A logo with a circular design

Description automatically generated with medium confidence

# End-to-End Data Transformation and Analytics using DBT and Google Big Query

**Team Composition:**

*Jeyadev L - G23AI2071*

*Bratati Rout - G23AI2074*

*Sharone Verma - G23AI2089*

# Overview

This project is an end-to-end data pipeline designed to deliver efficient, scalable, and real-time analytics using modern cloud-native technologies. Built with Apache Airflow, DBT (Data Build Tool), and Google Cloud Platform (GCP) services, the pipeline facilitates seamless data ingestion, transformation, and visualization. It integrates BigQuery for powerful data warehousing and leverages Kubernetes for scalable deployments.

## Key Features

1. **Modular Architecture:**
   * Apache Airflow orchestrates tasks with clear dependencies and retries for fault tolerance.
   * DBT models enable modular, testable, and documented SQL transformations.
2. **Cloud-Native Scalability:**
   * Uses Google Kubernetes Engine (GKE) for deployment, ensuring high availability and horizontal scaling.
3. **Data Warehousing with BigQuery:**
   * Optimized for fast, cost-efficient data queries and supports real-time analytics.
4. **Visual Insights:**
   * Provides dashboards for analytics trends and booking metrics using pre-designed BigQuery views.
5. **Automation and CI/CD:**
   * GitHub Actions automate testing and deployment, ensuring a robust development workflow.

## Best Practices

* Modularize DBT transformations for clarity and reusability.
* Utilize Airflow DAGs for robust scheduling and retry mechanisms.
* Monitor costs and performance in BigQuery to optimize resource utilization.
* Implement CI/CD pipelines to streamline code changes and deployments.

# Architecture

1. **Airflow DAGs**:
   * Orchestrates the end-to-end data pipeline, including ingestion, transformation, and deployment.
   * Python scripts for custom DAGs are provided in Data\_pipeline.py and main.py.
2. **DBT (Data Build Tool)**:
   * Manages transformations in BigQuery through a set of SQL models.
   * Staging, transformation, and final analytical metrics are defined in SQL files.
3. **BigQuery**:
   * Acts as the data warehouse where the pipeline processes, aggregates, and analyses hotel booking data.
4. **Deployment**:
   * Uses Docker containers orchestrated via Kubernetes (GKE) for scalability and robustness.
   * Configuration files for deployment are provided for both Airflow and DBT in YAML format.

A screenshot of a computer

Description automatically generated

# File Details

**1**. Python Scripts

* **Data\_pipeline.py**
  + Contains the Airflow DAG definition for the pipeline.
  + Tasks:
    - Ingestion of raw data into BigQuery.
    - Execution of DBT transformations.
    - Generation and serving of DBT documentation.
* **main.py**
  + A helper script for utility tasks in the pipeline.

## 2. SQL Files

* **stg\_hotel\_bookings.sql**
  + SQL script to create a staging table in BigQuery for raw hotel booking data.
  + Ensures data cleansing and schema standardization.
* **final\_booking\_analytics.sql**
  + Defines the final analytical model, aggregating metrics like total bookings, revenue, and average lead time.
* **trn\_booking\_metrics.sql**
  + Contains transformations for intermediate analytical metrics.

## 3. YAML Files

* **sources.yml**
  + DBT source configuration defining the raw tables and schema metadata.
  + Enables lineage tracking in DBT.
* **airflow\_deployment.yaml**
  + Kubernetes manifest for deploying Airflow on GKE.
  + Includes configuration for webserver, scheduler, worker, Redis, and Postgres.
* **dbt\_gke\_deployment.yaml**
  + Kubernetes manifest for deploying DBT on GKE.
  + Contains configuration for DBT container, BigQuery credentials, and resource limits.

# Deployment Guide

## 1. Local Setup

* Install the required tools:
  + Docker, Kubernetes CLI, kubectl, and Helm.
* Pull necessary Docker images for Airflow and DBT.
* Configure .env files for credentials and configurations.

## 2. Airflow Deployment

* **Steps**:
  + Use airflow\_deployment.yaml to deploy Airflow components on GKE.
  + Run kubectl apply -f airflow\_deployment.yaml.
  + Verify deployment via kubectl get pods.

## 3. DBT Deployment

* **Steps**:
  + Deploy DBT containers on GKE using dbt\_gke\_deployment.yaml.
  + Apply the YAML manifest via kubectl apply -f dbt\_gke\_deployment.yaml.

