/modify your objects inside the bucket

NOTE :

- whenever we use any service in any cloud platform, there are two costs associated
  1. storage cost
  2. Retrieval/Computation cost

Cloud Shell

09 February 2023 08:39

Note :

If anyone wanted to interact with GCP services there are two ways

1. Web-UI : Console
2. Command Line Tool : Cloud Shell Terminal/sdk

Cloud Shell :

a. It is small machine, linux OS
b. Command line tool
c. 5gb storage
d. Language support : sql, python, go, java, kubectl, terraform, docker, go, nodejs …….
e. If we wanted to work with the shell, first activate it, after that select the proper project
f. Use Shell commands/linux commands interact with services

Name	Summary
List all buckets and files	`gsutil ls, gsutil ls -lh gs://<bucket-name>`
Download file	`gsutil cp gs://<bucket-name>/<dir-path>/package-1.1.tgz .`
Upload file	`gsutil cp <filename> gs://<bucket-name>/<directory>/`
Cat file	`gsutil cat gs://<bucket-name>/<filepath>/`
Delete file	`gsutil rm gs://<bucket-name>/<filepath>`
Move file	`gsutil mv <src-filepath> gs://<bucket-name>/<directory>/<dest-filepath>`
Copy folder	`gsutil cp -r ./conf gs://<bucket-name>/`
Show disk usage	`gsutil du -h gs://<bucket-name/<directory>`
Create bucket	`gsutil mb gs://<bucket-name>`

**********interacting with Cloud Storage via Cloud Shell**********

Creating Buckets: default parameters

gsutil mb gs://<bucket-name>

with all properties:

```
gcloud storage buckets create gs://gjkhgjkhsdkjkjhjklhkljhlkjksd \
--project=project-demo-1-390105 \
--default-storage-class=NEARLINE \
--location=US-WEST2 \
--uniform-bucket-level-access \
--retention-period=50s
```

Listing Objects and Buckets:

gsutil ls: List all buckets.
gsutil ls gs://bucket-name: List objects in a specific bucket.

Copying Data:

gsutil cp file.txt gs://bucket-name: Copy a local file to a bucket.
gsutil cp gs://bucket-name/file.txt : Copy a file from a bucket to the current working directory.

Moving/Renaming Data:

gsutil mv file.txt gs://bucket-name/destination/: Move or rename a file within the same bucket or between buckets.

Removing Objects and Buckets:

gsutil rm gs://bucket-name/file.txt: Remove an object (file) from a bucket.
gsutil rm gs://bucket-name: Remove an empty bucket. Add the -r option to remove a non-empty bucket.

Getting Information:

gsutil du -sh gs://bucket-name: Get the total size of a bucket.
gsutil stat gs://bucket-name/file.txt: Get detailed information about an object (file).

Assignment :

1. create one bucket-1 default options
2. create one bucket-2 with all properties
3. cloud shell => copy file => bucket-1
4. bucket-1 => move file => bucket-2
5. delete file => bucket-2
6. delete bucket-1

Assignment :

a. Create a bucket using the **SDK (install sdk)**
b. Move a file from one bucket another bucket using sdk
c. **Using cloud SDK**, try to transfer a file from local machine to cloud storage bucket

gsutil ls gs://**/'*.csv'

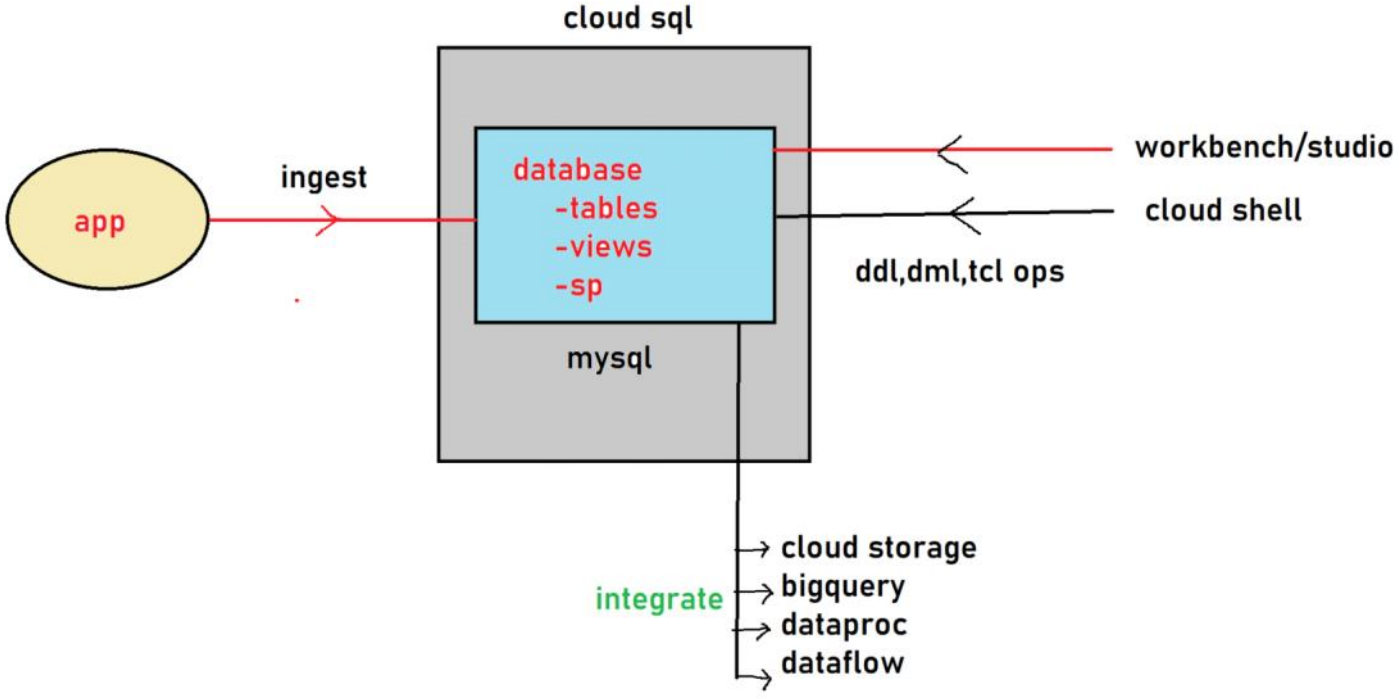
Question : I have list of zip files in my bucket, I want to get recent zip file and to unzip it and move to other folder. how can we achieved this cloud storage via cloud shell

```
gcloud config set project project-demo-1-390105

gsutil ls gs://source_buckets

gsutil ls gs://source_buckets/*.zip

recent_zip=$(gsutil ls -l gs://source_buckets/*.zip | sort -k 2 -r | grep -o "gs://.*.zip" | head -n 1)

gsutil cp "${recent_zip}" gs://source_bucket_21_06_2023/

gsutil cp "${recent_zip}" - | tar -x -C gs://source_bucket_21_06_2023/
```

Cloud SQL

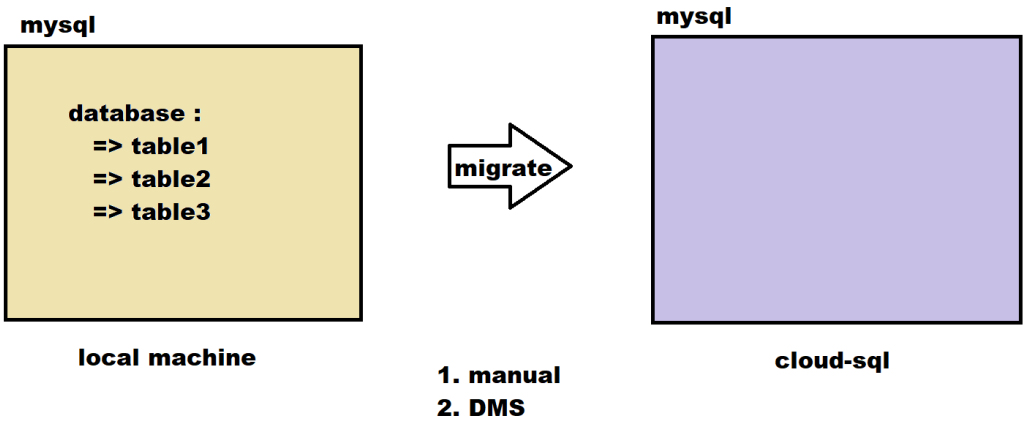
08 February 2023 16:36

1. Fully managed
2. RDBMS solution :
   a. structured data / tabular data
   b. columns and rows
   c. Schema must be defined properly
3. Database as a service
4. Database engine :
   a. MySQL
   b. SQL server
   c. PostgreSQL
5. Only Regional Service (multi-zones)
6. Secure and compliant
7. Scalability (10GB to 64TB) per instance => automatic storage increase option
8. Built in tools to create / setup your database ==> min
9. Automatic Replication
10. Managed backups
11. High Available
12. To work/connect with the cloud SQL :
    a. On-prem workbench/studio
    b. Cloud shell
    c. Apps
13. It can be integrated with other systems easily



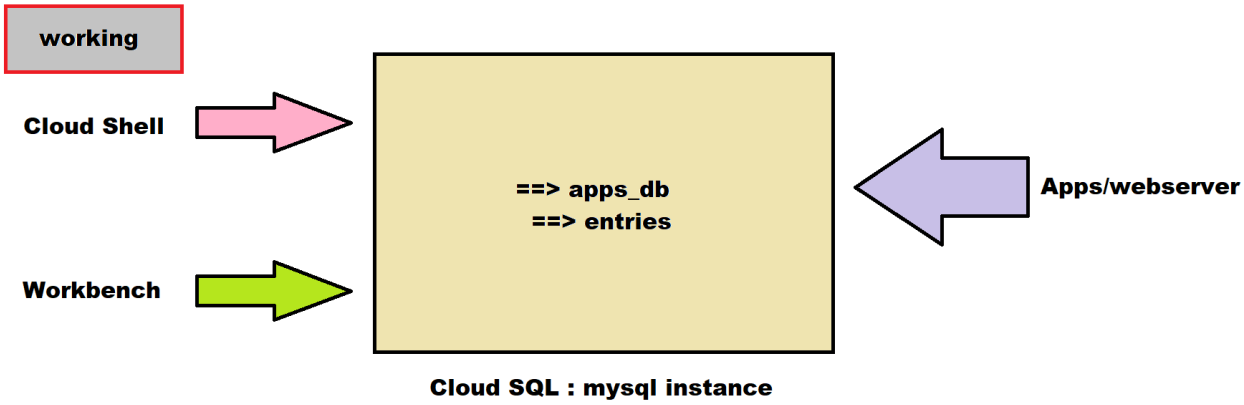
mysql download and install
https://www.youtube.com/watch?v=2c2fUOgZMmY

IPv4 :
https://whatismyipaddress.com/



Connecting to SQL

03 March 2023 22:13



Cloud Shell :

1. activate shell
2. gcloud sql connect <instance_name> --user=root
3. enter password
4. welcome to mysql shell

a. create database
b. create table
c. load data

Note : always end your query with " ; "

Work bench :

1. click on set up new connection (+)
2. connection name ; <*******>
3. connection method : TCP/IP
4. GCP => CloudSQL => hostname, user, password
5. test connection ==> fail

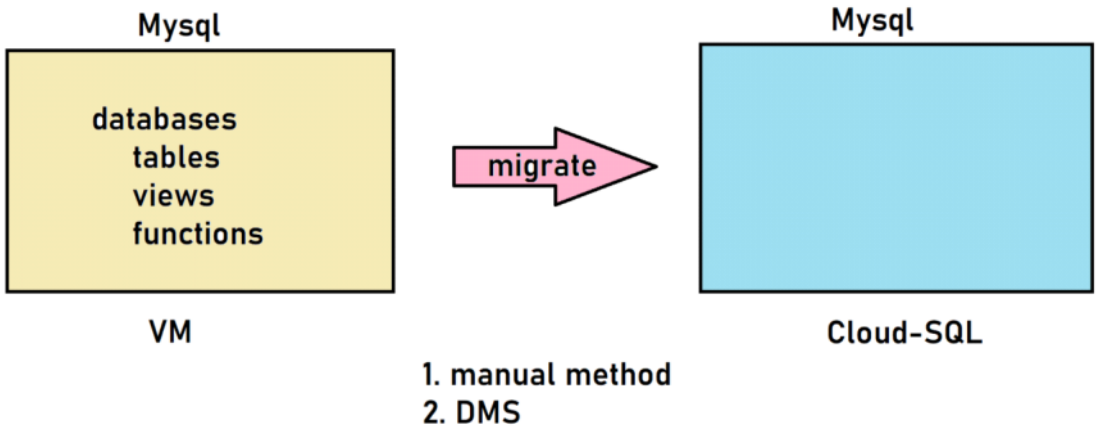
6. it failed => workbench doesnt have the permission to use the cloud sql instance

   it can be done by providing the permssioin in cloud sql connection => add connection => provide network

7. test connection => successful => have rich UI to work with mysql instance

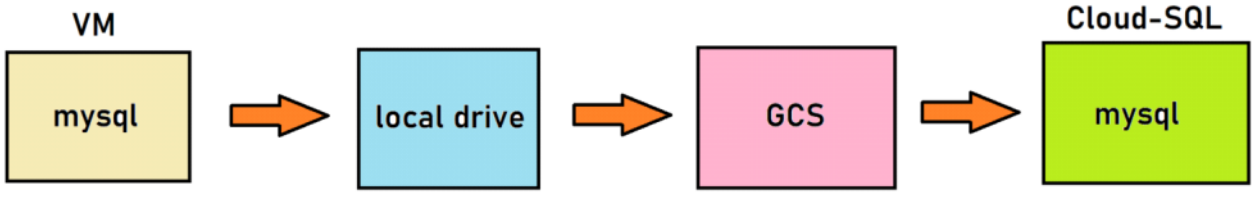
Data Migration

06 July 2023 21:14



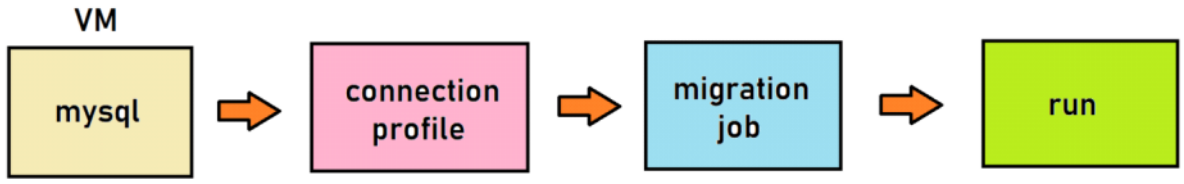
1. manual method
2. DMS

1. manual method :

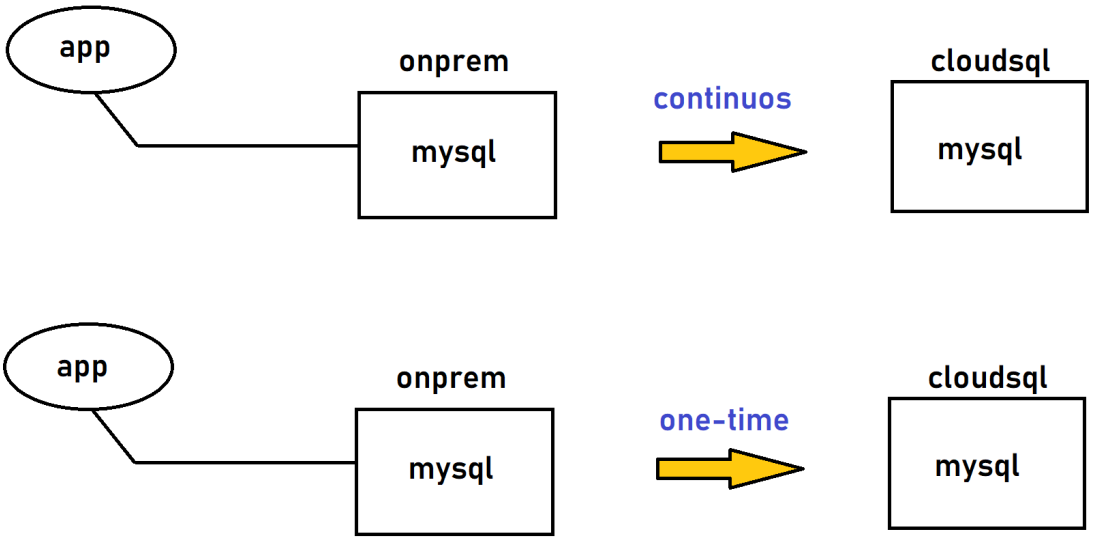


1. move all the files from local to GCS bucket using SDK

2. DMS :



1. get started
2. define source
3. define destination
4. connectivity method
5. Test

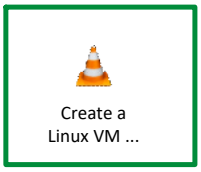

app
onprem
mysql
continuos
cloudsql
mysql
app
onprem
mysql
one-time
cloudsql
mysql

Mysql on Linux VM

Friday, October 13, 2023 8:52 PM

Create a Linux VM Machine in GCP and install MYSQL

Video Link =======>



1. **Create VM with**
   a. Name
   b. Region and zone
   c. Machine configuration
   d. UBUNTU OS
   e. Full access to cloud APIs
   f. allow all https/http
   g. Click on SSH ==> terminal opens

2. **Install mysql on VM :**
   a. https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04
   b. Installed mysql successfully
   c. Username : **root**, Password : **password**
   d. mysql -u root -p

3. **create remote user :** inside mysql

```
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypass';
CREATE USER 'myuser'@'%' IDENTIFIED BY 'mypass';

GRANT ALL ON *.* TO 'myuser'@'localhost';
GRANT ALL ON *.* TO 'myuser'@'%';
FLUSH PRIVILEGES;
```

1. **Command for configuring mysqld.cnf file :**
   a. sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
   b. bind-address : 0.0.0.0
   c. save(cntr+x)

2. **Create a firewall rule :**
   a. Select firewall
   b. Click on create firewall rule
      i. Name
      ii. Target : All instances in the network
      iii. Source IPv4 ranges = 0.0.0.0/0
      iv. Protocols and ports : allow-all
      v. Click on create

3. **Restart MySQL Service in Linux :**
   a. sudo service mysql restart

4. **open Mysql workbench :**
    - external IP : <vm-external-ip>
    - username : myuser
    - password : mypass
    - Port : 3306

sample-data

15 May 2023 21:39

```
-- Create the sample database
CREATE DATABASE sample_database;
USE sample_database;

-- Create the sample tables
-- Table 1: Users
CREATE TABLE Users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL,
  age INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table 2: Products
CREATE TABLE Products (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  price DECIMAL(10, 2),
  description TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table 3: Orders
CREATE TABLE Orders (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT,
  product_id INT,
  quantity INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES Users(id),
  FOREIGN KEY (product_id) REFERENCES Products(id)
);

-- Table 4: Categories
CREATE TABLE Categories (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL
);

-- Table 5: Product_Category
CREATE TABLE Product_Category (
  product_id INT,
  category_id INT,
  PRIMARY KEY (product_id, category_id),
  FOREIGN KEY (product_id) REFERENCES Products(id),
  FOREIGN KEY (category_id) REFERENCES Categories(id)
);
```

```
-- Insert records into the Users table
INSERT INTO Users (username, email, age)
VALUES
  ('john_doe', 'john.doe@example.com', 25),
  ('jane_smith', 'jane.smith@example.com', 30),
  ('mike_jackson', 'mike.jackson@example.com', 35),
  ('sara_williams', 'sara.williams@example.com', 28),
  ('alex_turner', 'alex.turner@example.com', 32),
  ('emily_brown', 'emily.brown@example.com', 27),
  ('chris_robinson', 'chris.robinson@example.com', 29),
  ('laura_miller', 'laura.miller@example.com', 31),
  ('ryan_harris', 'ryan.harris@example.com', 26),
  ('jessica_taylor', 'jessica.taylor@example.com', 33);

-- Insert records into the Products table
INSERT INTO Products (name, price, description)
VALUES
  ('Product A', 10.99, 'This is product A'),
  ('Product B', 20.99, 'This is product B'),
  ('Product C', 15.99, 'This is product C'),
  ('Product D', 5.99, 'This is product D'),
  ('Product E', 12.99, 'This is product E'),
  ('Product F', 8.99, 'This is product F'),
  ('Product G', 18.99, 'This is product G'),
  ('Product H', 7.99, 'This is product H'),
  ('Product I', 14.99, 'This is product I'),
  ('Product J', 9.99, 'This is product J');

-- Insert records into the Orders table
INSERT INTO Orders (user_id, product_id, quantity)
VALUES
  (1, 1, 2),
  (2, 3, 1),
  (3, 2, 3),
  (4, 4, 2),
  (5, 1, 1),
  (6, 3, 4),
  (7, 2, 2),
  (8, 4, 3),
  (9, 1, 3),
  (10, 3, 2);

-- Insert records into the Categories table
INSERT INTO Categories (name)
VALUES
  ('Category A'),
  ('Category B'),
  ('Category C'),
  ('Category D'),
  ('Category E'),
  ('Category F'),
  ('Category G'),
  ('Category H'),
  ('Category I'),
  ('Category J');

-- Insert records into the Product_Category table
INSERT INTO Product_Category (product_id, category_id)
VALUES
  (1, 1),
  (2, 2),
  (3, 3),
  (4, 4),
  (5, 5),
  (6, 6),
  (7, 7),
  (8, 8),
  (9, 9),
  (10, 10);
```

Assignment

Monday, February 12, 2024 8:58 PM

- **create CLOUD SQL instance**

- **install mysql workbench in your laptop**
  https://www.youtube.com/watch?v=2c2fUOgZMmY

- **Linux Vm with mysql installation**

SQL vs Spanner

16 February 2023 08:06

Zones => Regions => multi region/Global

Cloud SQL	Cloud Spanner
Fully Managed	Fully Managed
RDBMS solution(Transactional systems)	RDBMS solution(Transactional systems)
Tabular data, schema	Tabular data, schema
Regional Service	Global service
replication in Zones	Replication in regions
Zonal Availability(Multi Zones)	Regional Availability(Multi Regions)
It is built on top of the MySQL db	It is a google product, introduced by Google
Storage limit : 10GB - 64TB	No storage limit (4TB/node)
Vertical scalable	Horizontal Scalable
Properties = RDBMS	Properties = RDBMS + NoSQL db
Either work bench, shell	Rich UI where u can see dbs, tables, data,…
Cloud SQL is for those who are comfortable using the mysql, postgresql, sql server	If you want to have new solution for RDBMS
SQL	Standard SQL
Less cost	Costlier

