ASSIGNMENT

Table of Contents

Assignment - 1

Github Link - https://github.com/Shobhit2526/assigment/tree/main/Assignment-1

Section 1: Data Source Understanding

Explain the differences between Facebook Ads, Google Ads, RDS (Relational Database Service), and CleverTap in terms of data structure, API access, and data types.

Facebook Ads

- **Data Structure:** Facebook Ads provides data in a hierarchical structure, with campaigns, ad sets, and ads at different levels. Data is typically presented in JSON format.
- **API Access:** Facebook Marketing/Graph API allows programmatic access to data. It uses OAuth 2.0 for authentication and authorization.
- **Data Types:** Data includes metrics like impressions, clicks, conversions, cost, and demographic information. Data types vary from numerical (e.g., impressions, cost) to categorical (e.g., campaign name, ad group name).

Google Ads

- **Data Structure:** Google Ads data is also hierarchical, with campaigns, ad groups, and ads as the primary units. Data is typically presented in CSV or XML format.
- **API Access:** Google Ads API provides programmatic access to data. It uses OAuth 2.0 for authentication and authorization.
- **Data Types:** Data includes metrics like impressions, clicks, conversions, cost, and demographic information. Data types are similar to Facebook Ads, with numerical and categorical data.

RDS (Relational Database Service)

- **Data Structure:** RDS stores data in a structured tabular format, with rows and columns. Data is typically stored in SQL databases like MySQL, PostgreSQL, or Oracle.
- **API Access:** RDS can be accessed using various methods:
    - Direct database connections: Using SQL queries to extract data.
    - Database APIs: Using APIs provided by the database system (e.g., JDBC, ODBC).
- **Data Types:** Data types are diverse and depend on the specific database schema. Common data types include integers, floats, strings, dates, and booleans.

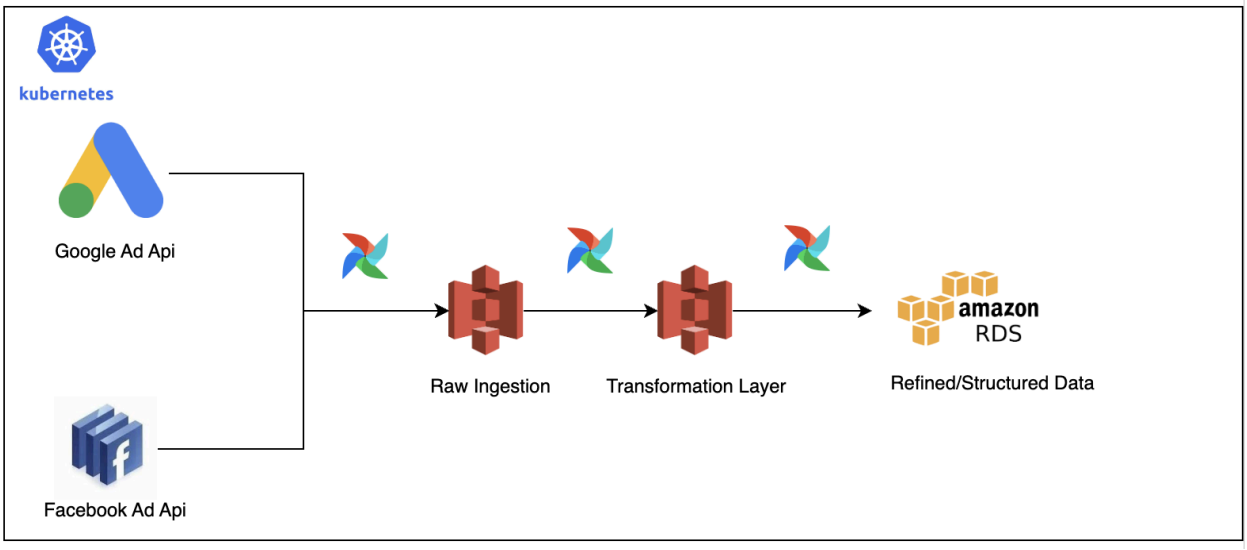
CleverTap

- **Data Structure:** Data is presented in JSON format.
- **API Access:** CleverTap provides a REST API for programmatic access to data. It uses API keys for authentication.
- **Data Types:** Data includes user events, user properties, and event properties. Data types can be numerical, categorical, or timestamp-based.

Platform	Data Structure	API Access	Data Types
Facebook Ads	Hierarchical	OAuth 2.0	Numerical, categorical
Google Ads	Hierarchical	OAuth 2.0	Numerical, categorical
RDS	Tabular	Direct connection, API, ETL	Diverse (depends on schema)
CleverTap	Structured	REST API	Numerical, categorical, timestamp-based

Section 2: ETL Pipeline Design

Design a high-level ETL pipeline architecture for extracting data from Facebook Ads and Google Ads, transforming it, and loading it into an RDS database. Consider data extraction frequency, data transformations, error handling, and scalability.