## 4. Running the Pipeline

* Access Airflow UI and trigger the DAG from the Data\_pipeline.py.
* Monitor the tasks:
  + Data ingestion → DBT transformations → Documentation generation.

# BigQuery Analysis

* **Tables**:
  + stg\_hotel\_bookings: Raw data after staging.
  + trn\_booking\_metrics: Intermediate transformations.
  + final\_booking\_analytics: Final aggregated metrics.
* **Sample Queries**:
  + Extract top-performing months:

A black screen with white text

Description automatically generated

* + Find average lead time:

A black screen with white text

Description automatically generated

# Key Features

## 1. Airflow Integration

Apache Airflow is a key component of this pipeline, orchestrating the entire ETL (Extract, Transform, Load) process. The integration is designed to ensure seamless task execution, monitoring, and error handling.

### Features:

* **Task Orchestration**:
  + The DAG (Data\_pipeline.py) defines a sequence of tasks, including:
    - Data ingestion from external sources.
    - Execution of DBT transformations on BigQuery.
    - Generation of DBT documentation.
  + Tasks are interconnected with clear dependencies to ensure correct execution order.
* **Error Handling**:
  + Airflow provides detailed logs for each task, helping debug errors during DAG execution.
  + Tasks have retry policies to handle transient errors.
* **Scheduler**:
  + The Airflow Scheduler automates task execution based on the defined schedule (e.g., daily runs).
  + Ensures that DAGs are re-run if a previous run fails and retries are configured.
* **Extensibility**:
  + New tasks can be easily added to the DAG by defining Python operators or using existing Airflow operators.

### Advantages:

* Centralized orchestration of data workflows.
* Scalable architecture to handle multiple concurrent DAGs.
* Customizable task dependencies.

## 2. DBT for Transformation

Data Build Tool (DBT) is a core transformation engine in this pipeline, enabling modular and version-controlled SQL transformations.

### Features:

* **Modular Transformations**:
  + The SQL transformations are broken into stages:
    - **Staging Models** (stg\_hotel\_bookings.sql):
      * Cleans and standardizes raw data.
      * Serves as the foundation for further transformations.
    - **Intermediate Models** (trn\_booking\_metrics.sql):
      * Applies business logic to calculate intermediate metrics like average lead time and booking frequency.
    - **Final Models** (final\_booking\_analytics.sql):
      * Aggregates metrics to provide actionable insights.
* **Testing**:
  + DBT provides built-in tests for:
    - Schema validation (e.g., column existence).
    - Data integrity (e.g., non-null checks).
  + Ensures that all transformations produce reliable outputs.
* **Documentation**:
  + DBT generates an interactive web-based documentation that includes:
    - Data lineage: Traceability from raw data to final outputs.
    - Model descriptions: Metadata about the purpose and functionality of each model.
* **Source Configuration**:
  + Sources (sources.yml) define the input datasets, ensuring clear documentation and lineage tracking.
  + Includes metadata such as source table names and schema definitions.

### Advantages:

* Simplified and version-controlled SQL development.
* Reusable and modular transformations.
* Built-in testing and documentation reduce errors and improve maintainability.

## 3. BigQuery as the Data Warehouse

Google BigQuery is the primary storage and query engine in the pipeline, enabling high-performance analytics on large datasets.

### Features:

* **Scalable Query Execution**:
  + BigQuery handles petabyte-scale datasets, making it suitable for large-scale analytics workloads.
* **Partitioning and Clustering**:
  + Tables are partitioned (e.g., by date) and clustered (e.g., by hotel type) to optimize query performance.
* **Real-time Analytics**:
  + BigQuery can ingest data in near real-time, making it possible to analyze up-to-date booking trends and metrics.
* **Integration with DBT**:
  + SQL models in DBT directly target BigQuery, leveraging its performance and scalability.
* **Interactive Analytics**:
  + Users can run ad-hoc queries to explore insights, such as:
    - Top-performing hotels.
    - Monthly revenue trends.
    - Booking behaviour by lead time.

### Advantages:

* Highly performant for analytical workloads.
* Fully managed, reducing operational overhead.
* Easy integration with other GCP services (e.g., GKE, Cloud Storage).