	Cloud Spanner	Traditional SQL	Traditional NoSQL
SQL	Yes ✓	Yes ✓	No ✗
Schema	Yes ✓	Yes ✓	No ✗
Consistency	Strong ✓	Strong ✓	Eventual ✗
Availability	High ✓	Failover ✗	High ✓
Scalability	Horizontal ✓	Vertical ✗	Horizontal ✓
Replication	Automatic ✓	Configurable	Configurable

1 Node = 4TB = 1000 Processing Units

Spanner instance

17 February 2023 08:28

Cloud Spanner Instance creation :

create an instance by providing name and ID

1. Create database

2. **Create Table**

1.
```
CREATE TABLE Singers (
SingerId   INT64 NOT NULL,
FirstName  STRING(1024),
LastName   STRING(1024),
SingerInfo INT64,
) PRIMARY KEY (SingerId);
```

2.
```
CREATE TABLE Albums (
SingerId     INT64 NOT NULL,
AlbumId      INT64 NOT NULL,
AlbumTitle   STRING(MAX),
) PRIMARY KEY (SingerId, AlbumId);
```

3.
```
CREATE TABLE Songs (
 SingerId     INT64 NOT NULL,
 AlbumId      INT64 NOT NULL,
 TrackId      INT64 NOT NULL,
 SongName     STRING(MAX),
) PRIMARY KEY (SingerId, AlbumId, TrackId),
 INTERLEAVE IN PARENT Albums ON DELETE CASCADE;

INSERT INTO
Singers (SingerId,FirstName, LastName, SingerInfo)
VALUES
(1,"Marc","Richards",1010),
(2,"Catalina","Smith", 1100 ),
(3,"Ravi","Shankar", 1001),
(4,"Shaik","Saidhul", 1011),
(5,"Madhav","Reddy" ,1111);
```

```
INSERT INTO
Albums (SingerId,AlbumId, AlbumTitle)
VALUES
(1,1, "RRR"),
(1,2, "Pushpa"),
(2,1, "Akanda"),
(2,2, "Ante Sundaraniki"),
(2,3, "Thank you");

INSERT INTO
 Songs (SingerId,AlbumId,TrackId,SongName)
VALUES
 (1,2,1,"title song" ),
 (1,2,2, "Lovers adda"),
 (2,1,1, "mohabbat pyari"),
 (2,1,2, "ra ra rakkamma"),
 (2,1,3, "rain song"),
 (2,3,1, "You are my love");
```

Query data :

```
SELECT
  *
FROM
  Singers AS s
JOIN
  Albums AS a
 ON
  s.SingerId = a.SingerId
JOIN
  Songs b
ON
  s.SingerId = b.SingerId;
```

DWH content

22 February 2023 22:00

1. What is Data Warehouse
2. What is Data Mart
3. Difference between Data Warehouse vs Data Mart
4. Data Warehouse design approaches
    a. Top-down
    b. Bottom-up
5. OLTP(Normalization) VS OLAP(De-normalization)
6. Dimension Modelling
    a. Fact/Measure
    b. Dimension
7. Introduction to Schemas
    a. Star
    b. Snow flake
8. Types of Fact and Dimension

Data Warehousing

22 February 2023 08:08

1. Data warehouse is a
   a. Centralized large repo/database
   b. Where it stores integrated, summarized and historical data for analysis and decision making purpose

2. Data warehouse
   a. Subject oriented (finance, Banking, production,…)
   b. Integrated (get the data from multiple sources)
   c. Non-volatile (read only data)
   d. Time variant (different periods of data)

3. Data Mart :
   a. Is a specific subject oriented database (loans)
   b. Dwh is a collection of data marts

Fact :

1. Measures or numerical values that we want to perform aggregations, slice, dice and analyze
2. Measures are always answer questions like
   a. How much
   b. How many

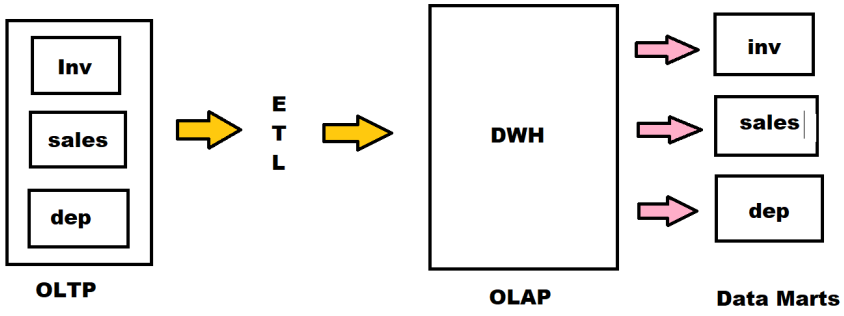
Dimensions :

1. Is a string or textual data, which always answer the questions like when, where, how, which, who..
2. Dimension table PK is FK in fact tables

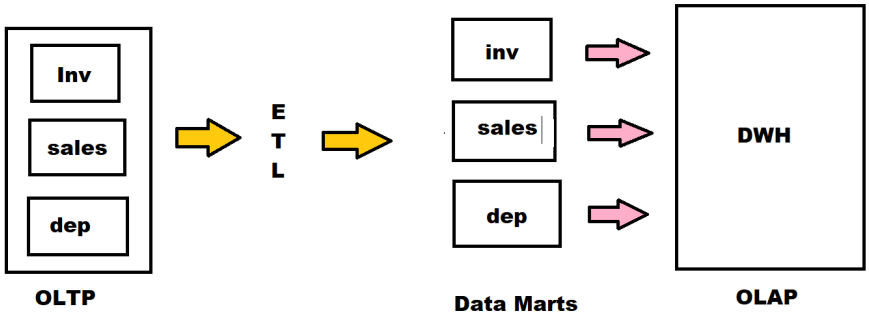
Concepts

23 February 2023 09:04

Top Down approach :



Bottom-UP approach :



OLTP : Normalization ERD modelling

1. master table (pk, string data, less data)

productmaster	(pid,	name,	price,	discount,	qty)
	1	pen	35	5	2000
	2	erasor	10	2	3000

customermaster	(cid,	name,	addr,	city,	pincode,	phone)
	101	john	***	blr	560035	*****
	102	raj	***	hyd	500001	*****

2. Transaction tables (fk, numerical, large data)

sales	(pid,	cid,	dos,	salesamt,	gst,	qty,	dod)
	1	101	10-02-2023	1000	10	12	12-02-2023

insert
update fast
delete

select slow

fresh data
small data
read and write (90%)

OLAP : DWH : De-Normalization : Dimension Modelling

historical data

OLAP : DWH : De-Normalization : Dimension Modelling

historical data
large data
read only(90%)

insert
update slow
delete

select fast

1. Dimension Table : (pk, string data, less data)

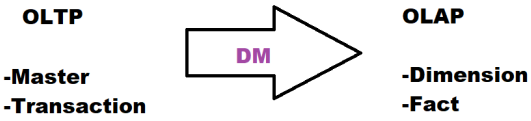
locationdim (lk, street, city, state, country)
dranchdim (brk, name, type)
itemdim (ik, name, brand, type, supplier)

2. Fact/Measure Table : (fk, numerical, large data)

salesfact (ik, lk, brk, qntsold, amount)

Dimension Modelling :

it is a process to convert the OLTP data into OLAP data in the form of dim and fact tables and maintain the relationship between them

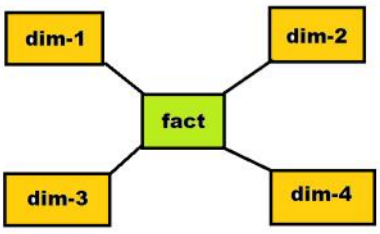


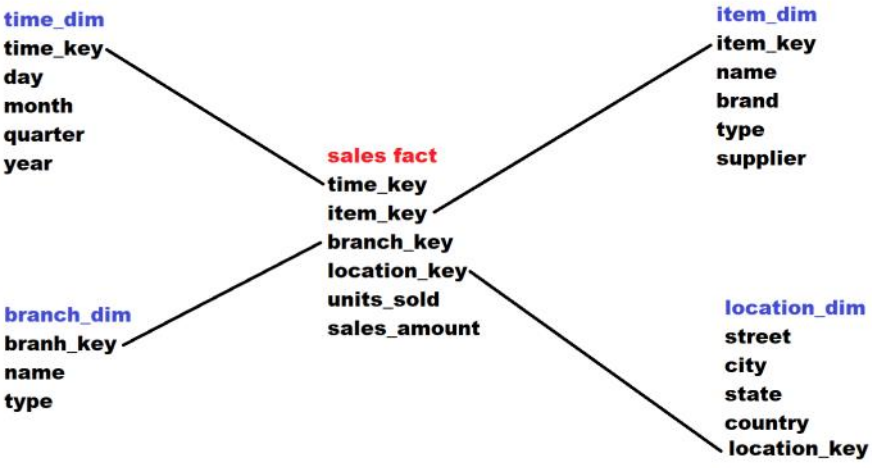
schema

24 February 2023 08:37

Schema

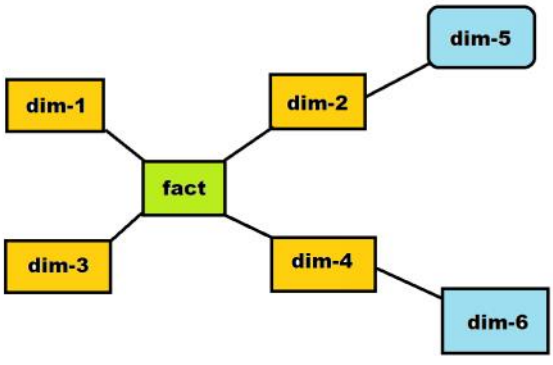
Star Schema : each dim table is connected to one or more fact tables via pk, fk relationship

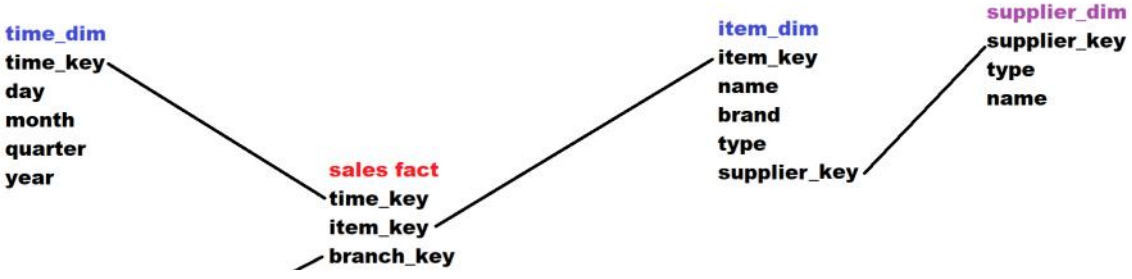


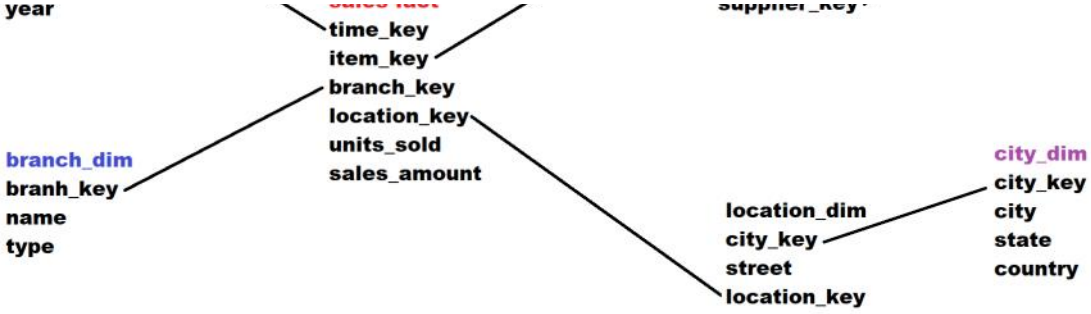


Snowflake Schema : it is a extension of star schema

there is possibility that a dim table can be connected to one or more dim tables as well






year
time_key
item_key
branch_key
location_key
units_sold
sales_amount
branch_dim
branh_key
name
type
location_dim
city_key
street
location_key
city_dim
city_key
city
state
country

types

24 February 2023 09:12

Types of Dimension tables

1. Slowly Changing Dimension :

id	name	country	nationality
1	ram	india	indian

type-1 : overwrite the previous data

id	name	country	nationality
1	ram	china	indian

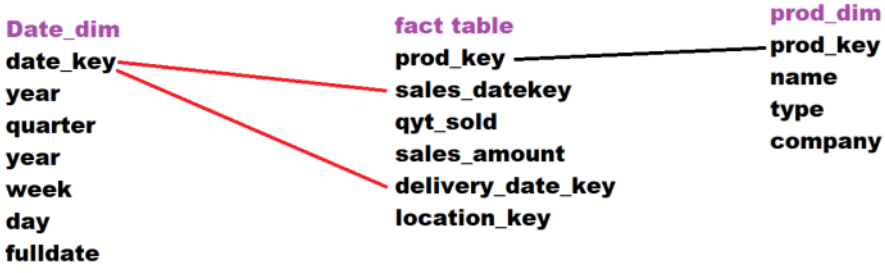
type-2 : add new row to store data

id	name	country	nationality
1	ram	india	indian
1	ram	china	indian

type-3 : add a new column

id	name	country	old_country	nationality
1	ram	china	india	indian

2. Roleplay dimension : a dim table having multiple relationships with fact table



3. Junk Dimension :

DIM_JUNK

JUNK_ID	TXN_CODE	COUPON_IND	PREPAY_IND
1	1	Y	Y
2	2	Y	Y
3	3	Y	Y
4	1	Y	N
5	2	Y	N

2	2	Y	Y
3	3	Y	Y
4	1	Y	N
5	2	Y	N
6	3	Y	N
7	1	N	Y
8	2	N	Y
9	3	N	Y
10	1	N	N
11	2	N	N
12	3	N	N

4. Confirmed dimension : dim table connected to more than one fact atble

Data Warehouse :

it is a centralized large database

=> subject oriented (finance, banking, manufacturing...)
=> Integrated (get the data from multiple data sources)
=> Non-Volatile (mostly read only data)
=> Time variant (different periods data)

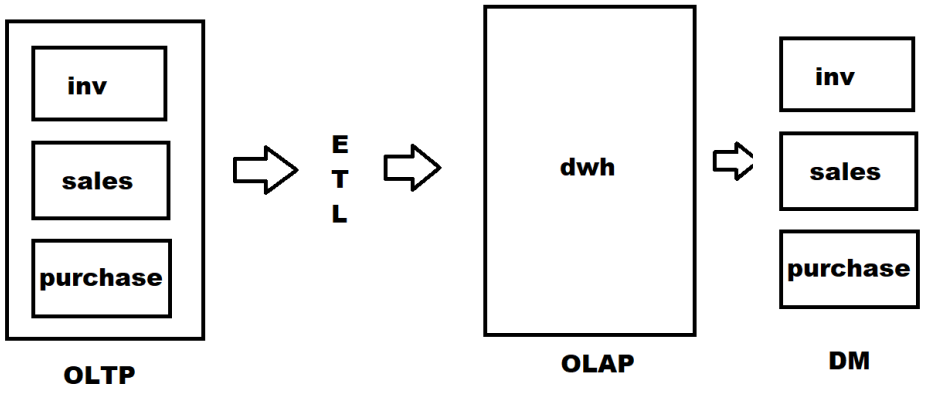
=> store and analyse => to take a proper decisions

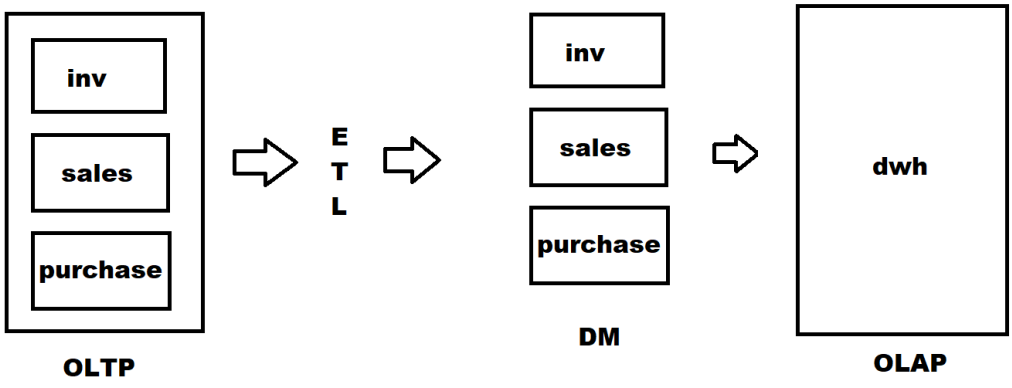
Data Mart :

=> a specific soubject oriented database (banking : loans, mutual, credit, ...)
=> collection of data marts = DWH

To build a dwh :

1. top down
2. bottom up





OLTP : Normalized tables ER Diagram

 # by increasing the no of tablkes we will be able to reduce the redundancy and imporve data integrity

by increasing the no of tablkes we will be able to reduce the redundancy and imporve data integrity

1. master table (pk, string data, less data)

pk
productmaster (pid, name, productdesc, productcategory)
1 pen ********** stationery

customermaster(cid, name, address, phone)
101 bob hyd 777777777

2. transaction tables (fk, numerical, large data)

fk fk
sales (pid, cid, dos, amount , tax, qty, dod)
1 101 4-5-2023 1000 10 1000 8-5-2023

insert
update => faster
delete

select => slower

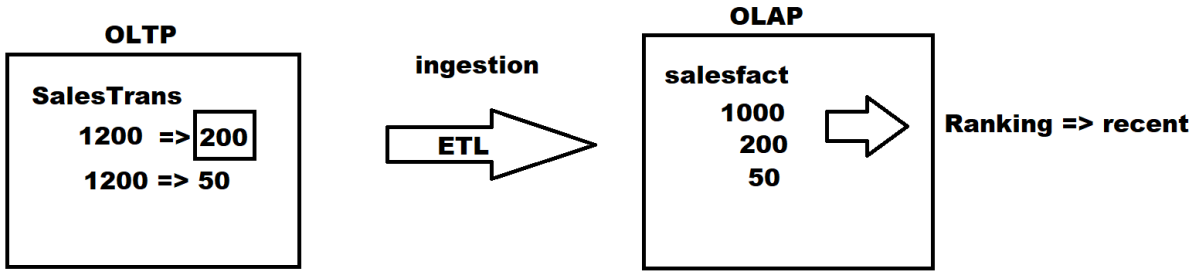
fresh data
small data
mostly writing

OLAP / DWH : Denormilization => reduce the no of table by increasing the redundancy

Dimension Modelling :

1. Dimension Table : (pk, string data, less)

2. Fact/Measure table : (fk, numerical, large data)

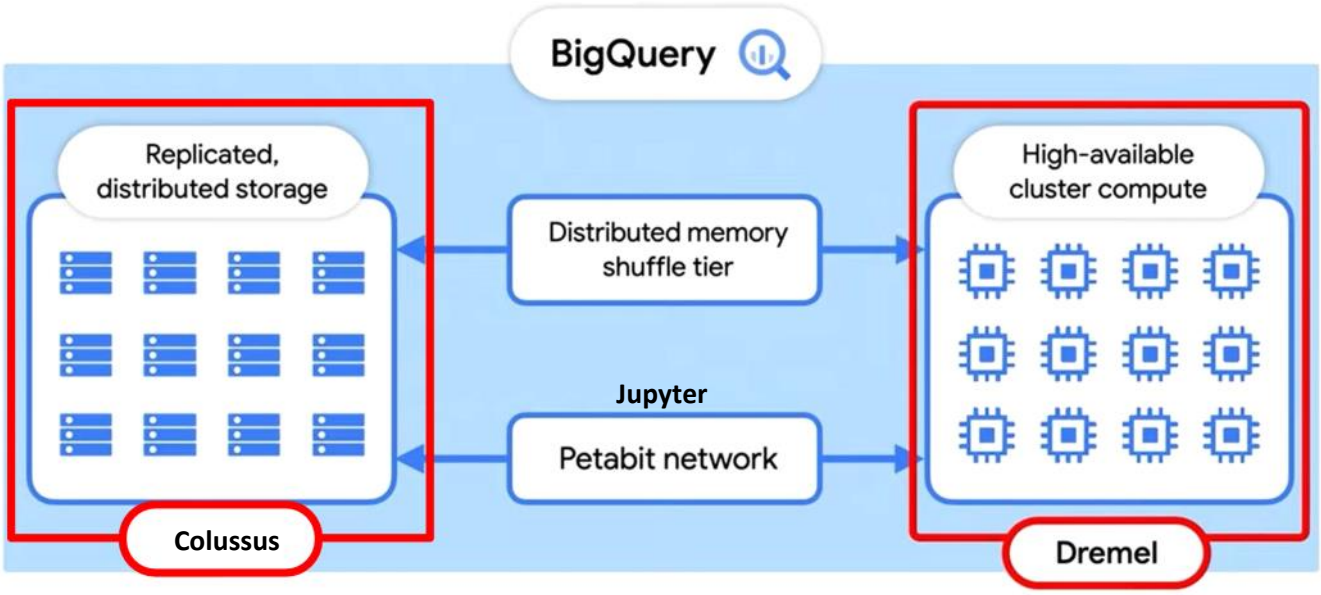


BigQuery

24 February 2023 21:09

1. Modern Data Warehouse
2. Server less platform
3. Highly scalable
4. To work with the large sets of data
5. Tabular/Structured (Schema define)
6. Comes up with built-in query engine
7. Capable of running SQL queries(standard SQL)
8. Performance
    a. Terabytes of data ==> seconds
    b. Petabytes of data ==> minutes
9. You get this performance without managing the infrastructure
10. There is no concept of indexes
11. Storage methods :
    a. BigQuery => columnar storage ==> both reads and updates
    b. Traditional => row storage ==> only for inserts
12. It provides both batch + real-time analytics
13. Working :
    a. projects
        (i) Datasets
            i. Tables
            ii. Views
            iii. Functions
            iv. Stored Functions
14. What we can do ?
    a. Collect (batch + streaming)
    b. Store
    c. Process/Transform
    d. Analyze

BigQuery Architecture :


BigQuery
Replicated, distributed storage
Distributed memory shuffle tier
High-available cluster compute
Jupyter
Petabit network
Colussus
Dremel

PROJECT_ID
- **DATASET_1**
  - **=> TABLE_1**
  - **=> TABLE_2**
  - **=> FUNCTION_1**
  - **=> VIEW_1**
  - **=> SP_1**
  - **=> MODEL_1**
- **DATASET_2**
  - **=> TABLE_1**
  - **=> TABLE_2**
  - **=> FUNCTION_1**
  - **=> VIEW_1**
  - **=> SP_1**
  - **=> MODEL_1**

```
-- PROJECT_ID
-- DATASET_ID = `PROJECT_ID.DATASET_NAME`
-- TABLE_ID = `PROJECT_ID.DATASET_NAME.TABLE_NAME` ;
```

Different ways of creating tables in BigQuery :

1. create a table from Editor and load data manually
2. create a table from Editor and Load data from GCS
3. create a table from Cloud Shell
4. create a table from Cloud Editor
5. create a table from Cloud Console
6. create a table from ETL services (Dataproc, Dataflow, Datafusion, ..)