Data Extraction:

1. **Scheduling:**

    - Use a scheduler like Apache Airflow to orchestrate the ETL process.
    - Define schedules for data extraction based on the frequency required (e.g., daily, hourly).
2. **Data Source Connection:**
    - Establish connections to Facebook Ads and Google Ads APIs using OAuth 2.0 authentication.
    - Use API libraries provided by Facebook and Google to interact with their respective APIs.
3. **Data Extraction:**
    - Extract relevant data from the APIs, including:
        - Campaign performance metrics (impressions, clicks, conversions, cost, etc.)
        - Ad group performance metrics
        - Ad performance metrics
        - Demographic and geographic data
        - Other relevant data based on specific business needs

Data Transformation:

1. **Data Cleaning:**
    - Handle missing values (impute or drop)
    - Remove duplicates
    - Correct data inconsistencies (e.g., invalid dates, incorrect data types)
2. **Data Standardization:**
    - Convert data to a consistent format (e.g., date formats, currency formats)
    - Create a common data model to unify data from different sources
3. **Data Enrichment:**
    - Combine data from multiple sources (e.g., enrich ad performance data with demographic information)
    - Calculate derived metrics (e.g., return on ad spend, click-through rate)

Data Loading:

1. **Data Staging:**
    - Load the transformed data into a staging area in the RDS database.
    - Use bulk loading techniques (e.g., COPY command) for efficient data transfer.
2. **Data Validation:**
    - Validate data integrity and consistency in the staging area.
    - Perform data quality checks (e.g., data type validation, range checks).
3. **Data Loading:**
    - Load the validated data into the target tables in the RDS database.
    - Use incremental loads to update existing data and avoid overwriting historical data.

Error Handling and Monitoring:

1. **Error Handling:**
    - Implement robust error handling mechanisms to catch and log exceptions.
    - Retry failed tasks with exponential backoff.
    - Send notifications for critical errors (e.g., email alerts).
2. **Monitoring:**
    - Monitor the pipeline's health using metrics like execution time, success rate, and error rates.

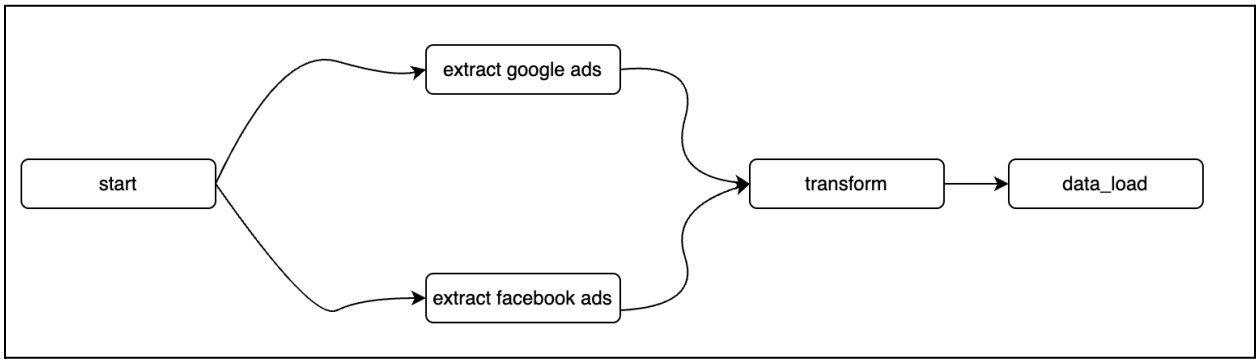
Scalability and Performance:

1. **Parallel Processing:**
    - Use parallel processing techniques (e.g., Apache Spark) to process large datasets efficiently.
    - Distribute the workload across multiple machines to improve performance.
2. **Incremental Loads:**
    - Load only the new or updated data to minimize processing time and database load.
3. **Caching:**
    - Cache frequently accessed data to reduce API calls and improve performance.
4. **Database Optimization:**
    - Optimize database queries and indexes to improve query performance.
    - Consider partitioning large tables to improve query efficiency.

Section 3 : Apache Airflow:

What is Apache Airflow, and how does it facilitate ETL pipeline orchestration? Provide an example of an Airflow DAG (Directed Acyclic Graph) for scheduling and orchestrating the ETL process described in Section 2.

- A platform to programmatically author, schedule, and monitor workflows.
- DAGs represent workflows as directed acyclic graphs.
- Nodes in a DAG represent tasks, and edges represent dependencies.