## 4. Kubernetes Deployment

Kubernetes (GKE) is used to deploy both the Airflow and DBT components, ensuring scalability, reliability, and high availability.

### Features:

* **Containerized Deployment**:
  + Both Airflow and DBT are deployed as Docker containers.
  + Ensures portability across different environments (e.g., local, staging, production).
* **Scalability**:
  + GKE automatically scales resources based on workload demands, ensuring high performance during peak loads.
* **Resource Management**:
  + YAML configuration files (airflow\_deployment.yaml and dbt\_gke\_deployment.yaml) define resource limits and requests (e.g., CPU, memory) for each component.
* **Networking**:
  + Services are exposed through Kubernetes Services, allowing inter-container communication and external access.
* **Monitoring and Logging**:
  + GKE integrates with Google Cloud Monitoring and Logging, providing real-time insights into container health and performance.

### Advantages:

* Ensures reliability through automatic failover and self-healing.
* Simplifies deployment and scaling.
* Reduces infrastructure management overhead.

## 5. Comprehensive Monitoring and Documentation

The pipeline is designed with observability and transparency in mind, making it easy to monitor performance and understand the data flow.

### Monitoring:

* **Airflow Logs**:
  + Detailed logs for each task, including execution times, errors, and retries.
  + Accessible through the Airflow UI.
* **BigQuery Monitoring**:
  + Query performance and cost are tracked via the Google Cloud Console.
* **Kubernetes Monitoring**:
  + GKE provides real-time metrics and logs for deployed containers.

### Documentation:

* **DBT Documentation**:
  + Automatically generated by DBT and served via a web interface.
  + Includes:
    - Data lineage diagrams.
    - Model descriptions.
    - Test results.
* **Airflow Documentation**:
  + DAGs are self-documenting, showing task dependencies and execution order in the Airflow UI.

### Advantages:

* Improves transparency and accountability.
* Simplifies debugging and performance tuning.
* Enables better collaboration among team members.

# End-to-End Flow

1. **Raw Data Ingestion**:
   * Airflow ingests raw data into BigQuery staging tables using stg\_hotel\_bookings.sql.
2. **Data Transformation**:
   * DBT applies SQL transformations in BigQuery:
     + Cleans and processes raw data into intermediate and final analytical models.
3. **Documentation and Testing**:
   * DBT generates and serves documentation.
   * Tests ensure data quality and schema compliance.
4. **Visualization and Analytics**:
   * Final metrics are available in BigQuery for visualization and ad-hoc analysis.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Best Practices for Building and Managing the Data Pipeline

Building a robust, scalable, and efficient data pipeline requires adhering to best practices. Below are some recommendations specific to the pipeline's components and operations, categorized by key areas.

## 1. Data Ingestion

* **Consistency and Formatting**:
  + Ensure that incoming data adheres to a consistent schema before ingestion.
  + Use staging layers to sanitize raw data without modifying its original format.
* **Validation**:
  + Validate incoming data for null values, duplicates, or incorrect data types.
  + Automate quality checks using tools like DBT's built-in tests.
* **Monitoring**:
  + Enable alerts for issues such as missing or delayed data files.

**Best Practice**: Use schema validation in the staging layer (stg\_hotel\_bookings.sql) to ensure downstream transformations are not affected by malformed data.

## 2. DBT Transformations

* **Modular Design**:
  + Break transformations into small, reusable, and testable steps:
    - Staging Models (e.g., stg\_hotel\_bookings.sql) handle raw data cleaning.
    - Intermediate Models (e.g., trn\_booking\_metrics.sql) apply business logic.
    - Final Models (e.g., final\_booking\_analytics.sql) aggregate insights.
* **Version Control**:
  + Commit all DBT models, tests, and configurations (sources.yml) to a version control system (e.g., GitHub).
  + Use branches for development, testing, and production environments.
* **Testing**:
  + Implement DBT’s built-in tests to validate:
    - Schema properties (e.g., not null, unique).
    - Data integrity and consistency.
  + Add custom SQL tests where built-in tests are insufficient.
* **Documentation**:
  + Use DBT’s auto-generated documentation for clear visibility into data lineage and model logic.
  + Update descriptions in sources.yml for clarity and accuracy.