Table Creations

► CREATE TABLE FROM EDITOR AND INSERT DATA MANUALLY :

```
CREATE OR REPLACE TABLE `gcp-evening-de-18.raw_dataset.customer`(
    id INT64,
    name STRING,
    dob DATE,
    email STRING,
    MaritalStatus BOOL,
    OrderDate TIMESTAMP,
```

```
    Amount FLOAT64,
    MobileNumber INT64
  );

INSERT INTO `gcp-evening-de-18.raw_dataset.customer` VALUES
  (1, 'John', '1994-12-2', 'john@gmail.com', TRUE, '2023-11-15 12:30:00', 1500.50, 123456789),
  (2, 'Jane Smith', '1985-08-20', 'jane.smith@example.com', FALSE, '2023-01-12 14:45:00', 1500.75, 9876543210),
  (3, 'Bob Johnson', '1978-03-03', 'bob.johnson@example.com', TRUE, '2023-01-15 10:00:00', 800.25, 5551112233);
```

► **CREATE TABLE FROM EDITOR AND LOAD DATA FROM GCS :**

```
CREATE OR REPLACE TABLE `gcp-evening-de-18.raw_dataset.customer_orders`(
  customer_id INT64,
  order_id INT64,
  order_date DATE,
  total_order_value FLOAT64
) ;
LOAD DATA INTO `gcp-evening-de-18.raw_dataset.customer_orders`
FROM FILES (
  format = 'CSV',
  uris = ['gs://bucket_test_213356/customer_orders.csv'],
  skip_leading_rows = 1
);
```

► **CREATE TABLE FROM SHELL : CSV, JSON, PARQUET, AVRO, ORC**

```
bq load \
--source_format=CSV \
--schema "student_id:INT64,subject:STRING,grade:INT64" \
--skip_leading_rows=1 \
sample_dataset.grades \
grades.csv
```

```
bq load \
--source_format=AVRO \
mydataset.mytable \
gs://mybucket/mydata.avro
```

```
bq load \
--source_format=PARQUET \
mydataset.mytable \
gs://mybucket/mydata.parquet
```

```
bq load \
--source_format=ORC \
mydataset.mytable \
gs://mybucket/mydata.orc
```

```
bq load \
--source_format=NEWLINE_DELIMITED_JSON \
mydataset.mytable \
gs://mybucket/mydata.json \
./myschema
```

► **CREATE TABLE FROM SDK : CSV, JSON, PARQUET, AVRO, ORC**

```
bq load --source_format=CSV --schema
"Name:STRING,Age:INT64,Location:STRING,Occupation:STRING,Email:STRING,Phone:STRING" --
skip_leading_rows=1 morning-batch-gcp-0666:raw_dataset.empTable2 "C:\Users\saidh\Desktop\GCP Trainings
```

\Data Engineer\PDE material\DataProc Project\sampledata\employee.csv"

Note : don’t use "\" (line continuation)

- **CREATE TABLE FROM CONSOLE:**
  - Bigquery console > select dataset > create table

Partitioning and Clustering

09 March 2023 22:10

https://cloud.google.com/bigquery/docs/creating-partitioned-tables#create_a_partitioned_table_from_a_query_result

Native Table :

bigquery = table structure (1) + data (2)

Table :

table structure :
rows => columns
schema => r & c

data : load

External Table :

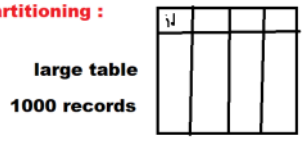
bigquery = table structure (1) external storage = data (2)

- gcs
- s3
- blob
- hdfs

Storage Optimization techniques :

1. Partitioning
2. Clustering

1. Partitioning :

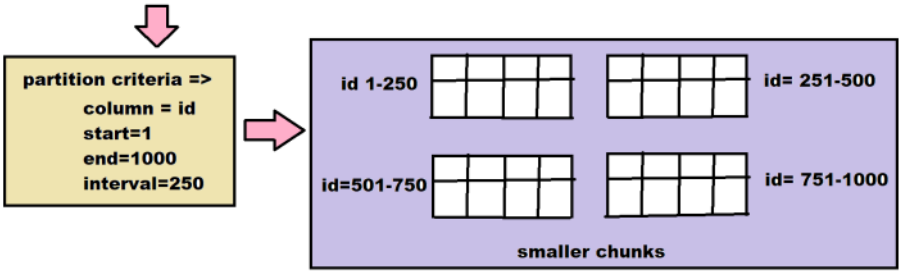


large table

1000 records

query => entire table scan => retrive

```
SELECT * FROM `employee`
where id = 251
```



deviding the larger table into smaller chunks is Partitiong

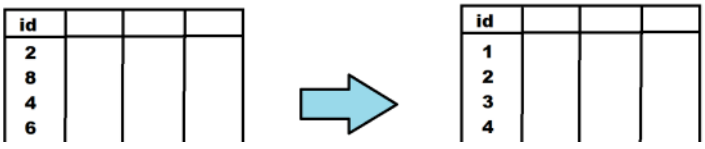
benefits :

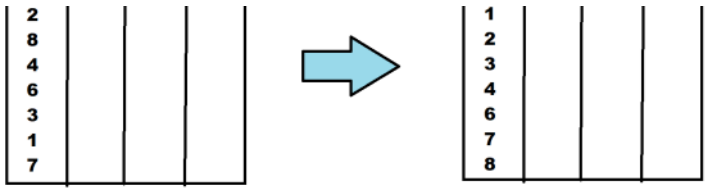
1. number of bytes processed will be less
2. control the cost
3. time taken is less

limitation :

1. partitioning column can only one
2. partitioning coluymn can be
   a. datetime
   b. integer
   c. ingestion time

2. Clustering : sorting the table data based on the column/s





1. sorting can be done on max 4 columns
2. there is no restriction on cluster column type
3. clustering can be applied on both Partion table and non-partition table

```
CREATE TABLE IF NOT EXISTS `project-bq-383715.raw_ds01.BOTH` (
  id int64,
  name string,
  dob date,
  salary float64,
  status bool,
  zipcode int64
)
  PARTITION BY dob
  CLUSTER BY id ;

INSERT INTO `project-bq-383715.raw_ds01.BOTH` (id, name, dob, salary,
status, zipcode)
VALUES
  (1, 'John', '1980-01-01', 5000.0, true, 12345),
  (2, 'Jane', '1990-03-15', 7000.0, true, 23456),
  (3, 'Bob', '1985-05-20', 6000.0, false, 34567),
  (4, 'Mary', '1995-07-31', 8000.0, true, 45678),
  (5, 'Joe', '2000-09-01', 9000.0, false, 56789),
  (6, 'Sarah', '1998-11-15', 10000.0, true, 67890),
  (7, 'David', '1992-02-28', 5500.0, true, 78901),
  (8, 'Emily', '1989-04-10', 6500.0, false, 89012),
  (9, 'Peter', '1987-06-25', 7500.0, true, 90123),
  (10, 'Lucy', '1983-08-10', 8500.0, false, 12345),
  (11, 'Tom', '1996-10-05', 9500.0, true, 23456),
  (12, 'Anna', '1984-12-20', 10500.0, false, 34567),
  (13, 'Mark', '1994-02-10', 11500.0, true, 45678),
  (14, 'Karen', '1991-04-05', 12500.0, false, 56789),
  (15, 'Mike', '2001-06-20', 13500.0, true, 67890),
  (16, 'Lisa', '1999-08-05', 14500.0, false, 78901),
  (17, 'Tim', '1993-10-15', 15500.0, true, 89012),
  (18, 'Julia', '1986-12-30', 16500.0, false, 90123),
  (19, 'Chris', '1982-02-28', 17500.0, true, 12345),
  (20, 'Amy', '1997-04-15', 18500.0, false, 23456);
```

Clustered and Partitioned Tables

Orders table
Not Clustered; Not partitioned

Order_Date	Country	Status
2022-08-02	US	Shipped
2022-08-04	JP	Shipped
2022-08-05	UK	Canceled
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-05	US	Processing
2022-08-04	JP	Processing
2022-08-04	KE	Shipped
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-05	JP	Canceled
2022-08-06	UK	Processing

Orders table
Clustered by Country; Not partitioned

Order_Date	Country	Status
2022-08-04	JP	Shipped
2022-08-04	JP	Processing
2022-08-05	JP	Canceled
2022-08-06	JP	Processing
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-04	KE	Shipped
2022-08-02	KE	Shipped
2022-08-05	UK	Processing
2022-08-06	UK	Canceled
2022-08-02	UK	Canceled
2022-08-06	UK	Processing

Orders table
Clustered by Country; Partitioned by Order_Date (Daily)

	Order_Date	Country	Status
Partition: 2022-08-02	2022-08-02	KE	Shipped
	2022-08-02	KE	Canceled
Clusters: Country	2022-08-02	UK	Processing
	2022-08-02	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-04	2022-08-04	JP	Shipped
	2022-08-04	JP	Processing
Cluster: Country	2022-08-04	KE	Shipped
	2022-08-04	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-05	2022-08-05	JP	Canceled
	2022-08-05	UK	Canceled
	2022-08-05	US	Shipped

2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-05	JP	Canceled
2022-08-06	UK	Processing
2022-08-05	US	Shipped
2022-08-06	JP	Processing
2022-08-02	KE	Shipped
2022-08-04	US	Shipped

2022-08-05	UK	Processing
2022-08-06	UK	Canceled
2022-08-02	UK	Canceled
2022-08-06	UK	Processing
2022-08-02	US	Shipped
2022-08-05	US	Processing
2022-08-05	US	Shipped
2022-08-04	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-05	2022-08-05	JP	Canceled
	2022-08-05	UK	Canceled
Cluster: Country	2022-08-05	US	Shipped
	2022-08-05	US	Processing

	Order_Date	Country	Status
Partition: 2022-08-06	2022-08-06	JP	Processing
	2022-08-06	KE	Shipped
Cluster: Country	2022-08-06	UK	Canceled
	2022-08-06	UK	Processing

what is partitioning :

deviding a large table into smaller chunks
only 1 column as pc
int, datetime columns only (string, float)
2000 partitions/per table

what is clustering :

sorting the tables based on column
max 4 columns as cc
int, string, datetime (float)

advantages :

1. query performance
2. cost reduction by reducing the bytes processed
3. less time taken

Partition Types :

```
-- Partition by Date
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable1`(
    event_id int64,
    event_date Date
    )
    PARTITION BY event_date   -- by date
    CLUSTER BY event_id;

-- Partition by Datetime (HOUR, DAY, MONTH, YEAR)
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable2`(
    event_id int64,
    event_date Datetime
    )
    PARTITION BY DATETIME_TRUNC(event_date,HOUR)
    CLUSTER BY event_id;

    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable3`(
    event_id int64,
    event_date Datetime
    )
    PARTITION BY DATE(event_date)
    CLUSTER BY event_id;

-- Partition by Timestamp (HOUR, DAY, MONTH, YEAR)
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable4`(
    event_id int64,
    event_date Timestamp
    )
    PARTITION BY TIMESTAMP_TRUNC(event_date, HOUR)
    CLUSTER BY event_id;
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable5`(
    event_id int64,
    event_date Timestamp
```

```
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable5`(
    event_id int64,
    event_date Timestamp
    )
    PARTITION BY DATE(event_date)
    CLUSTER BY event_id;

--Partition by integer range
    CREATE TABLE `gcp-batch-
    morning.sampleDb.eventTable6`(
    event_id int64,
    event_date Date
    )
    PARTITION BY RANGE_BUCKET(event_id,
    GENERATE_ARRAY(1,1000,100))
    CLUSTER BY event_id;
```

Nested and Repeated fields

01 March 2023 08:56

Nested Table :

- **To create a column with nested data, set the data type of the column to RECORD/STRUCT in the schema.**
- **A RECORD can be accessed as a STRUCT type in GoogleSQL. A STRUCT is a container of ordered fields.**

Repeated Table:

- **To create a column with repeated data, set the mode of the column to REPEATED/ARRAY in the schema.**
- **A repeated field can be accessed as an ARRAY type in GoogleSQL.**

Both :

- **A RECORD column can have REPEATED mode, which is represented as an array of STRUCT types.**
- **Also, a field within a record can be repeated, which is represented as a STRUCT that contains an ARRAY.**
- **An array cannot contain another array directly.**

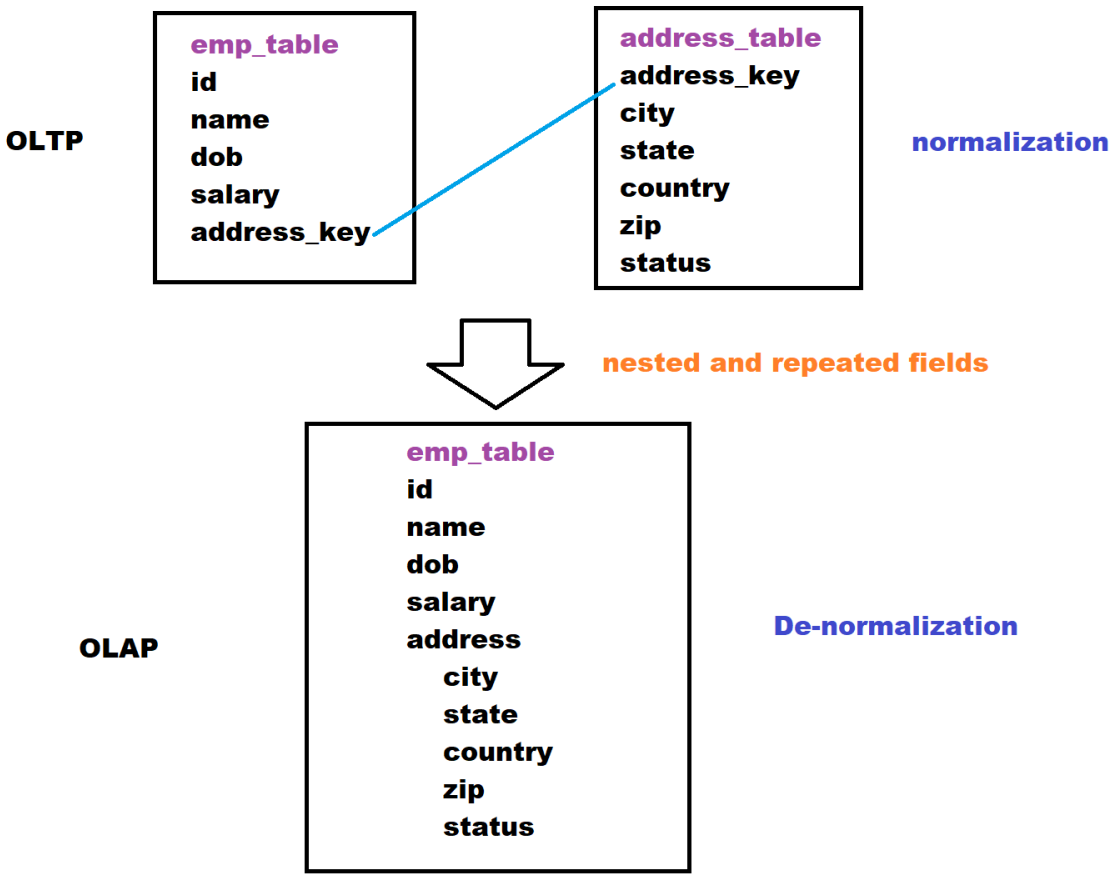
Note : A schema cannot contain more than 15 levels of nested RECORD types

Best practice: Use nested and repeated fields to de-normalize data storage and increase query performance.

- **Denormalization is a common strategy for increasing read performance for relational datasets that were previously normalized. The recommended way to de-normalize data in BigQuery is to use nested and repeated fields. It's best to use this strategy when the relationships are hierarchical and frequently queried together, such as in parent-child relationships.**

When to use nested and repeated columns ?

- **BigQuery performs best when your data is denormalized. Rather than preserving a relational schema such as a star or snowflake schema, de-normalize your data and take advantage of nested and repeated columns. Nested and repeated columns can maintain relationships without the performance impact of preserving a relational (normalized) schema.**




```
-- Create the nested table
CREATE TABLE `test-project-1278954.raw_ds.nestedTable1` (
  customer_id INT64,
  customer_name STRING,
  customer_info STRUCT<
    email STRING,
    phone INT64
  >,
  order_history STRUCT <
    order_id INT64,
    order_date DATETIME,
    total_amount INT64
  >
);


-- Insert sample data
INSERT INTO `test-project-1278954.raw_ds.nestedTable2`
VALUES
  (1, 'John Doe', STRUCT('john@example.com', 1234567890), STRUCT(101, '2024-02-13 08:00:00', 500)),
  (1, 'John Doe', STRUCT('john@example.com', 1234567890), STRUCT(104, '2024-02-13 09:00:00', 1500)),
  (1, 'John Doe', STRUCT('john@example.com', 1234567890), STRUCT(105, '2024-02-13 10:00:00', 2500)),
  (2, 'Jane Smith', STRUCT('jane@example.com', 9876543210), STRUCT(102, '2024-02-12 09:30:00', 750)),
  (3, 'Alice Johnson', STRUCT('alice@example.com', 1112223333), STRUCT(103, '2024-02-11 10:45:00', 1000)),
  (3, 'Alice Johnson', STRUCT('alice@example.com', 1112223333), STRUCT(106, '2025-02-11 18:45:00', 12000));
```

SCHEMA | DETAILS | PREVIEW | LINEAGE | DATA PROFILE | DATA QUALITY

Row	customer_id	customer_name	customer_info.email	custo... phone	ord... order_id	order_history.order_date	ctotal_amount
1	1	John Doe	john@example.com	1234567890	105	2024-02-13T10:00:00	2500
2	1	John Doe	john@example.com	1234567890	101	2024-02-13T08:00:00	500
3	1	John Doe	john@example.com	1234567890	104	2024-02-13T09:00:00	1500
4	2	Jane Smith	jane@example.com	9876543210	102	2024-02-12T09:30:00	750
5	3	Alice Johnson	alice@example.com	1112223333	103	2024-02-11T10:45:00	1000
6	3	Alice Johnson	alice@example.com	1112223333	106	2025-02-11T18:45:00	12000

To avoid the above duplication, we can go for Repeated fields also (while create the table)

```
-- Create the nested and repeated table

CREATE TABLE `test-project-1278954.raw_ds.nestedTable2` (
  customer_id INT64,
  customer_name STRING,
  customer_info STRUCT<
    email STRING,
    phone INT64
  >,
  order_history ARRAY <
    STRUCT <
      order_id INT64,
      order_date DATETIME,
      total_amount INT64
    >
  >
);

-- Insert sample data
INSERT INTO `test-project-1278954.raw_ds.nestedTable3`
VALUES
  (1, 'John Doe', STRUCT('john@example.com', 1234567890), [STRUCT(101, DATETIME'2024-02-13 08:00:00', 500), STRUCT(104, DATETIME'2024-02-13 09:00:00', 1500),STRUCT(105, DATETIME'2024-02-13 10:00:00', 2500)]),
  (2, 'Jane Smith', STRUCT('jane@example.com', 9876543210), [STRUCT(102, DATETIME'2024-02-12 09:30:00', 750)]),
  (3, 'Alice Johnson', STRUCT('alice@example.com', 1112223333), [STRUCT(103, DATETIME'2024-02-11 10:45:00', 1000), STRUCT(106, DATETIME'2025-02-11 18:45:00', 12000)]);
```

Loading JSON From Cloud Shell :

schema

sample_data

BQ LOAD :

- **bq load --schema=schema.json --source_format=NEWLINE_DELIMITED_JSON gcp-batch-morning:landing.jsonExample sample_data.json**

Views

12 July 2023 20:59

In Google BigQuery, you can create and work with views, materialized views, and authorized views to simplify data access, optimize query performance, and control data access permissions.

Views:

- What is a View: A view in BigQuery is a virtual table that is defined by a SQL query. It does not store data itself but provides a logical representation of data from one or more tables or other views.

- Use Cases: Views are useful for simplifying complex queries, abstracting underlying table structures, and providing a consistent interface for querying data.

- Key Features:
  - Views are read-only and do not store data; they execute the underlying query each time they are referenced.
  - Views can reference other views, allowing for a layered approach to data abstraction.
  - Views can be used to restrict access to specific columns or rows of data.

- Syntax to Create a View:

```
CREATE OR REPLACE VIEW project.dataset.view_name AS
SELECT column1, column2, ...
FROM project.dataset.table;
```

Materialized Views :

- What is a Materialized View: A materialized view is a precomputed summary table that stores the result of a query, making it faster to retrieve and ideal for frequently accessed or complex data.

- Use Cases: Materialized views are used to improve query performance for large datasets by precomputing and caching query results.

- Key Features:
  - Materialized views are periodically refreshed to keep the data up-to-date.
  - They are particularly useful for aggregations and summaries.
  - Materialized views can significantly speed up query execution times, but they also consume storage space.

- Syntax to Create a Materialized View :

```
CREATE MATERIALIZED VIEW project.dataset.mv_name
OPTIONS (
  partition_expiration_days = 60,
  refresh_interval_sec = 3600
) AS
SELECT column1, column2, ...
FROM project.dataset.table;
```

Authorized Views:

- What is an Authorized View: An authorized view in BigQuery is a view with defined row-level access control policies. It allows you to restrict data access based on user or group identity.

- Use Cases: Authorized views are used to control and restrict data access based on user roles and permissions, ensuring that users only see the data they are authorized to view.

- Key Features:
  - Authorized views can be used in combination with Google Cloud Identity and Access Management (IAM) to control who can access specific rows of data within a view.
  - Access to rows in an authorized view is determined by matching the user's identity to policies defined in IAM.