Instead of hardcoding the values like your_facebook_token, we now use environment variables with os.getenv(). This allows you to set these values in a secure way and makes the code more portable.

```
export FB_ACCESS_TOKEN="your_actual_facebook_token"
export GOOGLE_CREDENTIALS="your_google_credentials"
export DB_USER="your_db_username"
export DB_PASSWORD="your_db_password"
export DB_URL="jdbc:postgresql://localhost:5432/database"
```

Sample DAG Code

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
from pyspark.sql import SparkSession
from file_1 import FacebookExtractor, GoogleExtractor
from file_2 import DataTransformer
from file_3 import DataLoader
import os

# Initialize Spark session
spark = SparkSession.builder.appName('FacebookGoogleETL').getOrCreate()

# Create an ETL class to encapsulate logic
class FacebookGoogleETL:
    def __init__(self, spark_session):
        self.spark_session = spark_session
        self.facebook_extractor = FacebookExtractor(
            access_token=os.getenv('FB_ACCESS_TOKEN', 'tmp_facebook_token'),  # Use environment variable or fallback to placeholder
            ad_account_id=os.getenv('FB_AD_ACCOUNT_ID', 'tmp_facebook_ad_account_id'),  # Placeholder
            spark_session=spark_session
        )
        self.google_extractor = GoogleExtractor(
            credentials=os.getenv('GOOGLE_CREDENTIALS', 'tmp_google_credentials'),  # Placeholder
            client_id=os.getenv('GOOGLE_CLIENT_ID', 'tmp_google_client_id'),  # Placeholder
            customer_id=os.getenv('GOOGLE_CUSTOMER_ID', 'tmp_google_customer_id'),  # Placeholder
            spark_session=spark_session
        )
        self.transformer = DataTransformer()
        self.db_loader = DataLoader(
            db_url=os.getenv('DB_URL', 'jdbc:postgresql://localhost:5432/database'),  # Placeholder
            db_properties={
                'user': os.getenv('DB_USER', 'tmp_username'),  # Placeholder
                'password': os.getenv('DB_PASSWORD', 'tmp_password')  # Placeholder
            }
        )

    def extract_facebook_data(self):
        return self.facebook_extractor.extract()

    def extract_google_data(self):
        return self.google_extractor.extract()

    def transform_data(self, facebook_data, google_data):
        return self.transformer.transform(facebook_data, google_data)

    def load_data(self, data, table_name):
        self.db_loader.load(data, table_name)

# Define default parameters for DAG
default_args = {
    'owner': 'airflow',
    'retries': 1,
}
```

```python
# Create an ETL instance
etl = FacebookGoogleETL(spark_session=spark)

with DAG(
    'facebook_google_etl',
    default_args=default_args,
    schedule_interval='@daily',
    start_date=datetime(2024, 1, 1),
    catchup=False
) as dag:

    # Define tasks using the class methods
    extract_facebook_task = PythonOperator(
        task_id='extract_facebook',
        python_callable=etl.extract_facebook_data
    )

    extract_google_task = PythonOperator(
        task_id='extract_google',
        python_callable=etl.extract_google_data
    )

    transform_task = PythonOperator(
        task_id='transform_data',
        python_callable=etl.transform_data,
        op_args=[
            '{{ task_instance.xcom_pull(task_ids="extract_facebook") }}',
            '{{ task_instance.xcom_pull(task_ids="extract_google") }}'
        ]
    )

    load_task = PythonOperator(
        task_id='load_data',
        python_callable=etl.load_data,
        op_args=['{{ task_instance.xcom_pull(task_ids="transform_data") }}', 'ads_data']
    )