**Best Practice**: Test each model independently and ensure documentation is up-to-date for all transformations.

## 3. BigQuery Usage

* **Query Optimization**:
  + Partition and cluster tables in BigQuery to improve query performance.
  + Avoid SELECT \* queries—explicitly select required columns to minimize data scans.
* **Cost Management**:
  + Use BigQuery’s query estimation feature to understand query costs before execution.
  + Regularly monitor queries for inefficiencies, such as unnecessary joins or aggregations.
* **Storage Management**:
  + Use data expiration policies to automatically delete unused datasets or partitions.
  + Archive rarely accessed data to reduce storage costs.
* **Real-Time Analytics**:
  + Enable streaming ingestion for real-time data updates if required by business needs.

**Best Practice**: Regularly audit BigQuery usage to optimize costs and performance while ensuring data compliance.

## 4. Airflow Orchestration

* **Task Design**:
  + Keep tasks small and modular to simplify debugging and enhance reusability.
  + Use task retries for transient failures and set sensible retry intervals.
* **Dependencies**:
  + Explicitly define task dependencies to avoid circular dependencies or incomplete executions.
  + Use Airflow’s “Trigger Rules” to handle conditional execution paths.
* **Logging**:
  + Centralize task logs and retain them for a sufficient duration for debugging and audits.
  + Use Airflow’s built-in monitoring to identify task failures or delays promptly.
* **Extensibility**:
  + Leverage Airflow’s plugins or custom operators to integrate with new data sources or destinations.
  + Use the Python Operator for custom scripts when pre-built operators are unavailable.

**Best Practice**: Use Airflow’s built-in UI to monitor DAG performance and ensure proper scheduling without overloading resources.

## 5. Kubernetes Deployment

* **Resource Management**:
  + Set appropriate CPU and memory limits for each container in the dbt\_gke\_deployment.yaml and airflow\_deployment.yaml.
  + Use auto-scaling policies to handle workload spikes efficiently.
* **Isolation**:
  + Deploy Airflow and DBT in separate namespaces or pods to isolate workloads.
  + Use role-based access control (RBAC) to secure deployments.
* **Monitoring and Alerts**:
  + Enable GKE monitoring and logging for real-time insights into container performance.
  + Set up alerts for issues such as resource exhaustion or failed pods.
* **Rolling Updates**:
  + Use Kubernetes’ rolling update feature to deploy changes without downtime.
  + Always test deployments in staging environments before applying them to production.

**Best Practice**: Continuously monitor GKE resource utilization and optimize deployments for cost efficiency.

## 6. Documentation and Collaboration

* **Centralized Documentation**:
  + Maintain a single source of truth for pipeline configurations, architecture diagrams, and process documentation.
  + Regularly update DBT documentation with accurate descriptions and tags for new models or changes.
* **Collaboration**:
  + Use version control tools (e.g., GitHub) for collaborative development.
  + Encourage peer reviews for all changes to SQL models, Airflow DAGs, and deployment configurations.

**Best Practice**: Ensure every team member has access to the documentation and understands the pipeline’s architecture and processes.

## 7. Security

* **Data Security**:
  + Use Google Cloud IAM roles to control access to BigQuery datasets and GKE pods.
  + Encrypt sensitive data both in transit and at rest.
* **Credential Management**:
  + Store credentials (e.g., cred.json) securely using secrets management tools (e.g., Kubernetes Secrets or Google Secret Manager).
  + Avoid hardcoding sensitive information in code or configurations.
* **Audit Logs**:
  + Enable audit logging for all data operations in BigQuery and Airflow for compliance tracking.

**Best Practice**: Regularly review and update security policies to address new threats and vulnerabilities.

# Conclusion

This data pipeline provides a foundational framework for scalable and reliable data analytics. Key achievements include:

* **Automated Data Orchestration**: The Airflow DAGs ensure seamless task dependencies and scheduling, simplifying complex workflows.
* **Transformational Excellence**: DBT enables modular, testable, and documented SQL transformations that bring clarity and accuracy to data processes.
* **Real-Time Insights**: BigQuery powers real-time analytics, unlocking actionable insights from raw data.
* **Cloud-Native Scalability**: Kubernetes deployments offer flexibility, resource optimization, and fault tolerance.

This project is designed with scalability, extensibility, and sustainability in mind. It is not just a solution for today’s analytics needs but a platform for growth and evolution. Future extensions can incorporate additional data sources, advanced machine learning models, and enhanced visualization capabilities.