- Syntax to Create an Authorized View:

```
CREATE OR REPLACE VIEW project.dataset.authorized_view AS
SELECT column1, column2, ...
FROM project.dataset.table
WHERE user_id IN (SELECT user_id FROM project.dataset.user_roles);
```

Assignment-1

28 April 2023 08:54

```
--order table
create table if not exists `project-bq-383715.raw_ds01.orders` (
  order_id int64,
  customer_id int64,
  order_date date,
  order_total float64
);

insert into `project-bq-383715.raw_ds01.orders` values
  (1, 1001, '2022-01-01', 50.00),
  (2, 1002, '2022-01-02', 75.00),
  (3, 1003, '2022-01-03', 100.00),
  (4, 1001, '2022-01-04', 60.00),
  (5, 1002, '2022-01-05', 85.00),
  (6, 1003, '2022-01-06', 110.00);

--create customers table
create table if not exists `project-bq-383715.raw_ds01.customers` (
  customer_id int64,
  customer_name string,
  customer_email string,
  customer_age int64,
  customer_gender string
) ;

insert into `project-bq-383715.raw_ds01.customers` values
  (1001, 'John Smith', 'john.smith@example.com', 35, 'male'),
  (1002, 'Jane Doe', 'jane.doe@example.com', 28, 'female'),
  (1003, 'Bob Johnson', 'bob.johnson@example.com', 42, 'male'),
  (1004, 'Sara Lee', 'sara.lee@example.com', 51, 'female'),
  (1005, 'Tom Johnson', 'tom.johnson@example.com', 31, 'male');

--create sales table
create table if not exists `project-bq-383715.raw_ds01.sales` (
  sales_id int64,
  order_id int64,
  product_id int64,
  quantity int64,
  price float64
) ;

insert into `project-bq-383715.raw_ds01.sales` values
  (1, 1, 1001, 2, 10.00),
  (2, 1, 1002, 1, 30.00),
  (3, 2, 1003, 3, 5.00),
  (4, 3, 1002, 2, 25.00),
  (5, 3, 1004, 1, 50.00),
  (6, 4, 1005, 4, 5.00),
  (7, 5, 1001, 1, 85.00),
  (8, 6, 1002, 2, 50.00);

--create products tables
create table if not exists `project-bq-383715.raw_ds01.products` (
  products_id int64,
  product_name string,
  product_description string,
  product_category string
) ;

insert into `project-bq-383715.raw_ds01.products` values
  (1001, 'Product A', 'A description', 'Category A'),
```

```
  (1002, 'Product B', 'B description', 'Category A'),
  (1003, 'Product C', 'C description', 'Category B'),
  (1004, 'Product D', 'D description', 'Category B'),
  (1005, 'Product E', 'E description', 'Category C');

--create employees tables
create table if not exists `project-bq-383715.raw_ds01.employees` (
  employees_id int64,
  employees_name string,
  employees_email string,
  employees_role string,
  department_id int64
) ;

insert into `project-bq-383715.raw_ds01.employees` values
  (1, 'Bob Johnson', 'bob.johnson@example.com', 'Manager', 1),
  (2, 'Jane Doe', 'jane.doe@example.com', 'Sales Rep', 2),
  (3, 'John Smith', 'john.smith@example.com', 'Sales Rep', 2),
  (4, 'Sara Lee', 'sara.lee@example.com', 'Customer Service', 3),
  (5, 'Tom Johnson', 'tom.johnson', 'Customer Service', 3)
```

Questions :

1. **Join the orders, customers, and sales tables to get a report of all sales, including the customer and product information**
2. **Calculate the average age of customers by product category**
3. **Calculate the top 5 products by total sales**
4. **Calculate the total revenue for each year and month:**
5. **Calculate the top 5 customers by total number of orders**
6. **Calculate the percentage of sales for each product category**
7. **Calculate the top 2 customers BY total sales, WITH AT least 50 sales:**
8. **Calculate the total sales for each region and year**

Answers :

Join the orders, customers, and sales tables to get a report of all sales, including the customer and product information:

```
SELECT o.order_id, o.order_date, c.customer_name, p.product_name, s.quantity, s.price
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN sales s ON o.order_id = s.order_id
JOIN products p ON s.product_id = p.product_id
ORDER BY o.order_id;
```

Calculate the total sales by product category:

```
SELECT p.product_category, SUM(s.quantity * s.price) AS total_sales
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY p.product_category;
```

Calculate the average age of customers by product category:

```
SELECT p.product_category, AVG(c.customer_age) AS average_age
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN sales s ON o.order_id = s.order_id
JOIN products p ON s.product_id = p.product_id
GROUP BY p.product_category;
```

Calculate the total sales for each employee:

```
SELECT e.employee_id, e.employee_name, SUM(s.quantity * s.price) AS total_sales
FROM employees e
JOIN orders o ON e.employee_id = o.employee_id
JOIN sales s ON o.order_id = s.order_id
GROUP BY e.employee_id, e.employee_name;
```

Calculate the top 5 products by total sales:

```
SELECT p.product_name, SUM(s.quantity * s.price) AS total_sales
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_sales DESC
LIMIT 5;
```

Calculate the total revenue for each year and month:

```
SELECT EXTRACT(YEAR FROM o.order_date) AS year,
```

```
    EXTRACT(MONTH FROM o.order_date) AS month,
    SUM(s.quantity * s.price) AS revenue
FROM orders o
JOIN sales s ON o.order_id = s.order_id
GROUP BY year, month
ORDER BY year, month;

Calculate the top 5 customers by total number of orders:
SELECT c.customer_name, COUNT(DISTINCT o.order_id) AS num_orders
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
ORDER BY num_orders DESC
LIMIT 5;

Calculate the percentage of sales for each product category:
WITH total_sales_by_category AS (
  SELECT p.product_category, SUM(s.quantity * s.price) AS total_sales
  FROM sales s
  JOIN products p ON s.product_id = p.product_id
  GROUP BY p.product_category
),
total_sales AS (
  SELECT SUM(total_sales) AS total_sales
  FROM total_sales_by_category
)
SELECT
  t.product_category,
  ROUND(t.total_sales / ts.total_sales * 100, 2) AS sales_percentage
FROM
  total_sales_by_category t,
  total_sales ts
ORDER BY sales_percentage DESC;

  --Calculate the top 2 customers BY total sales,WITH AT least 50 sales:
WITH
  customer_sales AS (
  SELECT
    c.customer_name,
    SUM(s.quantity * s.price) AS total_sales
  FROM
    `project-bq-383715.raw_ds01.sales` s
  LEFT JOIN
    `project-bq-383715.raw_ds01.orders` o
  ON
    s.order_id=o.order_id
  LEFT JOIN
    `project-bq-383715.raw_ds01.customers` c
  ON
    o.customer_id=c.customer_id
  GROUP BY
    c.customer_name
  ORDER BY
    total_sales DESC
)

SELECT  *
  FROM
  customer_sales
WHERE
  total_sales >= 50
    LIMIT 2

Calculate the total sales for each region and year:
SELECT r.region_name, EXTRACT(YEAR FROM o.order_date) AS year, SUM(s.quantity * s.price) AS total_sales
FROM regions r
JOIN employees e ON r.region_id = e.region_id
JOIN orders o ON e.employee_id = o.employee_id
JOIN sales s ON o.order_id = s.order_id
GROUP BY r.region_name, year
ORDER BY r.region_name, year;
```

Case study-1 : Spotify

Tuesday, October 17, 2023 8:58 PM

dataset

Load from SDK into BQ :

bq load --
schema="playlist_url:string,year:int64,track_id:string,track_name:string,track_popularity:int64,album:string,artist_id:string,artist_name:string,artist_genres:string,artist_popularity:int64,danceability:float64,energy:float64,key:int64,loudness:float64,mode:int64,speechiness:float64,acousticness:float64,instrumentalness:float64,liveness:float64,valence:float64,tempo:float64,duration_ms:int64,time_signature:int64" --source_format=CSV --skip_leading_rows=1 bronze_ds.spotify_data "C:\Users\saidh\Desktop\GCP Trainings\Data Engineer\PDE material\Bigquery\case-study-1\dataset_playlist_2010to2022.csv"

Note : change the file path, dataset

```
CREATE OR REPLACE TABLE
  `gcp-batch-morning.spotify_ms.masterTable` AS
SELECT
  DISTINCT playlist_url,
  year,
  track_id,
  track_name,
  track_popularity,
  album,
  artist_id,
  artist_name,
  artist_genres,
  artist_popularity,
  danceability,
  energy,
  KEY,
  loudness,
  mode,
  speechiness,
  acousticness,
  instrumentalness,
  liveness,
  valence,
  tempo,
  duration_ms,
  time_signature
FROM
  `gcp-batch-morning.spotify_raw.spotifyTable`


 --based ON the popularity : top10 songs ??
SELECT track_name, track_popularity
FROM `gcp-evening-de-18.mysqldb.spotify_data`
ORDER BY track_popularity DESC
LIMIT 10;

--top 10 artists (Number OF songs ON Spotify FOR each artist)
SELECT artist_name, count(track_id) as no_of_songs
FROM `gcp-evening-de-18.mysqldb.spotify_data`
GROUP BY artist_name
ORDER BY no_of_songs DESC
LIMIT 10;

--Total number of songs on spotify based on year
SELECT year, count(track_id) as no_of_songs
FROM `gcp-evening-de-18.mysqldb.spotify_data`
group by year
order by no_of_songs desc;

--Top song for each year (2000-2022) based on popularity
with ranksongs as (
  SELECT
    track_name,
    year,
    track_popularity,
    rank() over(partition by year order by track_popularity desc) as rnk
```

```
  FROM `gcp-evening-de-18.mysqldb.spotify_data`
)
SELECT
  track_name,
  year,
  track_popularity
FROM
  ranksongs
WHERE
  rnk =1;
```

Analysis based on Tempo :

```
--3. Top 10 songs with highest tempo
SELECT
  track_name,
  tempo
FROM `gcp-batch-morning.spotify_ms.masterTable`
order by tempo desc
limit 10;

-- Based on the average tempo categorize the tracks
WITH TempoStats AS (
  SELECT AVG(tempo) AS avg_temp FROM `gcp-batch-morning.spotify_ms.masterTable`
)
SELECT
  track_name,
  tempo,
  CASE
    WHEN tempo > t.avg_temp THEN 'Above average Temp'
    WHEN tempo = t.avg_temp THEN 'Average Temp'
    ELSE 'Below average Temp'
  END AS TempoAverage
FROM
  `gcp-batch-morning.spotify_ms.masterTable`,TempoStats t;


--4. Number of songs with different tempo range
SELECT
  COUNTIF(tempo BETWEEN 60.00 AND 100.00) AS Classical_Music,
  COUNTIF(tempo BETWEEN 100.001 AND 120.00) AS Modern_Music,
  COUNTIF(tempo BETWEEN 120.001 AND 150.01) AS Dance_Music,
  COUNTIF(tempo >= 150.01) AS HighTemp_Music
FROM `gcp-batch-morning.spotify_ms.masterTable`;
```

Analysis based on energy :

1. **Avg energy**

1. **Based on the average energy categorize the tracks**

1. **Top 10 songs with highest energy(above, below, avg)**

1. **Number of songs with different energy range**
    i. **classic music => 0.1-0.3**
    ii. **Moderate music => 0.3-0.6**
    iii. **Energetic music => >0.6**

Analysis based on Danceability :

1. **Avg Danceability**

1. **Based on the average Danceability categorize the tracks**

1. **Top 10 songs with highest Danceability**

1. **Number of songs with different Danceability range**

Analysis based on Loudness :

1. **Avg Loudness**

1. **Based on the average Loudness categorize the tracks**

1. **Top 10 songs with highest Loudness**

1. **Number of songs with different Loudness range**

Analysis based on Valence :

1. **Avg Valence**

1. **Based on the average Valence categorize the tracks**

1. **Top 10 songs with highest Valence**

1. **Number of songs with different Valence range**

Analysis based on Speechiness :

1. **Avg Speechiness**

1. **Based on the average Speechiness categorize the tracks**

1. **Top 10 songs with highest Speechiness**

1. **Number of songs with different Speechiness range**

INSIGHTS

- **From the above analysis, It is observed that Drake had released the most number of songs on Spotify between 2010-2022.**
- **Cruel Song from the album Lover by Taylor Swift was the most popular song on Spotify between 2010-2022.**
- **Taylor Swift is probably the most popular singer as she had 3 songs among the top 10 songs in Spotify.**
- **Majority of songs on Spotify have high tempo that means those songs have more energy and Speechiness.**
- **Majority of songs on Spotify aren't danceable.**

RECOMMENDATIONS

- **In Recent times, there is a rise in calm and slowed songs and Spotify needs to add such songs to be with the trend.**
- **Spotify needs to balance the valence songs.**
- **Spotify needs to add more songs with less/below average loudness as they are starting to get in trend.**

- **There is great rise in the number of people going for Dancing and Spotify needs to add more songs which are danceable.**
- **Spotify needs to add more rap/high speechiness songs, as there is decline in number of rap songs over the years.**

Casestudy-2 : **Social Media**

Friday, December 8, 2023 4:29 PM

Social Media Database Project :

- Through this project we are creating a basic structure of social media database which could easily be connected to frontend interface.
- Here we are managing the data of multiple users, their followers , interests and public activity on the social media platform which includes post likes ,comments , comment likes , hashtag followed ,bookmarks and many more...
- Through this analysis we have shown a clear cut description of connection and inter relation between different activities on social media
- Here we have tried to fetch and store the data in its true storage form into this database like (img/videos in url)

Sample data : SQL files

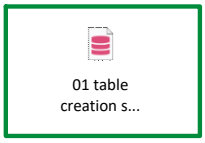
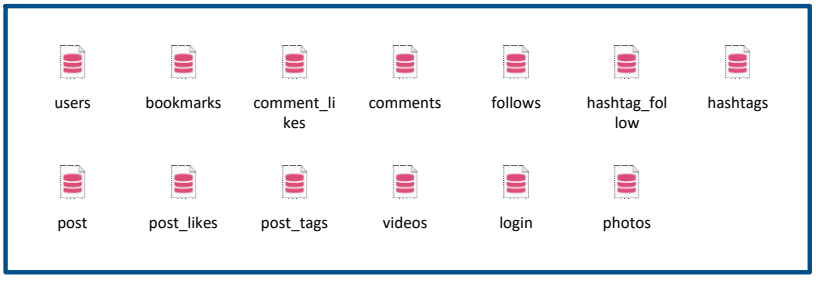


Table creation script



Insert scripts with sample data

Data Analysis as per Use Cases :

```
-- 1. Location of User
SELECT *
FROM `rich-tine-405803.raw_dataset.post`
WHERE location IN ('agra', 'maharashtra', 'west bengal');

-- 2. Most Followed Hashtag
SELECT
  hashtag_name AS `Hashtags`,
  COUNT(b.hashtag_id) AS `Total_Follows`
FROM `rich-tine-405803.raw_dataset.hashtags` a
JOIN `rich-tine-405803.raw_dataset.hashtag_follow` b
  ON a.hashtag_id = b.hashtag_id
GROUP BY hashtag_name
ORDER BY COUNT(b.hashtag_id) DESC
LIMIT 5;

-- 3. Most Used Hashtags
SELECT
  a.hashtag_name AS `Trending Hashtags`,
  COUNT(b.hashtag_id) AS `Times Used`
FROM `rich-tine-405803.raw_dataset.hashtags` a
JOIN `rich-tine-405803.raw_dataset.post_tags` b
  ON a.hashtag_id = b.hashtag_id
GROUP BY a.hashtag_name
ORDER BY COUNT(b.hashtag_id) DESC
LIMIT 10;

-- 4. Most Inactive User
SELECT
  user_id,
  username AS `Most Inactive User`
```

```
FROM
  `rich-tine-405803.raw_dataset.users`
WHERE
  user_id NOT IN (SELECT user_id FROM `rich-tine-405803.raw_dataset.post`);

-- 5. Most Likes Posts
SELECT
  a.user_id,
  a.post_id,
  COUNT(a.post_id)
FROM
  `rich-tine-405803.raw_dataset.post_likes` a
JOIN
  `rich-tine-405803.raw_dataset.post` b
ON
  b.post_id = a.post_id
GROUP BY
  a.post_id,a.user_id
ORDER BY
  COUNT(a.post_id) DESC
Limit 10;

-- 6. Average post per user
SELECT
  ROUND((COUNT(post_id) / COUNT(DISTINCT user_id)), 2) AS `Average Post per User`
FROM
  `rich-tine-405803.raw_dataset.post`;

-- 7. No. of login by per user
SELECT
  a.user_id,
  email,
  username,
  b.login_id AS `login_number`
FROM
  `rich-tine-405803.raw_dataset.users` a
JOIN
  `rich-tine-405803.raw_dataset.login` b
ON
  a.user_id = b.user_id;

-- 8. User who liked every single post
SELECT
  username,
  COUNT(*) AS `num_likes`
FROM
  `rich-tine-405803.raw_dataset.users` users
JOIN
  `rich-tine-405803.raw_dataset.post_likes` post_likes
ON
  users.user_id = post_likes.user_id
GROUP BY
  username
HAVING
  num_likes = (SELECT COUNT(*) FROM `rich-tine-405803.raw_dataset.post`);

-- 9. User Never Comment
SELECT
  user_id,
  username AS `User Never Comment`
FROM
  `rich-tine-405803.raw_dataset.users`
WHERE
  user_id NOT IN (SELECT user_id FROM `rich-tine-405803.raw_dataset.comments`);

-- 10. User who commented on every post
SELECT
  username,
  COUNT(*) AS `num_comment`
```

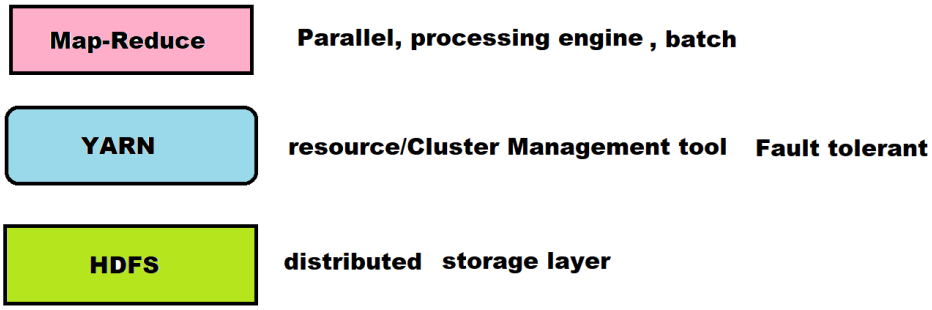
```
FROM
  `rich-tine-405803.raw_dataset.users` users
JOIN
  `rich-tine-405803.raw_dataset.comments` comments
ON
  users.user_id = comments.user_id
GROUP BY
  username
HAVING
  num_comment = (SELECT COUNT(*) FROM `rich-tine-405803.raw_dataset.comments`);

-- 11. User Not Followed by Anyone
SELECT
  user_id,
  username AS `User Not Followed by Anyone`
FROM
  `rich-tine-405803.raw_dataset.users`
WHERE
  user_id NOT IN (SELECT followee_id FROM `rich-tine-405803.raw_dataset.follows`);

-- 12. User Not Following Anyone
SELECT
  user_id,
  username AS `User Not Following Anyone`
FROM
  `rich-tine-405803.raw_dataset.users`
WHERE
  user_id NOT IN (SELECT follower_id FROM `rich-tine-405803.raw_dataset.follows`);

-- 13. Posted more than 5 times
SELECT
  user_id,
  COUNT(user_id) AS `post_count`
FROM
  `rich-tine-405803.raw_dataset.post`
GROUP BY
  user_id
HAVING
  post_count > 5
ORDER BY
  COUNT(user_id) DESC;

-- 14. who have Followers > 40
SELECT
  followee_id,
  COUNT(follower_id) AS `follower_count`
FROM
  `rich-tine-405803.raw_dataset.follows`
GROUP BY
  followee_id
HAVING
  follower_count > 40
ORDER BY
  COUNT(follower_id) DESC;

-- 15. Any specific word in comment
SELECT
  *
FROM
  `rich-tine-405803.raw_dataset.comments`
WHERE
  comment_text like '%good%';

-- 16. Longest captions in post
SELECT
  user_id,
  caption,
  LENGTH(post.caption) AS `caption_length`
FROM
```

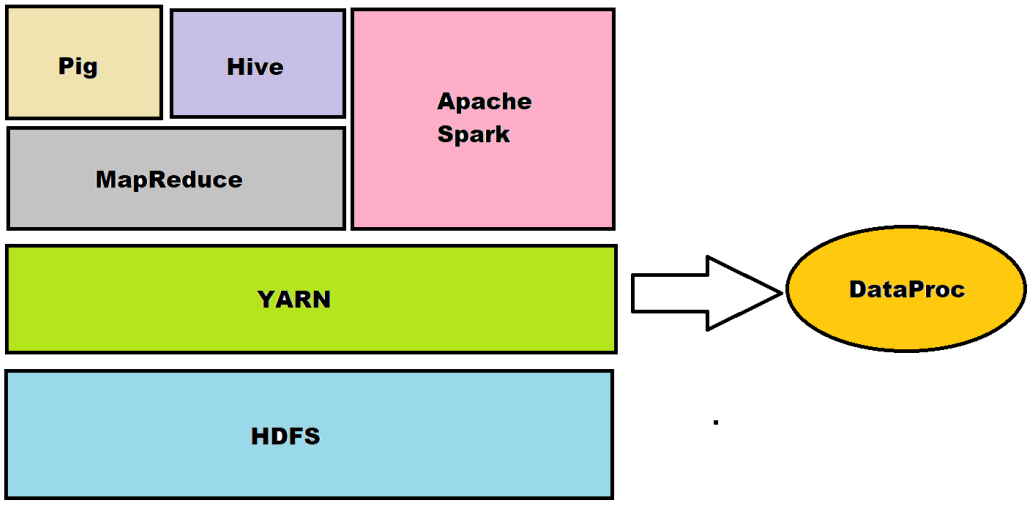
```
  `rich-tine-405803.raw_dataset.post` post
ORDER BY
  caption_length DESC
LIMIT
  5;
```

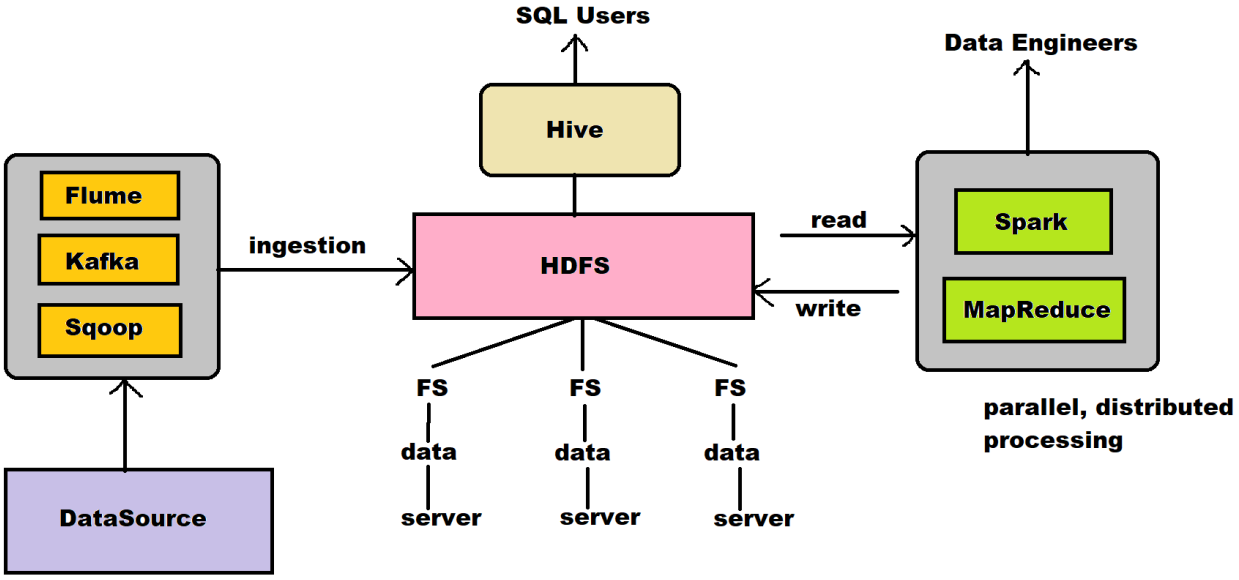
Introduction To Hadoop

02 March 2023 09:00

Hadoop EcoSystem







Cluster : Collection of Nodes : collection of VMs

Cluster : Collection of Nodes : collection of VMs

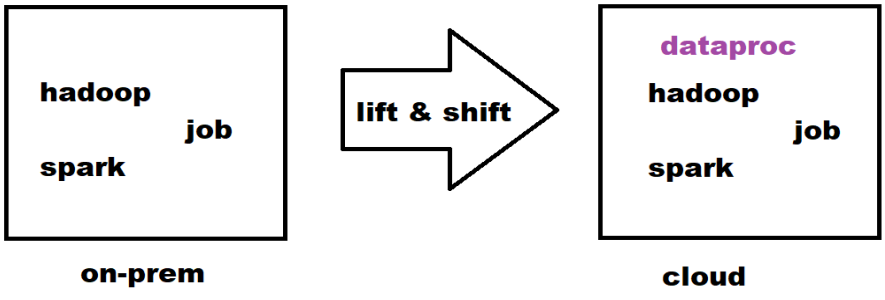
Nodes : Master/Manager Node 1 to 3

Worker/Slave Node 0 to Many

ex : 1 cluster = 1 master node, 2 worker nodes

Cloud DataProc

the purpose of cloud dataproc in life is to run the hadoop and spark jobs

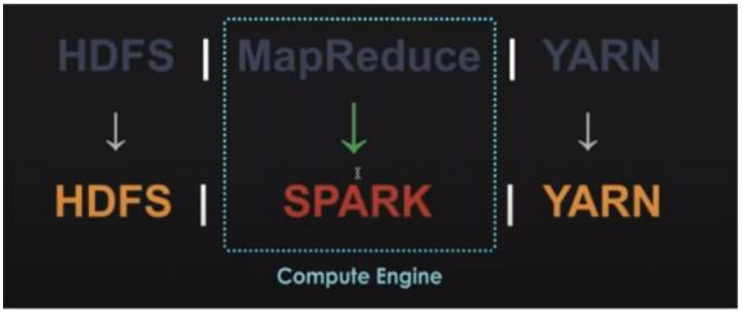


cluster = pool of virtual machines

Hadoop vs Spark

03 March 2023 09:16

MAP REDUCE	SPARK
Computing Framework Engine, open source managed by Apache	Computing Framework Engine, open source managed by Apache
Yes , Map Reduce is Faster than traditional system but it does not leverage the memory of hadoop cluster to the maximum	spark has been proved to execute the batch processing jobs 10 to 100 times faster
Map Reduce is disk Oriented completely. Higher latency. No caching support.	Spark ensures lower latency computations by caching the partials results across its memory of distributed hardware. Stores data in memory
MapReduce is a cheaper option available while comparing it in terms of cost.	As spark requires a lot of RAM to run in-memory. Thus, increases the cluster, and also its cost.
Writing Map reduce pipelines is complex and lengthy as it is purely Java	Writing Spark code is always easy and we can write in 4 languages
Batch Processing	Batch/Iterative/ Real Time /Interactive Processing
Fault Tolerance and Highly Scalable and Cross platform	Fault Tolerance and Highly Scalable and Cross platform
Map Reduce has been tested on 15000 nodes	Spark has been tested on 8000 nodes
it has not inbuilt support to various things like SQL,ML,RT	it has in built support to various things like SQL,ML,RT
It is basic data processing engine.	It is data analytics engine. Hence, it is a choice for Data Scientist.
MapReduce runs very well on commodity hardware.	Spark needs mid to high-level hardware.



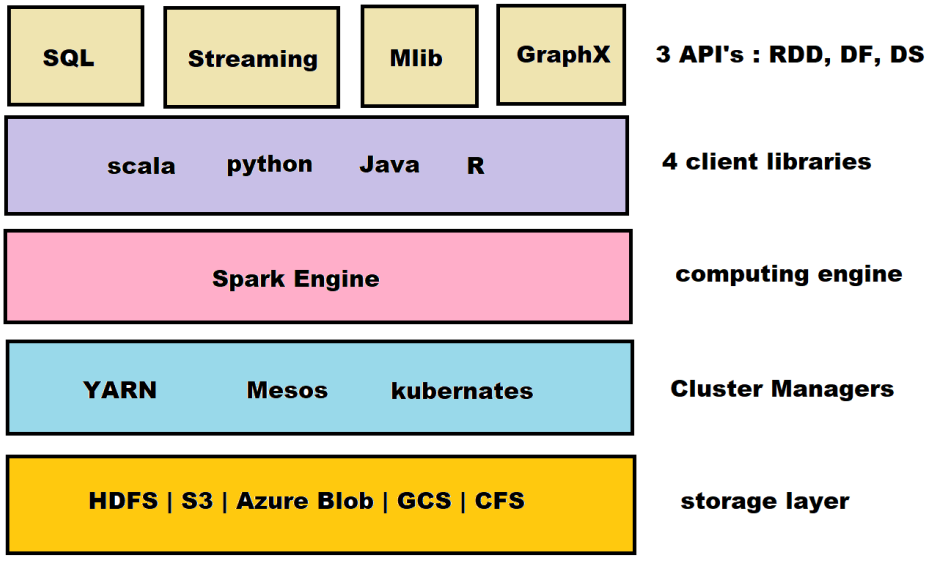
Spark and Pyspark

03 March 2023 08:39

1. What is Spark ?

- In-memory (it reads data from disk, then do all the data processing activity in RAM/Memory, write back to disk)
- distributed (split the activity to multiple file systems)
- parallel data processing framework/engine

a. Fault-Tolerance
b. Lazy Evaluations(all the transformations in spark are lazy, they do not compute the results right away)
c. Open source
d. Supports multiple languages (python, Scala, java and R)
e. Batch + Real-time/streaming
f. 100 times faster than Hadoop Map Reduce



What is Pyspark :

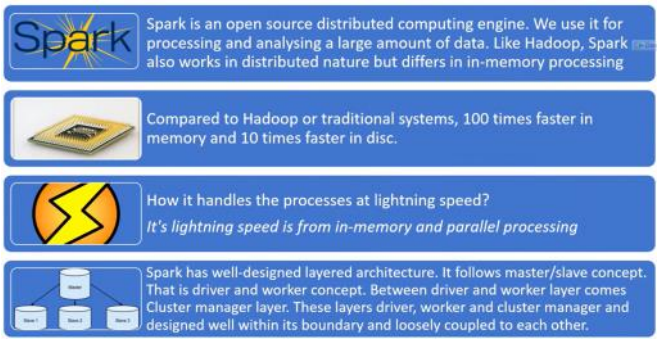
a. Apache Spark community released a tool, Pyspark.
b. Pyspark is the Python API written in python to support Apache Spark.
c. It lets us use the power of Apache Spark in order to tame Big Data.
d. It is because of a library called Py4j that they are able to achieve this.
e. To use Pyspark you will have to install python and Apache spark on your machine. (DataProc)

What is DataProc :

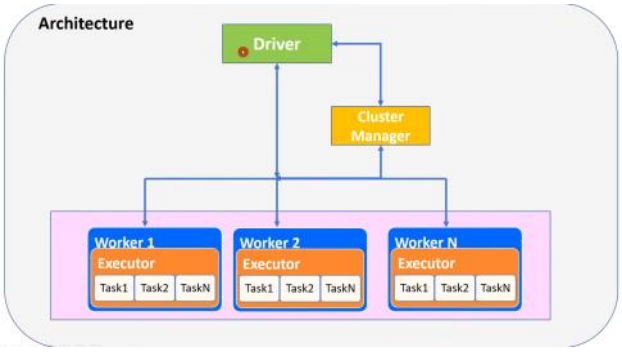
a. Fully managed service for running Hadoop and spark jobs

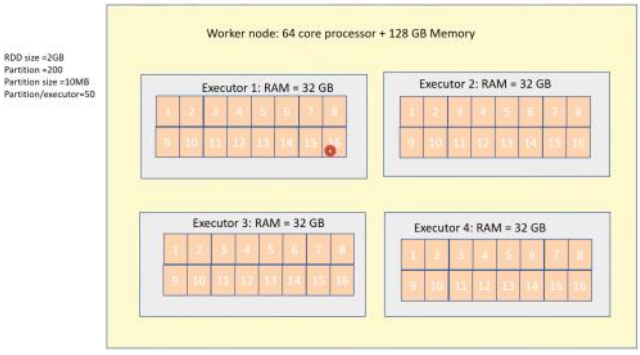
Spark Intro

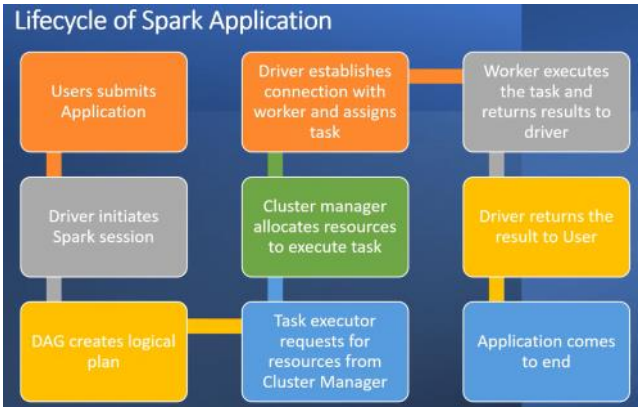
Tuesday, December 5, 2023 10:08 AM

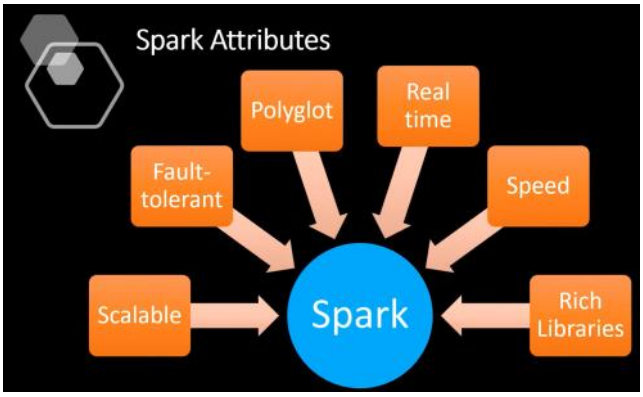



```
Memory
==============

6 Nodes and each node have 15 cores, 64 GB RAM

Cores - no of concurent task

for exampl we can give 5 parallel task for each executor

num cores : 5

num executor :  15/5  => 3 Executor per node

num executor memory : 64 / 3  => 21 gb - 1 or 2 gb for yarn memory = 18 gb ( we can have 5 gb to 18 gb memory for each job )

Driver memory - 8 gb
=======================

node 1

executor 1
task 1 task2 task3 task4 task5

executor 2
task 1 task2 task3 task4 task5

executor 3
task 1 task2 task3 task4 task5
```



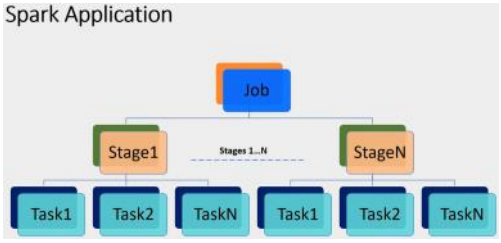






Terminologies

- **Driver and worker Process:** These are nothing but JVM process. Within one worker node, there could be multiple executors. Each executor runs its own JVM process.
- **Application:** It could be single command or combination of multiple notebooks with complex logic. When code is submitted to spark for execution, Application starts.
- **Jobs:** When an application is submitted to Spark, driver process converts the code into job.
- **Stage:** Jobs are divided into stages. If the application code demands shuffling the data across nodes, new stage is created. Number of stages are determined by number of shuffling operarions. Join is example of shuffling operation
- **Tasks:** Stages are further divided into multiple tasks. In a stage, all the tasks would execute same logic. Each task will process 1 partition at a time. So number of partition in the distributed cluster determines the number of tasks in each stage

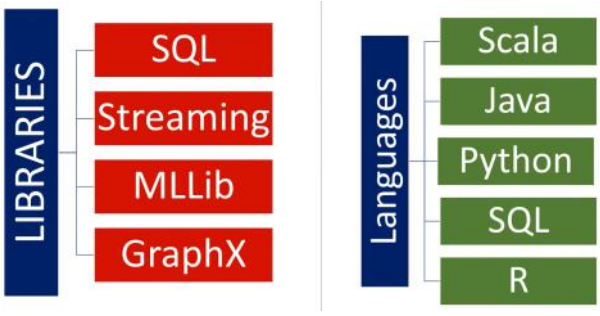


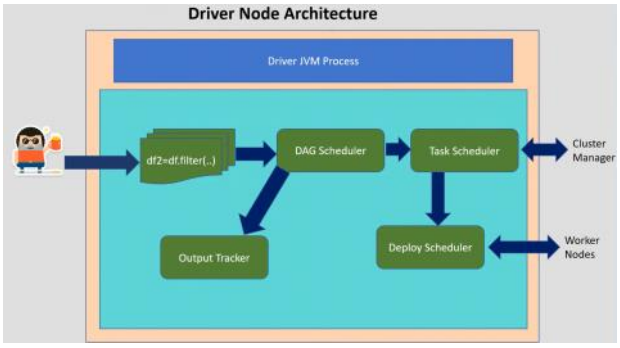
Terminologies

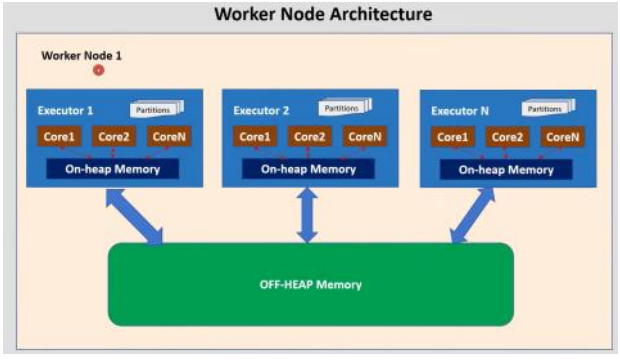
- **Transformation:** transforms the input RDD and creates new RDD. Until action is called, transformations are evaluated lazily
- **DAG:** Directed Acyclic Graph keeps track of all transformation. For each transformation, logical plan is created and lineage graph is maintained by DAG
- **Action:** When data output is needed for developer or for storage purpose, action is called. Action would be executed based on DAG and processes the actual data
- **RDD:** Resilient Distributed Dataset is basic data structure of Spark. When spark reads or creates data, it creates RDD which is distributed across nodes in the form of partition.

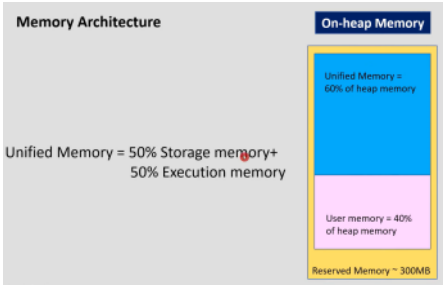
Terminologies

- **Executor:** Each worker node can consists of many executors. It can be configured by spark settings
- **Partition:** RDD/Dataframe is stored in-memory of cluster in the form of partition
- **Core:** Each executor can consists of multiple cores. This is configurable by spark settings. Each core(if single thread) can process one task at a time.
- **On-heap memory:** The executor memory that lies within JVM process managed JVM
- **Off-heap memory:** The executor memory that lies outside JVM process managed by OS


LIBRARIES
SQL
Streaming
MLLib
GraphX
Languages
Scala
Java
Python
SQL
R


Driver Node Architecture
Driver JVM Process
df2=df.filter(..)
DAG Scheduler
Task Scheduler
Cluster Manager
Output Tracker
Deploy Scheduler
Worker Nodes


Worker Node Architecture
Worker Node 1
Executor 1
Partitions
Core1
Core2
CoreN
On-heap Memory
Executor 2
Partitions
Core1
Core2
CoreN
On-heap Memory
Executor N
Partitions
Core1
Core2
CoreN
On-heap Memory
OFF-HEAP Memory


Memory Architecture
On-heap Memory
Unified Memory = 50% Storage memory+
50% Execution memory
Unified Memory = 60% of heap memory
User memory = 40% of heap memory
Reserved Memory ~ 300MB

Cluster creation

13 May 2023 07:42

DataProc , DataBricks, elastic MR => Spark, Hadoop

data engineer : analytics => business

data =====> BigData

challenges :

1. storage : 100 gb of data => single machine ??? => no ==> we need a distributed system => multiple machines
   100 gb => 4 machines => 25 gb each

   who will manage this data distribution ? HDFS
   who will organize this data distribution ? HDFS
   who will actually store this large data ? workers

   HDFS : Hadoop Distributed File System : split data and distribute the data amoung multiple machines in a cluster

   cluster : collection of machines/Nodes (node = machine)
   nameNode/managerNode/masterNode
   dataNode/slaveNode/workerNode

2. computation : processing

Hadoop : Hadoop is an open-source framework used for distributed storage and processing of big data.
1. hdfs for storage
2. mapreduce for computation

HDFS :
1. Hadoop distributed file system designed to store and manage large amounts of data across multiple machines in a Hadoop cluster.
2. HDFS is a key component of the Hadoop ecosystem and is optimized for handling large files and streaming data.
3. It provides high fault tolerance by replicating data across multiple nodes in the cluster, ensuring that data remains available even if individual nodes or components fail.

MapReduce :
1. MapReduce is a programming model and processing framework used for distributed computing and large-scale data processing
2. The MapReduce model simplifies the processing of big data by breaking it into smaller, independent tasks that can be executed in parallel across a cluster of computers. It consists of two main phases: the Map phase and the Reduce phase.
3. In the Map phase, the input data is divided into smaller chunks, and a Map function is applied to each chunk independently. The Map function takes the input data, performs some computation or transformation on it, and produces a set of intermediate key-value pairs.
4. In the Reduce phase, the intermediate key-value pairs are grouped by their keys and processed by a Reduce function. The Reduce function takes a key and a set of values associated with that key and performs further computation or aggregation on the data. The output of the Reduce function is typically the final result of the MapReduce job.
5. disc oriented processing
6. complex, not easy to write the complex pipelines, no library support
7. Batch data processing

Spark :
1. Apache Spark is an open-source, distributed computing system and analytics engine designed for big data processing and analytics. It was developed to address the limitations of MapReduce and provide a faster and more versatile framework for large-scale data processing.
2. Spark offers a unified computing framework that supports various data processing tasks, including batch processing, real-time stream processing, machine learning, and graph processing. It provides high-level APIs in different programming languages like Scala, Java, Python, and R, making it accessible to a wide range of developers and data scientists.
3. im-memory computation
4. python, java, scala, R
5. batch + stream

PySpark :
spark libraries written in python

1. Create DataProc Cluster :

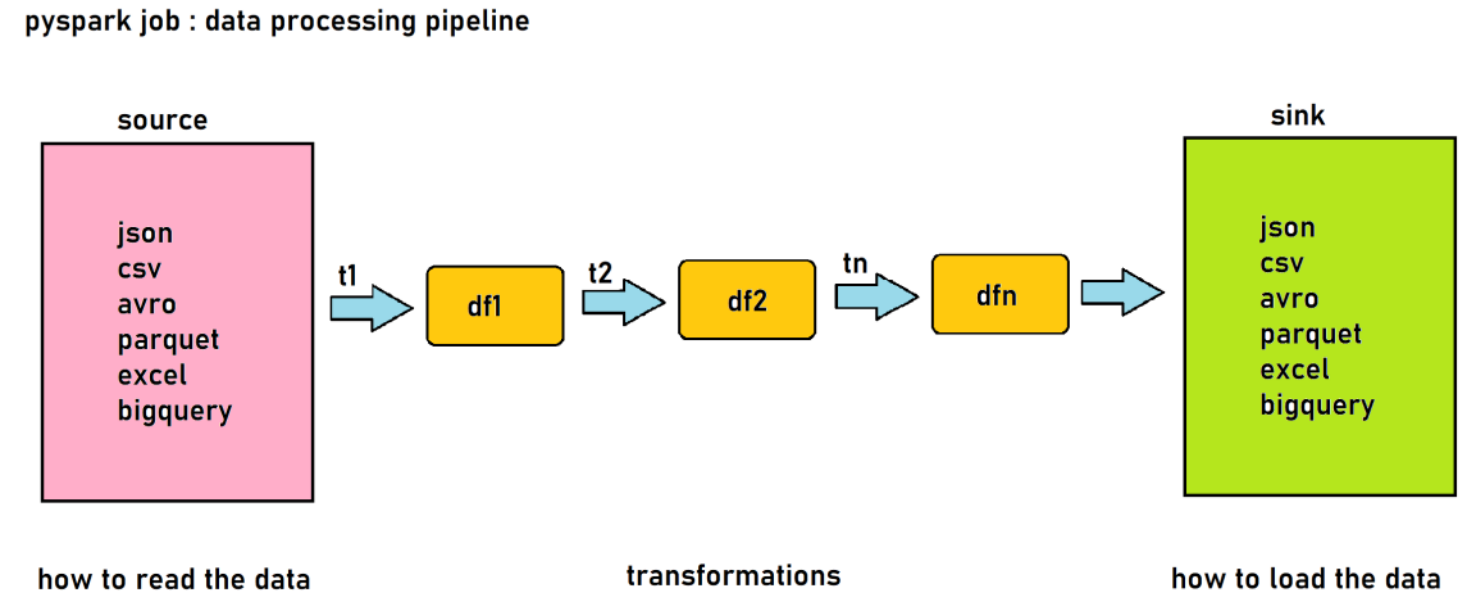
singleNode:

```
CLUSTER_NAME=my-cluster-demo REGION=us-east1
gcloud dataproc clusters create ${CLUSTER_NAME} \
--region ${REGION} \
--single-node \
```

--master-machine-type n1-standard-2 \
--master-boot-disk-size 500 \
--image-version 2.0-debian10 \
--enable-component-gateway \
--optional-components=JUPYTER \
--initialization-actions gs://goog-dataproc-initialization-actions-${REGION}/connectors/connectors.sh \
--metadata bigquery-connector-version=1.2.0 \
--metadata spark-bigquery-connector-version=0.21.0

multiNode:

```
CLUSTER_NAME=my-cluster-demo1
REGION=us-east1
gcloud dataproc clusters create ${CLUSTER_NAME} \
--region ${REGION} \
--num-workers=2 \
--worker-machine-type=n1-standard-2 \
--worker-boot-disk-size=50 \
--master-machine-type=n1-standard-2 \
--master-boot-disk-size=50 \
--image-version=2.0-debian10 \
--enable-component-gateway \
--optional-components=JUPYTER \
--initialization-actions=gs://goog-dataproc-initialization-actions-${REGION}/connectors/connectors.sh \
--metadata bigquery-connector-version=1.2.0 \
--metadata spark-bigquery-connector-version=0.21.0
```



2. Open JupiterLab, select pyspark :

```
# install pyspark
! pip install pyspark

# import sparkSession : entry point to the pyspark functionality
import pyspark
from pyspark.sql import SparkSession

# create SparkSession
spark = SparkSession.builder \
            .master("local") \
            .appName("demo") \
            .getOrCreate()
```

3. Topics in pyspark sql Dataframes :

1. create df :

a. hardcoded values
b. csv
c. json

d. parquet
e. bigquery
f. avro (assignment)
g. txt (assignment)
h. cloud sql : MYSQL

2. write df :

a. txt
b. csv
c. json
d. parquet
e. bigquery
f. avro (assignment)
g. cloud sql

3. transformations :

1. create dataframe :
   a. hard codes values
   b. file formats(csv, json, parquet..)
2. apply/rename columns dynamically
3. lit() : To add new constant column
4. withColumn():
   a. to add a constant column
   b. to add a derived column
5. StructType() and StructField():
   a. to change the column datatype
6. filter() or where() : both are same
   a. to filter the data
   b. apply multiple conditions (&, |) (and, or)
7. distinct()
8. dropDuplicates()
   a. is to remove the duplicates
   b. on each column
9. sort() or orderBy():
   a. to sort the data
   b. multiple columns
10. Union() or unionAll():
    a. to merge the data
    b. conditions : no of columns, datatypes
11. groupBy() :
    a. to group the data
12. aggregation :
    a. min, max, count, avg, sum
13. cast() :
14. when() and otherwise() :
15. split()
17. concat() : for concatenation
18. substring() : to get a part of string
19.Joines
20. Ranking functions

Lab sessions

Thursday, September 21, 2023 10:25 AM

← Code

← Sample Data

Pyspark SQL dataframes Documentation :

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/index.html

```
# create df from mysql table

mysql_props = {
   "driver" : "com.mysql.cj.jdbc.Driver",
   "url" : "jdbc:mysql://34.89.79.221:3306/sample_database",
   "user": "myuser",
   "password" : "mypass"
}

df6 = spark.read.jdbc(url=mysql_props["url"], table="Users", properties=mysql_props)
df6.show()

df6.createOrReplaceTempView("Users")

df7 = spark.sql("select * from Users where id=1")
df7.show()
```

Datetime Functions

Wednesday, December 13, 2023 3:40 PM

In PySpark, we have a powerful toolkit of datetime functions to help us manipulate and analyze temporal data.

Let's dive into some fundamental datetime functions and their usage with examples:

1 to_date():

- This function converts a string representation of a date to a DateType. It's great for filtering or grouping by date.

```
from pyspark.sql.functions import to_date
df = df.withColumn("date_column", to_date("date_string_column", "yyyy-MM-dd"))
```

2 date_add() and date_sub():

- These functions allow you to add or subtract days from a date.

```
from pyspark.sql.functions import date_add, date_sub
df = df.withColumn("future_date", date_add("date_column", 7))
df = df.withColumn("past_date", date_sub("date_column", 3))
```

3 datediff():

- Calculates the difference in days between two dates.

```
from pyspark.sql.functions import datediff
df = df.withColumn("date_difference", datediff("date2", "date1"))
```

4 date_format():

- This function converts a date to a string with a specified format.

```
from pyspark.sql.functions import date_format
df = df.withColumn("formatted_date", date_format("date_column", "dd-MM-yyyy"))
```

5 trunc():

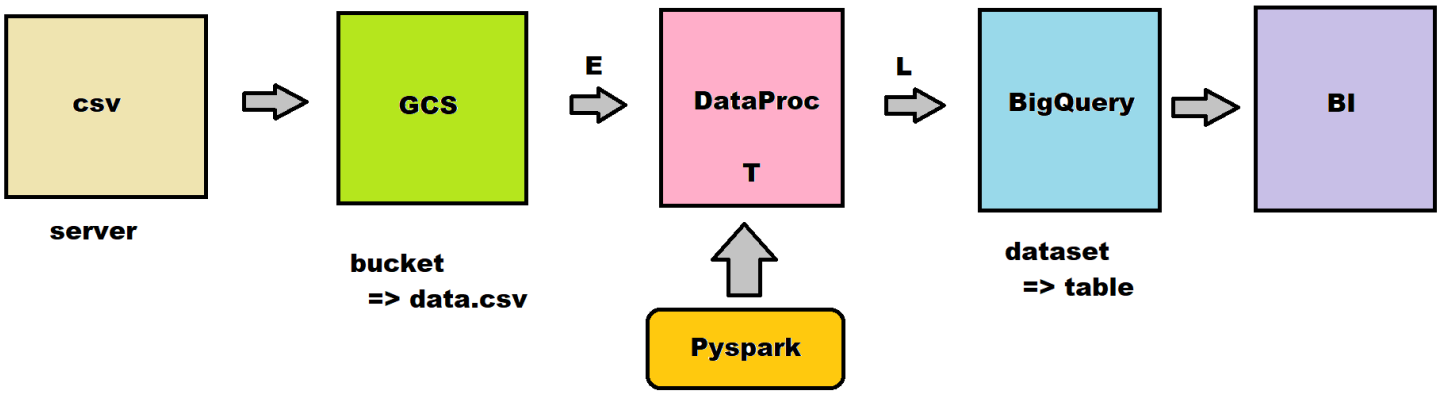
- Truncates a date to a specified level (day, month, year).

```
from pyspark.sql.functions import trunc
df = df.withColumn("year", trunc("date_column", "yyyy"))
df = df.withColumn("month", trunc("date_column", "MM"))
```

These functions empower PySpark users to handle datetime data efficiently, making it easier to perform tasks like data aggregation, filtering, and feature engineering. Harnessing the power of datetime functions is a key step in any data project.

Case study - 1

09 March 2023 09:03



1. source connection(E)
2. Transformations(T)
3. destination connection(L)

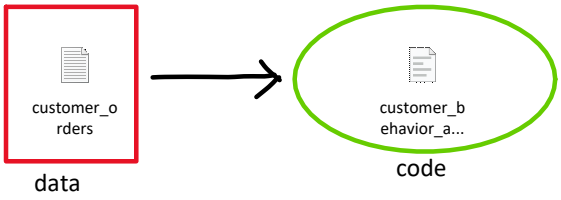
CaseStudy

sample data

ETL script

Casestudy - 2

Friday, December 8, 2023 4:29 PM



You are working on a project for an e-commerce company, and your task is to perform customer behavior analysis.

The dataset = "**customer_orders**"

contains the following columns:

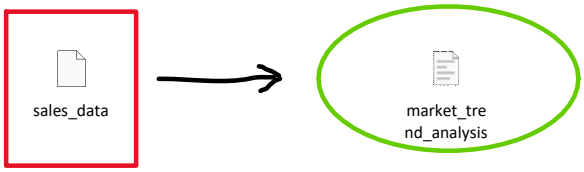
- customer_id: The unique identifier for each customer.
- order_id: The unique identifier for each order.
- order_date: The date when the order was placed.
- total_order_value: The total value of the order.

Write Pyspark to analyze customer behavior by calculating the following metrics for each customer:

1. Customer lifetime value (CLV): the total revenue generated by a customer over their entire history with the company.
2. Purchase frequency: the average number of days between orders.
3. Average order value: the mean of the total order values for all their orders.
4. Customer churn rate: customers who have not made a purchase in the last 90 days

Case study - 3

Wednesday, December 13, 2023 1:58 PM



You are working on a PySpark project for a retail analytics company, and your task is to perform a market trend analysis

The dataset = "**sales_data**"

contains the following columns:

- product_id: The unique identifier for each product.
- product_name: The name of the product.
- quantity_sold: The quantity of each product sold.
- order_date: The date when the product was sold.

Write a PySpark code to analyze market trends by calculating the monthly sales growth for each product category.

The output should include the month, product category, and the percentage change in sales compared to the previous month.

Assignment-1

Thursday, September 28, 2023 8:21 AM

1. Apply header dynamically :

customer_records

headerlist

applyingHeader_dyn...

Assignement-2

Wednesday, December 13, 2023 3:48 PM

#Suppose you are having some purchase data and product data from that solve the following 3 questions.

- ◇ Find customers who have bought only product A.
- ◇ Find customers who upgraded from product B to product E (they might have bought other products as well).
- ◇ Find customers who have bought all the models in the new Product Data.

```
purchase_data = spark.createDataFrame([
 (1, "A"),
 (1, "B"),
 (2, "A"),
 (2, "B"),
 (3, "A"),
 (3, "B"),
 (1, "C"),
 (1, "D"),
 (1, "E"),
 (3, "E"),
 (4, "A")
], ["customer", "product_model"])

product_data = spark.createDataFrame([
 ("A",),
 ("B",),
 ("C",),
 ("D",),
 ("E",)
], ["product_model"])
```

1. **Find customers who have bought only product A**

```
only_product_A_customers = spark.sql("""
 SELECT customer
 FROM purchase_data
 GROUP BY customer
 HAVING COUNT(DISTINCT product_model) = 1
""")
```

2. **Find customers who upgraded from product B to product E**

```
upgrade_customers = spark.sql("""
 SELECT DISTINCT p1.customer
 FROM purchase_data p1
 JOIN purchase_data p2 ON p1.customer = p2.customer
 WHERE p1.product_model = 'E' AND p2.product_model = 'B'
""")
```

3. **Find customers who have bought all models in the new Product Data**

```
all_models = spark.sql("""
 SELECT customer
 FROM purchase_data
 GROUP BY customer
 HAVING COUNT(DISTINCT product_model) = (SELECT COUNT(*) FROM product_data)
```

"""")

End to End Pipeline

Tuesday, January 2, 2024 11:02 AM

It's a PySpark job that reads data from a BigQuery table, filters it based on a specified tag, and then writes the filtered d ata to Google Cloud Storage (GCS) in a compressed CSV file.

Here's an explanation:

a. **Importing Libraries:**
   1) sys, re, time, datetime: Standard Python libraries for system-specific parameters and functions, regular expressions, time-related operations, and date/time.

b. **Py4JJavaError:**
   1) An exception class to catch specific PySpark errors.

c. **storage:**
   1) A module for interacting with Google Cloud Storage.

d. **Importing Spark Session:**
   1) Imports the SparkSession class from PySpark.

e. **Creating Spark Session:**
   1) Initializes a SparkSession named "stackoverflow" for Spark SQL operations.

f. **Setting Variables:**
   1) table: Specifies the name of the BigQuery table (bigquery-public-data.stackoverflow.posts_questions).
   2) bucket_name: Specifies the name of the Google Cloud Storage bucket.
   3) tag: Specifies the tag to be used for filtering the data.
   4) path: Specifies a temporary location on GCS.
   5) tmp_output_path: Specifies the output path on GCS.

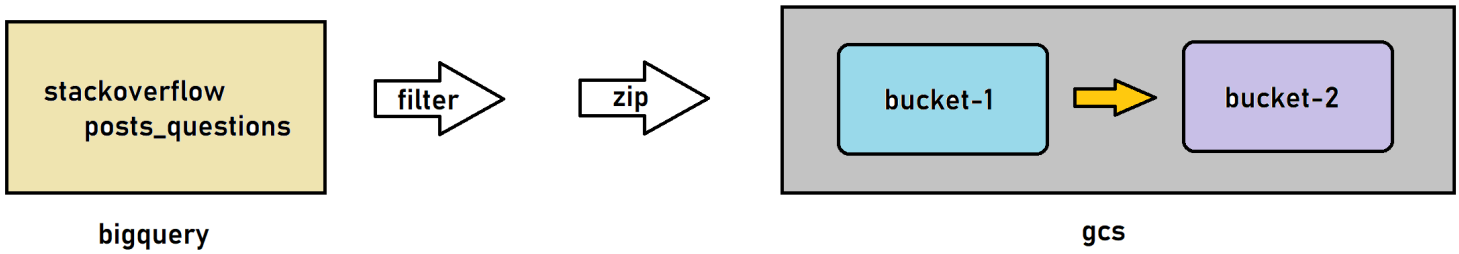
g. **Reading Data from BigQuery:**
   1) Uses the spark.read.format('bigquery').option('table', table).load() method to read data from the specified BigQuery table.
   2) If the specified table does not exist, catches a Py4JJavaError and prints a message before exiting the script.

h. **Filtering Data:**
   1) Uses DataFrame operations to filter rows where the "tags" column matches the specified tag.

i. **Writing Filtered Data to GCS:**
   1) Specifies the output path on GCS using the provided bucket name and a temporary path.
   2) Coalesces the DataFrame to a single partition, gzips the output file, and writes the data in CSV format to the specified GCS path.

j. **Renaming and Copying Files on GCS:**
   1) Defines a regular expression (regex) to match the output file(s) on GCS.
   2) Renames the output file to a new path (stackoverflow_question_posts.csv.gz) for simplicity.
   3) Uses the Google Cloud Storage client to copy the identified file to the new path.
   4) Deletes the temporary directory on GCS after copying the file.

Note: The script assumes that the specified BigQuery table exists and that the GCS bucket is properly configured with the nec essary permissions. If the table doesn't exist, it prints a message and exits gracefully.


stackoverflow
posts_questions
filter
zip
bucket-1
bucket-2
bigquery
gcs

Introduction

29 May 2023 21:00

Apache Beam :

- Unified programming model for defining both batch and streaming data-parallel processing that can build portable Big data pipelines.
- Unified API to process both batch and streaming data.
- Batch + Stream -> **Beam**
- Portable, beam pipeline once created in any language can be able to run on any execution frameworks
- Beam is a programming model whereas flink, Dataflow and spark are execution engines.

Languages:

- Java
- Python
- Go

Runner:

- Spark
- Flink
- Apex
- Google Dataflow
- Samza

Cloud Dataflow:

- Unified stream and batch data processing that's serverless, fast, and cost-effective.
- Fully managed data processing service
- Automated provisioning and management of processing resources
- Horizontal autoscaling of worker resources to maximize resource utilization
- OSS community-driven innovation with Apache Beam SDK
- Reliable and consistent exactly-once processing

spark programming model => python, java, go, r => code => DataProc => managed => bigdata processing
Apache beam programming model => python, java, go, r => code => dataflow => serverless => bigdata processing

DataProc => spark pipeline => (spark + python) => source => dataframe1 => transformation1 => dataframe2 => transformation2 => dataframe3 => destination
Dataflow => beam pipeline => (beam + python) => source => pcollection1 => ptransformation1 => pcollection2 => ptransformation2 => pcollection3 => destination

Driver Program :

1. Create driver program using the classes from Beam SDK
2. Driver program defines our pipeline

Driver Program consists of :
Inputs = From Batch and Stream data sources
Transforms = ParDo, Map, FlatMap, GroupByKey, Filter ..etc
Outputs = To Sinks like BigQuery,GCS ...etc
Pipeline options = Set Runners, batch/stream and other options

- Create a Pipeline object and set the pipeline execution options, including the Pipeline Runner
- Create an initial PCollection using the IOs to read data from an external storage system /PCollection from in -memory data
- Apply PTransforms to each PCollection. Transforms can change, filter, group, analyze
- Use IOs to write the final, transformed PCollection(s) to an external source
- Run the pipeline using the designated Pipeline Runner.

Beam Pipeline :

pTransform1 pTransform2 pTransform3
DataSource ==========> pCollection1 ============> pCollection2 ==========> DataSink

Pcollection: It represents a distributed, multi-element data set that our beam pipeline operates on.

1. Immutability: Pcollections are immutable in nature. Applying a transformations on a pcollection results in creation of new pcollection.
2. Element type: The elements in pcollection may be of any type, but all must be of same type.
3. Operation type: Pcollection does not support grained operations. We cannot apply transformations on specific elements in pcollection.
4. Timestamps: Each element in pcollection has an associated timestamp with it.
5. Unbounded Pcollections: An unbounded Pcollection represents a data set of unlimited size. Source assigns the timestamps.
6. Bounded Pcollections: A bounded Pcollection represents a data set of a known fixed size. Every element is set to same time stamp.
7. A Pcollection is owned by the specific Pipeline object for which it is created; multiple pipelines cannot share a Pcollect ion.

Ptransform: Ptransform represent a data processing operation, or a step in our pipeline.

1. Map: Applies a simple 1-to-1 mapping function over each element in the collection.
2. FlatMap: Applies a simple 1-to-many mapping function over each element in the collection.
3. Filter: Given a predicate, filter out all elements that don't satisfy that predicate.
4. Lambda: is an anonymous function. A lambda function can take any number of arguments, but can only have one expression.
5. ParDo
6. GroupByKey
7.CombibePerKey

workbench or Apache Beam Notebook :

1. Apache Beam notebooks are made available through Vertex AI Workbench user-managed notebooks (Service that hosts VM notebooks)
2. Apache Beam interactive runner with Jupiter Lab notebooks lets you iteratively develop pipelines Inspect your pipeline gra ph, see and validate individual Pcollections using interactive beam
3. Apache Beam notebooks already come with Apache Beam and Google Cloud connector dependencies installed

Apache beam :

1. programming models
2. data processing
3. distributed
4. python, java, Go
5. it does not have its own execution engine,
==> execution engine agnostic
spark engine, flink, samza, Dataflow
6. portability
7. both Batch + strEAM ==> BEAM ==> Unified

spark :

1. programming models
2. data processing
3. distributed
4. scala, python, java, R
5. spark is the execution engine
   (cluster of machines)
6. less portable
7. batch + stream

Spark vs Beam

Tuesday, October 3, 2023 10:24 AM

let's break down the differences between Apache Spark and Apache Beam :

1. Programming Model:

Apache Spark:
- Provides a programming model for distributed data processing.
- Allows you to write data processing applications using high-level APIs in languages like Scala, Java, Python, and R.

Apache Beam:
- Also provides a programming model for distributed data processing.
- Offers a unified API for defining data processing pipelines that can run on various execution engines.

2. Execution Engine:

Apache Spark:
- Includes its own execution engine.
- Can run applications written using its programming model on a cluster of machines.
- Offers a standalone cluster manager and can integrate with other cluster managers like YARN and Mesos.

Apache Beam:
- Designed to be execution engine agnostic.
- Does not have its own execution engine.
- Can run on various execution engines, including Apache Spark, Apache Flink, Google Dataflow, and others.
- Offers portability, allowing you to write your data processing logic once and run it on different engines.

3. Data Processing Paradigms:

Apache Spark:
- Originally designed for batch processing but has added streaming capabilities (Structured Streaming).
- Provides batch processing and micro-batch processing for stream processing.

Apache Beam:
- Designed for both batch and stream data processing from the beginning.
- Offers a unified programming model for batch and stream processing, making it versatile for hybrid use cases.

4. Supported Languages:

Apache Spark:
- Supports multiple programming languages, including Scala, Java, Python, and R.
- Has a rich ecosystem of libraries and extensions for various data processing tasks.

Apache Beam:
- Also supports multiple programming languages, including Java, Python, and others, through its SDKs.
- Offers language-agnostic pipeline portability through its model.

5. Use Cases:

Apache Spark:
- Well-suited for a wide range of data processing tasks, including ETL, data warehousing, machine learning, and graph processing.
- Particularly strong in batch processing.

Apache Beam:
- Suitable for data processing tasks where portability across different execution engines is required.
- Provides a unified approach to handling batch and stream processing.

6. Deployment:

Apache Spark:
- Often deployed on clusters with Spark's own cluster manager or integrated with other cluster managers.

Apache Beam:
- Can be deployed on various cloud-based services, including Google Dataflow, and on-premises clusters.
- Offers a serverless execution option when used with Google Dataflow.

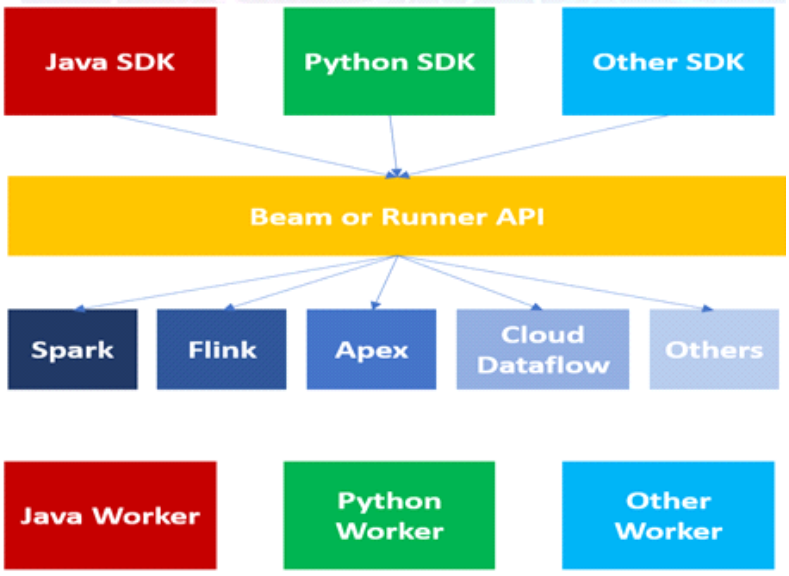
In summary, both Apache Spark and Apache Beam are programming models for distributed data processing, but Apache Spark includes its own execution engine, while Apache Beam is execution engine agnostic, focusing on portability across different engines. The choice between the two depends on your specific use case and deployment requirements.

Apache beam :

1. programming models
2. data processing
3. distributed
4. python, java, Go
5. it does not have its own execution engine,
   ==> execution engine agnostic
   spark engine, flink, samza, Dataflow
6. portability
7. both Batch + strEAM ==> BEAM ==> Unified

spark :

1. programming models
2. data processing
3. distributed
4. scala, python, java, R
5. spark is the execution engine
   (cluster of machines)
6. less portable
7. batch + stream



Dataproc vs Dataflow

Tuesday, October 3, 2023 10:17 AM

let's compare Google Cloud Dataproc and Google Cloud Dataflow :

1. Service Purpose:

Google Cloud Dataproc:
- Dataproc is a fully managed cloud service for running Apache Hadoop, Apache Spark, Apache Hive, Apache Pig, and other big data frameworks on Google Cloud.
- It provides clusters for batch and stream processing, data exploration, and machine learning tasks.

Google Cloud Dataflow:
- Dataflow is a fully managed stream and batch data processing service that enables you to build data pipelines for ETL (Extract, Transform, Load), real-time analytics, and more.
- It provides a unified programming model for both batch and stream processing.

2. Data Processing Paradigms:

Google Cloud Dataproc:
- Primarily designed for batch and interactive processing.
- Can be used for batch ETL jobs, data exploration, and offline data processing.

Google Cloud Dataflow:
- Designed for both batch and stream data processing from the beginning.
- Offers a unified programming model for hybrid batch and stream processing.

3. Ease of Use:

Google Cloud Dataproc:
- Suitable for users already familiar with Hadoop and Spark.
- Provides more fine-grained control over cluster configuration.

Google Cloud Dataflow:
- Offers a simpler and more abstracted way to create data pipelines.
- Suitable for users who want a serverless, managed service for data processing.

4. Language Support:

Google Cloud Dataproc:
- Supports various programming languages depending on the framework used (e.g., Scala, Java, Python for Spark).

Google Cloud Dataflow:
- Offers multiple SDKs, including Java and Python, for building data pipelines.

5. Serverless vs. Cluster-Based:

Google Cloud Dataproc:
- Requires you to manage and provision clusters for data processing tasks.
- Offers more control over cluster configuration.

Google Cloud Dataflow:
- Provides a serverless experience where you don't need to manage clusters; Google handles the underlying infrastructure.
- Scales automatically based on the workload.

6. Use Cases:

Google Cloud Dataproc:
- Well-suited for big data processing tasks that require specific frameworks like Spark or Hadoop.
- Suitable for running long-lived clusters for repeated batch processing.

Google Cloud Dataflow:
- Suitable for building ETL pipelines, real-time analytics, and event-driven data processing.
- Designed for dynamic, serverless data processing needs.

7. Integration:

Google Cloud Dataproc:
- Integrates well with other Google Cloud services and third-party tools compatible with Hadoop and Spark.

Google Cloud Dataflow:
- Offers integration with various data sources and sinks, including Pub/Sub, BigQuery, Cloud Storage, and more.

8. Cost Model:

Google Cloud Dataproc:
- Costs are primarily based on the number and size of the clusters you provision.

Google Cloud Dataflow:
- Costs are based on the actual processing resources used during execution and the duration of the job.

In summary, Google Cloud Dataproc is well-suited for users who want more control over cluster configuration and need to run specific big data frameworks. Google Cloud Dataflow, on the other hand, is designed for users who prefer a serverless, and unified data processing service for both batch and stream processing tasks. The choice between the two depends on your specific data processing requirements and preferences.

1. Runner to execute the Apache beam pipelines
2. GCP serverless platform
3. auto-provisioning of processing resources
4. batch + stream processing
5. python, java, Go
6. ETL jobs
7. cost ==> resources utilized during executions

1. To run spark and Hadoop jobs
2. GCP managed service
3. cluster ==> 3 machines => no-auto-provision
4. batch + stream processing
5. python, java, R, scala
6. ETL jobs
7. cost ==> size of the cluster

Lab sessions

Tuesday, October 3, 2023 10:23 AM


Introductio
n


Extract (2)


Transforma
tions


Load (2)


job-1


sales


products

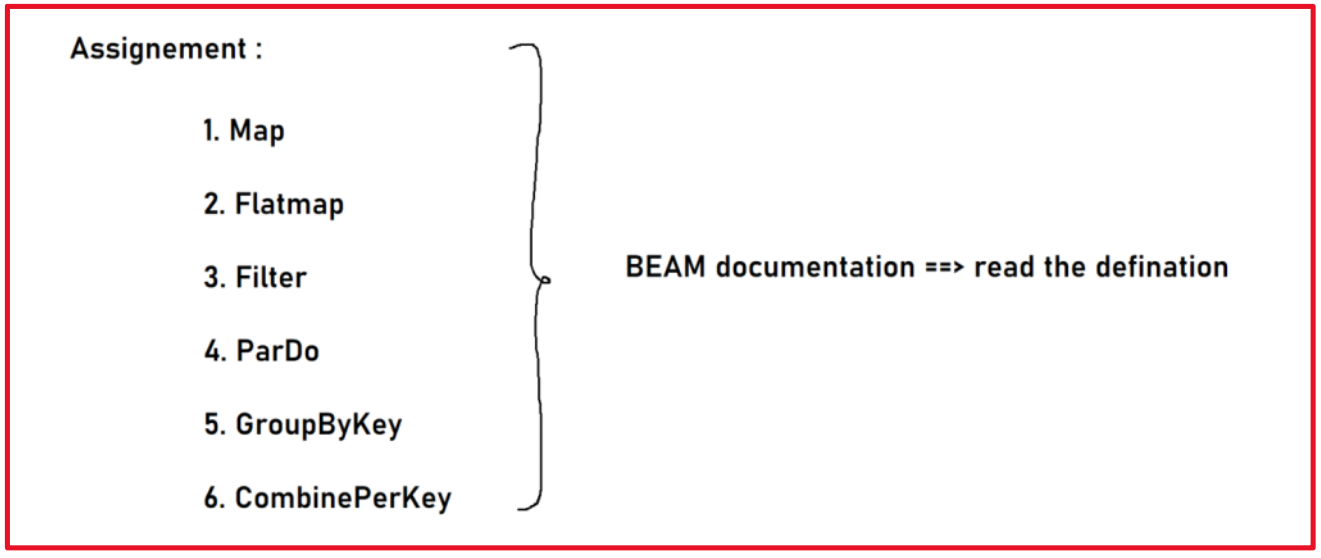

employee


usedata


sampledata


employee_
data


userdata2



Javafile

Json file

sample data

Cloud Storage to BigQuery : batch pipeline

1. Create a storage bucket and BigQuery Dataset
2. Place input file , java script file and JSON schema
3. Run Dataflow pipeline job to read cloud storage and write to BigQuery

Pub/Sub to BigQuery : streaming pipeline

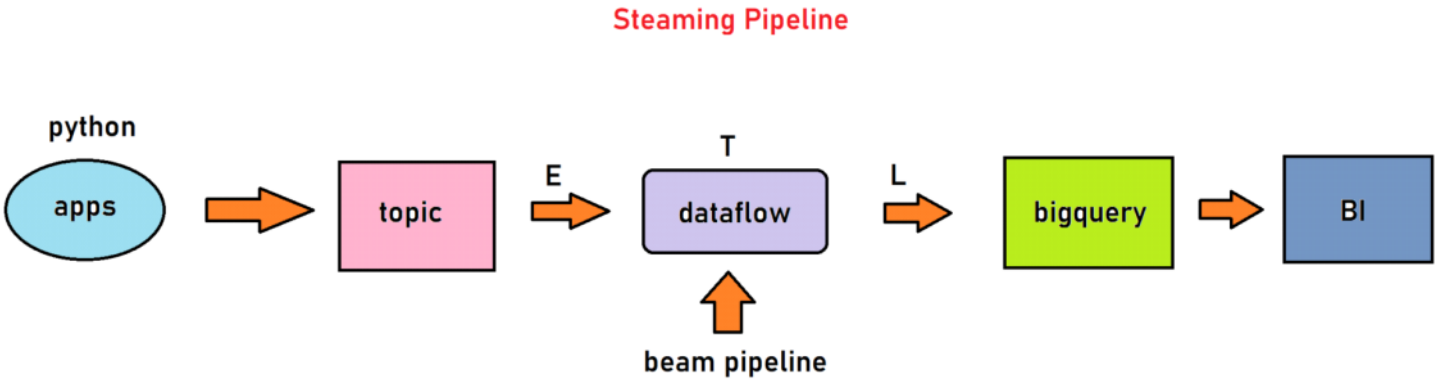
1. Generating messages in Pub/Sub
2. Dataflow pipeline read and process this message
3. Output will be reached to Big Query table


```
CREATE TABLE `test-project-1278954.sample_dataset.employee`(
  id int64,
  name string,
  city string
) ;
```

```
{"id":1, "name":"ravi", "city":"hyd"}
```

End to End Stream Pipeline

Monday, January 8, 2024 1:48 PM



PubSub

16 March 2023 08:02

- ✓ Pub/Sub is an asynchronous and scalable messaging service that decouples services producing messages from services processing those messages.
- ✓ Pub/Sub allows services to communicate asynchronously, with latencies on the order of 100 milliseconds.
- ✓ Pub/Sub is used for streaming analytics and data integration pipelines to load and distribute data.
- ✓ It's equally effective as a messaging-oriented middleware for service integration or as a queue to parallelize tasks.
- ✓ Pub/Sub lets you to create systems of event producers and consumers, called **publishers** and **subscribers**. Publishers communicate with subscribers asynchronously by broadcasting events, rather than by synchronous remote procedure calls (RPCs).

Let's break it down in simple terms:

Publishing:

- Imagine you have a message or piece of information that you want to share.
- In PubSub, you "publish" this message to a central hub or a topic.

Subscribing:

- On the other end, there are entities or components that are interested in receiving certain types of messages.
- These entities "subscribe" to specific topics or categories.

Communication:

- When you publish a message to a topic, all the entities that have subscribed to that topic will receive the message.
- It's a way for different parts of a system to communicate without knowing each other directly.

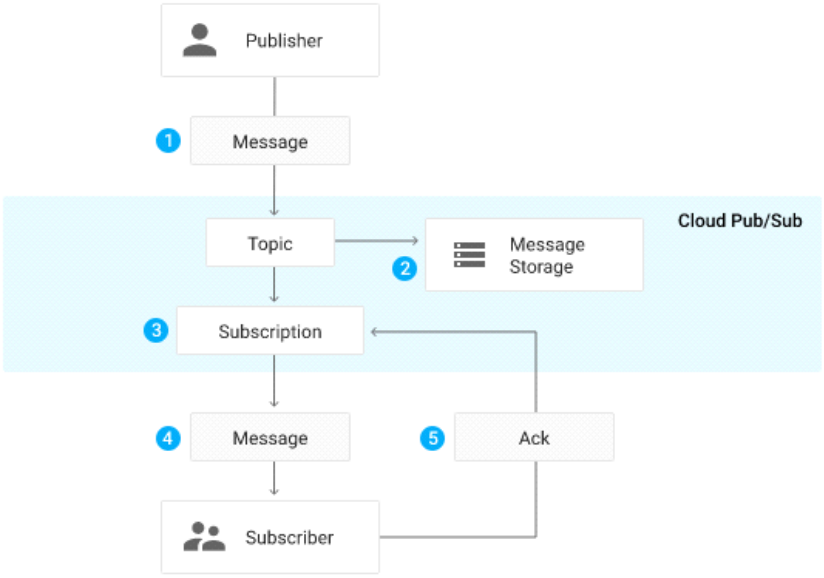
Decoupling:

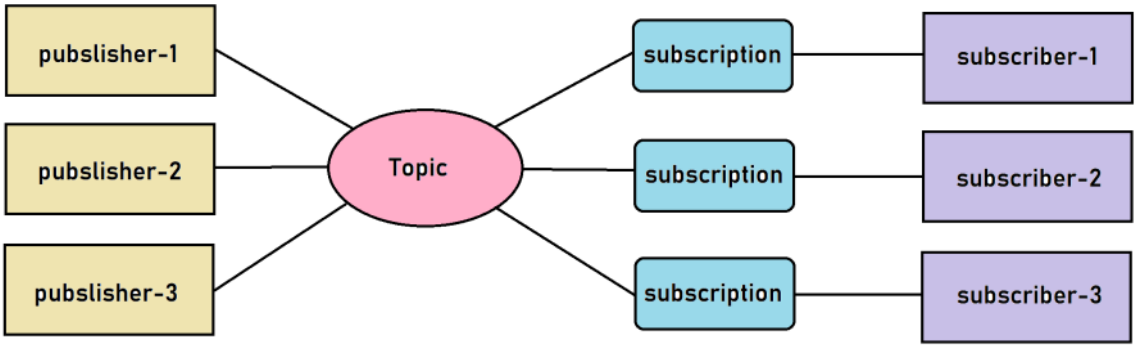
- One key advantage of PubSub is decoupling. Publishers and subscribers don't need to know about each other. They only need to know about the topics.
- This makes the system more flexible. You can add or remove components without affecting others as long as they adhere to the same message format and topics.

Example:

- Think of it like a newspaper. The newspaper (publisher) prints news articles and distributes them.
- Subscribers can choose which sections they want (sports, politics, etc.).
- The publisher doesn't know who reads what, and readers don't know where the articles come from; they just get what they're interested in.

In the context of software development, PubSub is often used in distributed systems, microservices architectures, and event-driven systems to enable communication between different parts of the application without direct dependencies.


Publisher
1
Message
Cloud Pub/Sub
Topic
2
Message Storage
3
Subscription
4
Message
5
Ack
Subscriber


pubslisher-1
pubslisher-2
pubslisher-3
Topic
subscription
subscription
subscription
subscriber-1
subscriber-2
subscriber-3

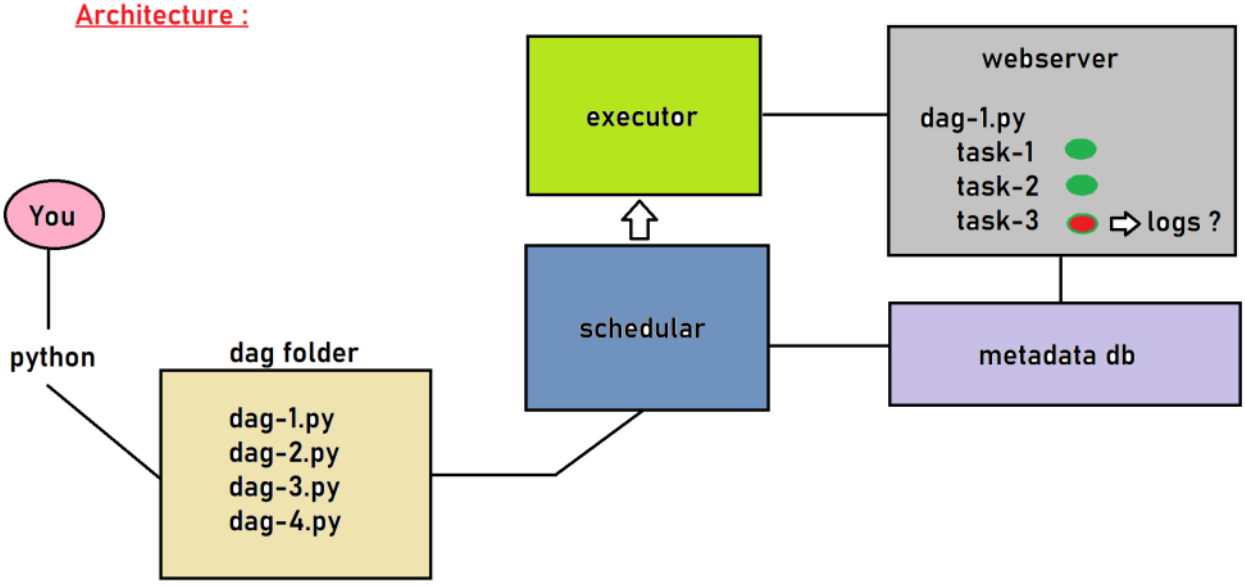
Composer or Airflow

17 March 2023 08:01

- ✓ fully managed workflow orchestration service
- ✓ Managed Airflow service
- ✓ It is based on Apache Airflow, an open-source platform designed for
  - Creating,
  - Scheduling
  - Managing
  - Monitoring complex workflows.
- ✓ Workflow orchestration tool
- ✓ Code based tool : Python
- ✓ Batch pipelines only (no streaming pipelines)
- ✓ Create DAG (Directed Acyclic Graphs)
- ✓ Common use cases for Cloud Composer include
  - data processing pipelines, scheduling
  - ETL (Extract, Transform, Load) workflows,
  - Automation of routine tasks.

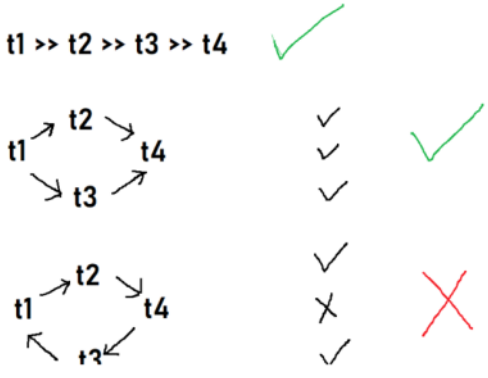
Pipeline = workflow = DAG = Series of tasks

Architecture :



What is DAG :

Directed ==>
Acyclic ==>
Graphs ==>



Create Dag :

1. import all the modules, libraries, ..

2. define the DAG :
   a. DAG ID, b. schedule, interval, startdate (cron job format)
   c. retry , d. owner

3. define tasks :
   a. task-id
   b. operator (GcsToBigqueryOperator) , parameters

4. define Dependency :
   a. bitshift operator (>>)
   b. t1 >> t2 >> t3 >> t4
   c. t1 >> (t2, t3) >> t4

cron job format

everyday at 5pm ==> 0 17 * * *

every Monday at 3:45 PM ==> 45 15 * * 1

1st of every month at midnight ==> 0 0 1 * *

every weekday at 8:00 AM ==> 0 8 * * 1-5

every 15 minutes ==> */15 * * * *

alternate days 8:00 am ==> 0 8 */2 * *

sun-0
mon-1
tue-2
wed-3
thurs-4
fri-5
sat-6

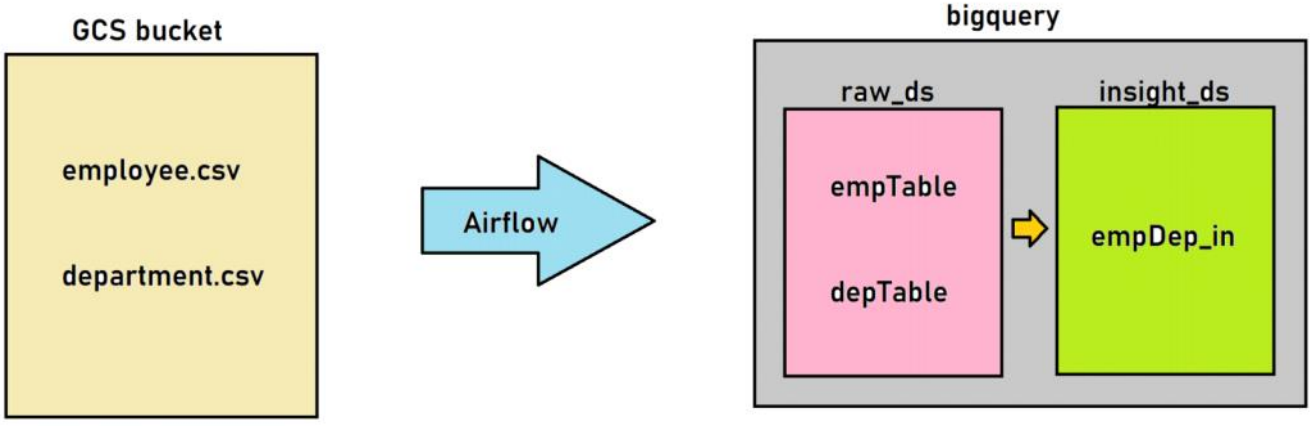
LEVEL1_DAG

Monday, January 8, 2024 3:00 PM

level_1_dag

employee

department
s

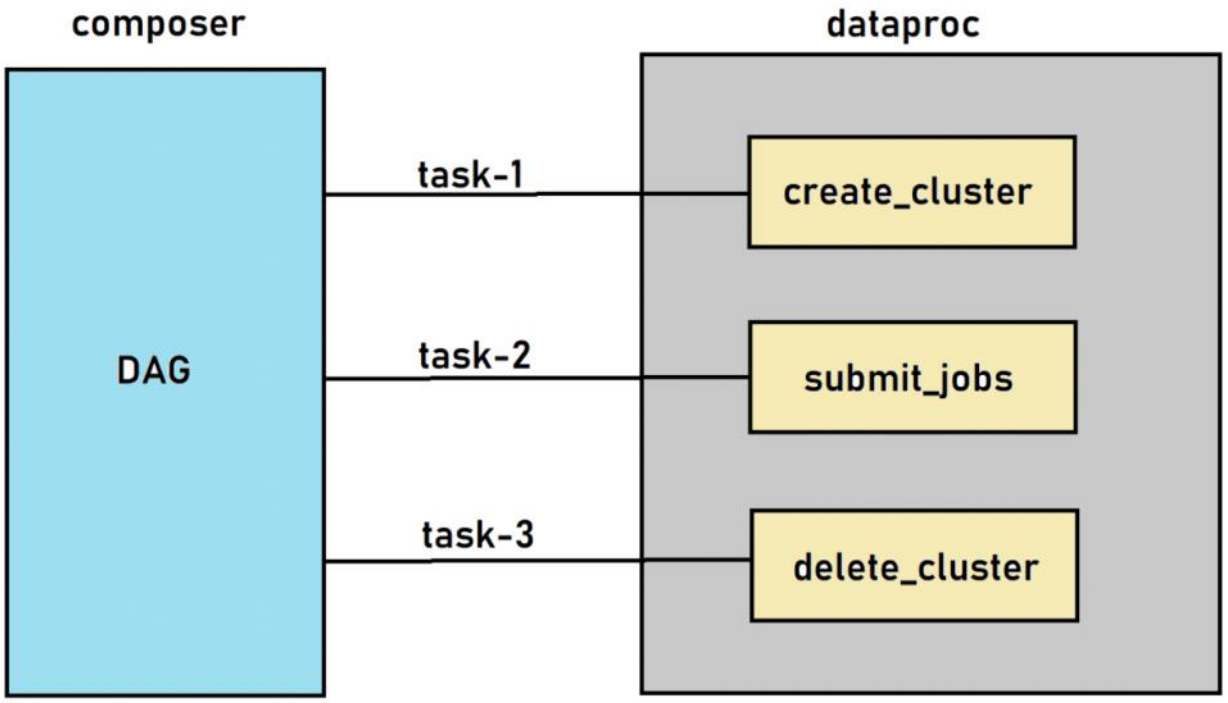


LEVEL2_DAG

Monday, January 8, 2024 3:00 PM


level_2_da
g


main



LEVEL3_DAG

Monday, January 8, 2024 3:01 PM

level_3_dag

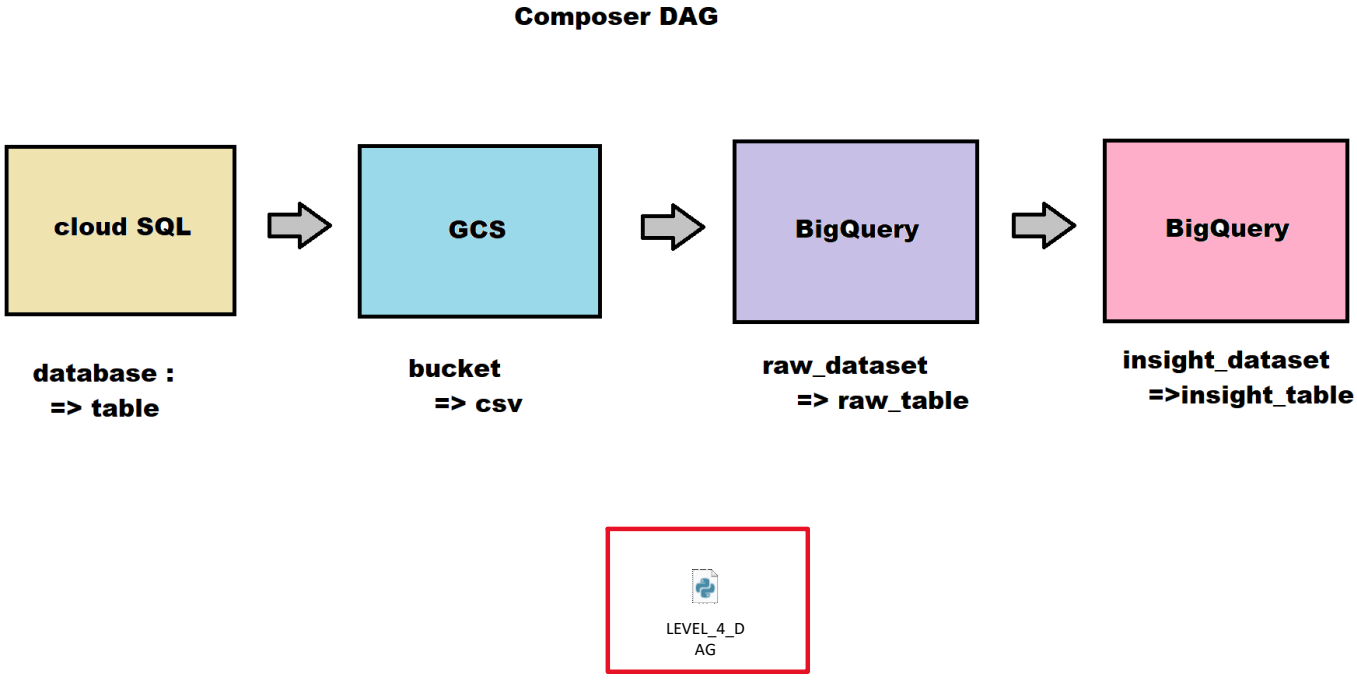
parent dag

child_dag_1

child_dag_2

Assignment : LEVEL4_DAG

20 March 2023 08:58



Introduction

22 March 2023 08:31

- ✓ Cloud Data Fusion is a fully-managed, cloud native, enterprise data integration service for quickly building and managing data pipelines.
- ✓ It provides a graphical interface to increase time efficiency and reduce complexity, and allows business users, developers, and data scientists to easily and reliably build scalable data integration solutions to cleanse, prepare, blend, transfer and transform data without having to wrestle with infrastructure.

Cloud DataFusion

1. brand new, fully managed
2. cloud native
3. data integration service
4. quicly and efficiently build and manage ELT/ETL data pipelines
5. intents to shift the focus from code so that Data engineers can focus on insights
6. allows you to build scalable data integration pipelines to clean, prepare, transform data
   a. without managing the infrastructure
   b. without writing any code
7. datafusion is build on top of open source CDAP project

=> build pipeline without writing any code
=> do transofrmation without writing any code

data fusion is a drag and drop UI tool to create data solutions

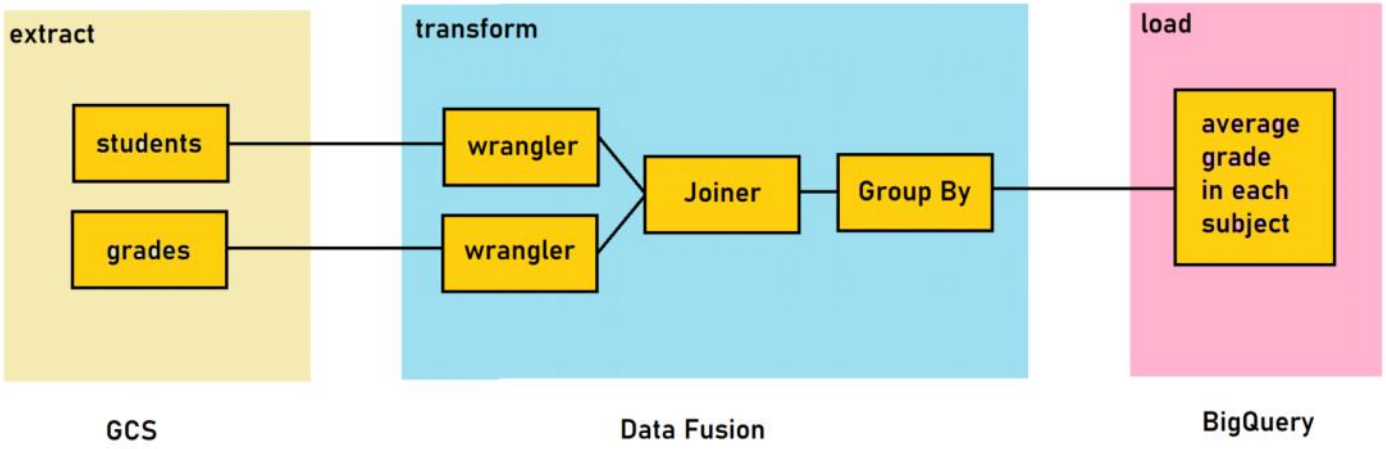
Data Processing systems :
1. dataproc
2. dataflow
3. datafusion

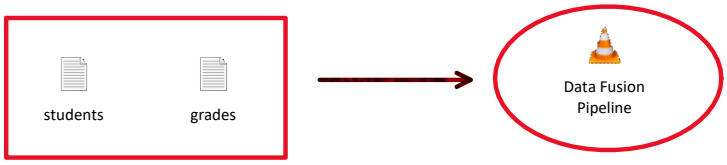
dataproc and dataflow
1. pipeline : code
2. transform : code

datafusion :
1. pipeline : UI
2. transform : UI

End to End Pipeline

Tuesday, February 20, 2024 5:10 PM





Sample data

Cloud Functions

27 March 2023 08:52

Google Cloud Functions is a serverless computing service provided by Google Cloud Platform (GCP). In simple terms, a serverless architecture means you can run your code without having to explicitly provision or manage servers.

Here's an overview of Google Cloud Functions:

Serverless Computing:

- With Cloud Functions, you can write and deploy single-purpose functions that are triggered by various events, such as HTTP requests, changes in Google Cloud Storage, modifications in a database, or the publishing of a message in Google Cloud Pub/Sub.

Event-Driven:

- Cloud Functions are designed to be event-driven. This means your function is executed in response to specific events, and you only pay for the computation time used during the execution.

Supported Languages:

- Google Cloud Functions supports multiple programming languages, including Node.js, Python, Go, and more. You can write your functions in the language you are most comfortable with or that best suits your application.

Scalability:

- Cloud Functions automatically scales based on demand. If your function receives more events, Google Cloud Platform automatically allocates more resources to handle the increased load, and scales down when the demand decreases.

Integration with GCP Services:

- Cloud Functions seamlessly integrates with other Google Cloud Platform services, allowing you to build serverless applications with various GCP components.

Stateless and Short-Lived:

- Functions are designed to be stateless and short-lived. They perform a specific task and then terminate. If you need to maintain state or run tasks continuously, other GCP services like Google Cloud Run or Compute Engine might be more suitable.

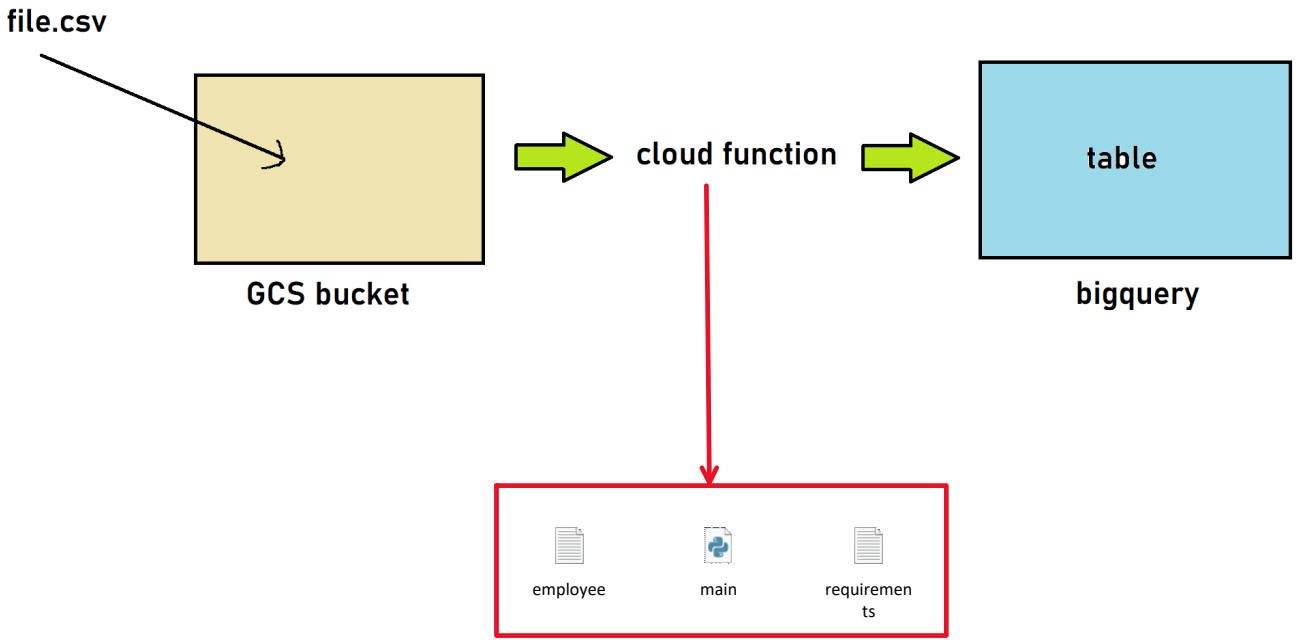
Billing Model:

- With serverless computing, you only pay for the actual compute time your function uses. There are no charges when your function is not running.

In summary, Google Cloud Functions is a serverless compute service that allows you to run your code in response to events without the need to manage servers. It provides flexibility, scalability, and seamless integration with other Google Cloud services.

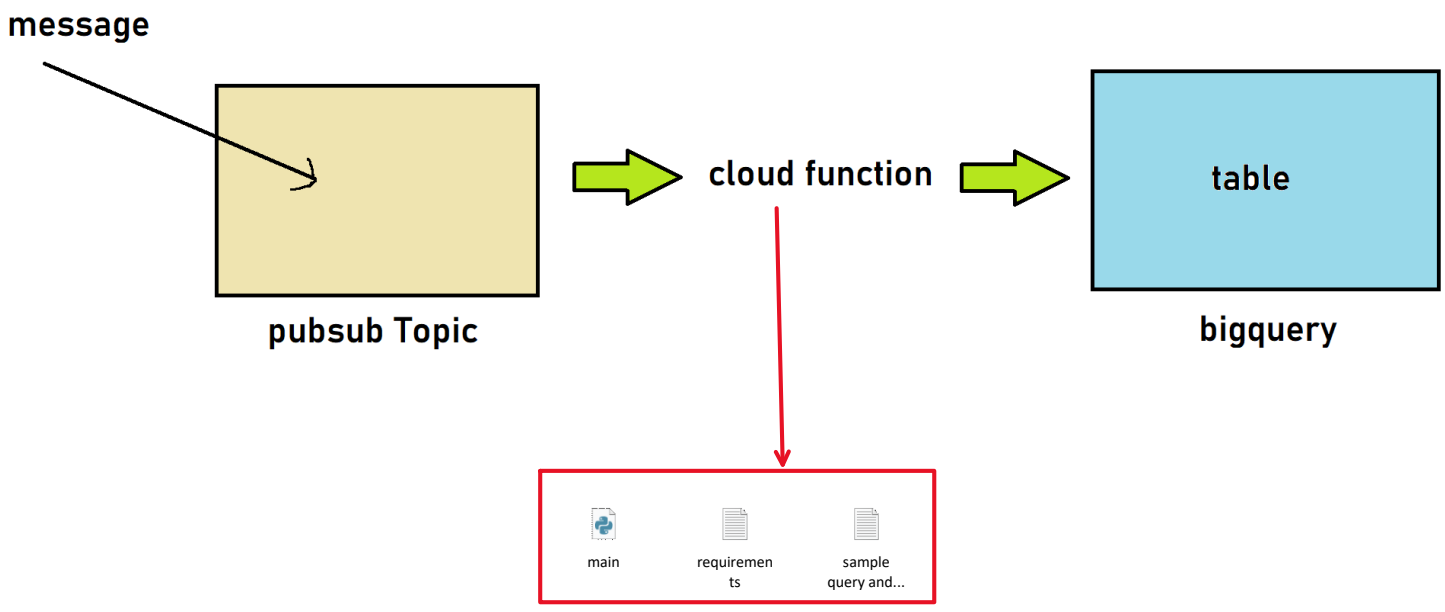
Cloud Function-1

Monday, January 8, 2024 2:26 PM



Cloud Function-2

Monday, January 8, 2024 2:26 PM



Bigtable

08 April 2023 11:03

Cloud BigTable : Loading and Querying Data with Cloud Bigtable:

- Bigtable is a fully managed, wide-column NoSQL database that offers low latency and replication for high availability.

To use Bigtable, create an instance and then set up your development environment to access Bigtable so that you can add data and monitor performance.

- BigTable doesn't support RDBMS concept.
- serverless
- Milli second latency
- Handles millions of request per second.
- It's the same database that powers many core Google services, including Search, Analytics, Maps, and Gmail.

Objective:

- Design a Bigtable schema and row key
- Parse CSV data into Bigtable
- Query data in Bigtable with HBase

Requirements:

- A GCP Project
- Cloud console
- Cloud shell
- Python Code and CSV file

Create Instance :

For Instance name, enter Quickstart instance.

For Instance ID, enter quickstart-instance.

For Storage type, select SSD.

For Cluster ID, enter quickstart-instance-c1.

For Region, select us-east1.

For Zone, select us-east1-c.

Click Create to create the instance.

Practical Implmentation:

1. Configure cbt to use your project and instance by creating a .cbtrc file,

replacing project-id with the ID for the project where you created your Bigtable instance:

- echo project = bigquery-practice-2022 > ~/.cbtrc
- echo instance = quickstart-instance >> ~/.cbtrc

2. Verify that you set up the .cbtrc file correctly:
   - cat ~/.cbtrc

3. The terminal displays the contents of the .cbtrc file, which looks similar to the following:
   - project = project-id
   - instance = <bigtable-instance name>

 # cbt cli commands:

1. Run a cbt command to verify installation of the tool:

- cbt listinstances

2. Create a table named my-table.
   - cbt createtable fires

3. List your tables:
   - cbt ls

4. Add one column family named fwi and metric:
   - cbt createfamily fires fwi
   - cbt createfamily fires metric

5. List your column families:
   - cbt ls fires

6. To install cloud-bigtable in cloud shell
   - sudo pip3 install google-cloud-bigtable

7. Upload csv files and python code : git clone https://github.com/vigneshSs-07/Cloud-AI-Analytics.git
   - dataloader.py
   - firestore.csv

8. Edit the dataloader.py with project-id and bigtable instance name:
   - vim dataloader.py

   (update your project and instance details with vim editor)
   - ls
   - cat dataloader.py

(to see the updated code and verify it once)

9. Execute the python script files:
   - python3 dataloader.py

10. Clone the public repo for the use case:
    - git clone https://github.com/ACloudGuru-Resources/Course_Google_Certified_Professional_Data_Engineer.git
    - cd cloud-bigtable-examples
    - cd quickstart

to land into hbase terminal :
- ./quickstart.sh

11. Hbase terminal will open after executing quickstart.sh file:
    - scan 'fires'
    - scan 'fires', {ROWPREFIXFILTER => "2#2#", COLUMNS => "metric:area"}
    - scan 'fires', {ROWPREFIXFILTER => "2#2#aug#", COLUMNS => "metric:area"}

Resources:
- https://cloud.google.com/bigtable/docs/create-instance-write-data-cbt-cli
- https://cloud.google.com/bigtable/docs/cbt-reference

Create a table named my-table, with one column family named cf1:

```
create 'my-table', 'cf1'
```

List your tables:

```
List
```

Put the value test-value in the row r1, using the column family cf1 and the column qualifier c1:

```
put 'my-table', 'r1', 'cf1:c1', 'test-value'
```

Use the scan command to scan the table and read the data you added:

```
scan 'my-table'
```

Delete the table my-table:

```
disable 'my-table'
drop 'my-table'
```

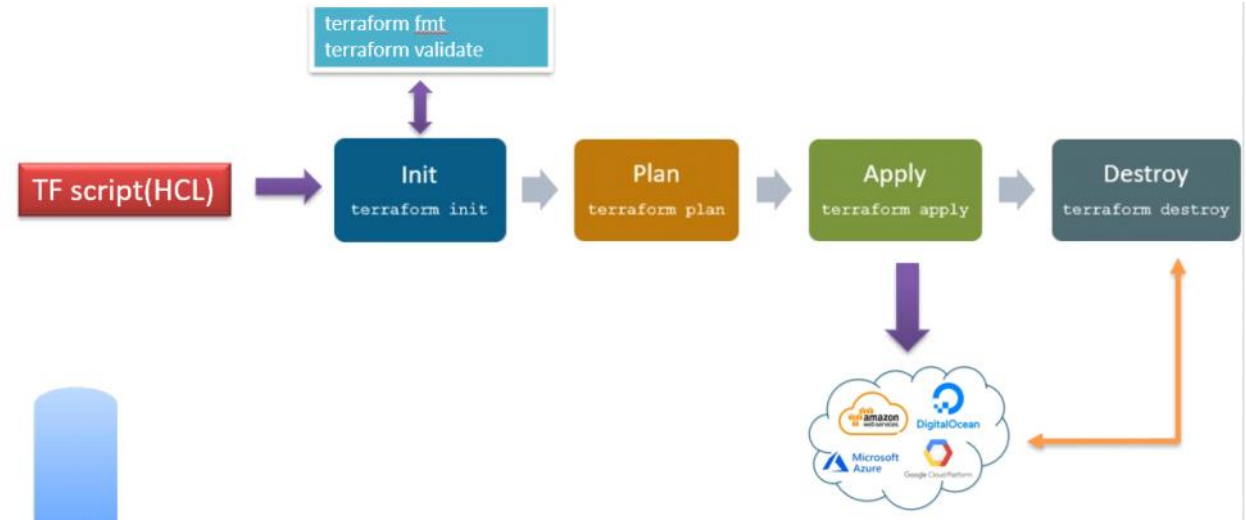
Terraform

Thursday, October 19, 2023 8:42 AM

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.

Terraform can manage existing and popular service providers as well as custom in-house solutions.

- Terraform is an open source software
- Terraform software developed by HashiCorp
- Terraform software developed using GO language
- Terraform will be called as IAC software
- IAC means Infrastructure as Code
- Terraform supports almost all cloud platforms
- Terraform scripts are re-usable
- We need to use HCL for terraform scripts



Terraform is an open-source infrastructure as code (IaC) tool developed by HashiCorp. It enables users to define and provision infrastructure resources using a high-level configuration language. Terraform is designed to help automate the creation, modification, and management of infrastructure across various cloud providers, on-premises environments, and third-party services.

Here's some basic information about Terraform:

Key Concepts:

- Infrastructure as Code (IaC): Terraform allows you to define and manage infrastructure using code, which is typically written in HashiCorp Configuration Language (HCL). This approach provides several benefits, including version control, automation, and repeatability.
- Declarative Syntax: Terraform uses a declarative syntax, meaning you describe the desired state of your infrastructure, and Terraform will work to make the current state match the desired state.

Terraform Lifecycle: The typical lifecycle of using Terraform involves several steps:

- Authoring Configuration: You create configuration files (usually with a .tf extension) that describe the infrastructure you want to provision. These files

include resource definitions, providers, variables, and outputs.

- Initializing: Run **terraform init** to initialize a working directory. This step downloads the necessary providers and modules defined in your configuration.
- Planning: Use **terraform plan** to create an execution plan. Terraform analyzes your configuration and calculates the actions it needs to take to achieve the desired state. It also shows you any changes that will be made to your infrastructure.
- Applying: Run **terraform apply** to execute the changes outlined in the plan. Terraform will create, update, or delete resources as necessary to match the desired state.
- Destroying: When you want to tear down resources, you can use **terraform destroy** to remove them. Be cautious when using this command in a production environment.
- Managing State: Terraform keeps track of the state of your infrastructure in a state file (by default named terraform.tfstate). It's important to manage this file carefully to prevent conflicts and ensure consistency.

Terraform Architecture:

Terraform follows a client-server architecture:

- Client: The client is where you run Terraform commands (e.g., init, plan, apply). It communicates with cloud providers, interacts with the configuration, and manages the state.
- Providers: Providers are responsible for managing API interactions with specific cloud providers or services (e.g., AWS, GCP, Azure). They translate Terraform configuration into API calls to create, update, and delete resources.
- State: The state file (terraform.tfstate) keeps track of the current state of your infrastructure. It's a JSON file that records resource IDs and metadata. State management is crucial to Terraform's functionality.
- Resource Graph: Terraform builds a dependency graph based on your configuration. It understands which resources depend on others and ensures that they are created or updated in the correct order.
- Plugins: Terraform uses plugins to interact with providers. Each provider has its plugin that extends Terraform's functionality to manage resources in that provider's ecosystem.