    # Set task dependencies (parallel extraction, then sequential transform and load)
    [extract_facebook_task, extract_google_task] >> transform_task >> load_task
```

Section 4: Kubernetes Integration

Explain the role of Kubernetes in deploying and managing ETL pipelines. How can Kubernetes ensure scalability, fault tolerance, and resource optimization for ETL tasks?

Kubernetes:

- Manages containerized applications.
- Ensures scalability, fault tolerance, and resource optimization.
- Can be used to deploy Airflow and ETL tasks as containers.

Kubernetes ensures efficient deployment and management of ETL pipelines by providing:

1. **Scalability**: Automatically scales ETL tasks horizontally based on resource usage or custom metrics, enabling high throughput.
2. **Fault Tolerance**: Detects and reschedules failed tasks, restarts unresponsive containers, and supports replication for high availability.

3. **Resource Optimization**: Manages CPU/memory allocation using resource requests/limits, ensuring efficient utilization and preventing resource contention.
4. **Dynamic Workload Management**: Supports batch processing through `Job` and `CronJob` for scheduled and repeatable ETL tasks.
5. **Flexibility**: Integrates with modern tools (e.g., Airflow) and supports dynamic scaling, making it ideal for complex, high-volume ETL workflows.

Section 5: Data Transformation

Given a JSON data sample from Facebook Ads containing ad performance metrics, write a Python function to transform this data into a structured format suitable for storage in an AWS Redshift database.

Transforming a JSON data sample from Facebook Ads into a structured format suitable for an AWS Redshift database. This involves:

1. **Parsing the JSON data:** Extracting relevant fields from the complex JSON structure.
2. **Data Cleaning:** Handling missing values, inconsistencies, and outliers.
3. **Data Type Conversion:** Converting data types to match the Redshift schema (e.g., strings to timestamps, numbers to decimals).
4. **Data Enrichment:** Potentially adding calculated fields or derived metrics.
5. **Data Structuring:** Organizing the data into a tabular format suitable for loading into Redshift.

Loading Data into Redshift:

Once your data is structured correctly, you can use various methods to load it into Redshift:

1. **Copy Command:** Use the `COPY` command in SQL to load data from external sources (e.g., S3) into Redshift.
2. **SQL Loader:** Use the `SQL Loader` utility to load data from flat files into Redshift.
3. **Data Integration Tools:** Utilize tools like AWS Glue or Apache Airflow to automate the data loading process.

Sample Code

Data Transformation

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Initialize Spark session
spark = SparkSession.builder.appName("FacebookAdsTransformation").getOrCreate()

def transform_facebook_ads_data(json_data):
  # Read JSON data into a DataFrame
  df = spark.read.json(spark.sparkContext.parallelize([json_data]))

  # Select and rename the necessary columns
  transformed_df = df.select(
      col("ad_id"),
      col("campaign_id"),
      col("ad_name"),
      col("impressions"),
      col("clicks"),
      col("spend"),
      col("date")
  )

  # Convert data types (optional)
  transformed_df = transformed_df.withColumn("spend", col("spend").cast("float")) \
                                 .withColumn("impressions", col("impressions").cast("int")) \
                                 .withColumn("clicks", col("clicks").cast("int"))

  return transformed_df

# Sample JSON data
json_data = [
  {
      "ad_id": "12345",
      "campaign_id": "67890",
      "ad_name": "Ad 1",
      "impressions": 1000,
      "clicks": 50,
      "spend": 120.50,
      "date": "2024-11-01"
  },
  {
      "ad_id": "12346",
      "campaign_id": "67891",
      "ad_name": "Ad 2",
      "impressions": 2000,
      "clicks": 100,
      "spend": 250.75,
      "date": "2024-11-01"
  }
]

# Transform the data
df_transformed = transform_facebook_ads_data(json_data)

# Show the transformed DataFrame
df_transformed.show()
```

Loading Data to RedShift

```
# Redshift JDBC connection details
url = "jdbc:postgresql://<redshift_host>:5439/<database>"
properties = {
    "user": "<username>",
    "password": "<password>",
    "driver": "org.postgresql.Driver"
}

# Load the transformed DataFrame into Redshift
df_transformed.write \
    .jdbc(url, "<table_name>", mode="overwrite", properties=properties)
```

Section 6: Error Handling and Monitoring

Describe strategies for handling errors that may occur during the ETL process. How would you set up monitoring and alerting mechanisms to ensure the health and performance of the ETL pipelines?

Error Handling:

- **Retry Mechanisms:** Implement retry logic for failed tasks with exponential backoff to avoid overwhelming systems.
- **Logging:** Log detailed error messages, including stack traces, to aid in troubleshooting.
- **Alerting:** Set up alerts for critical failures, such as data extraction failures or load failures.
- **Error Notifications:** Send notifications to relevant teams via email or Slack.
- **Data Quality Checks:** Validate data integrity before and after transformations.

Monitoring:

- **Pipeline Monitoring:** Use Airflow's built-in monitoring tools to track DAG runs, task durations, and failures.
- **Data Quality Monitoring:** Set up data quality checks to ensure data accuracy and completeness.
- **Infrastructure Monitoring:** Monitor the health of underlying infrastructure (e.g., databases, servers).
- **Alerting:** Configure alerts for anomalies, such as unexpected data volumes or performance degradation.

Section 7: Security and Compliance

Data security is crucial when dealing with sensitive user information. Describe the measures you would take to ensure data security and compliance with relevant regulations while pulling and storing data from different sources.

- **Data Encryption:** Encrypt sensitive data both at rest and in transit.
- **Access Controls:** Implement strong access controls to limit access to sensitive data.
- **Secure API Access:** Use API keys and OAuth tokens with appropriate permissions.
- **Compliance:** Adhere to relevant data privacy regulations and role based access.
- **Regular Security Audits:** Conduct regular security audits to identify vulnerabilities.

Section 8: Performance Optimization

Discuss potential performance bottlenecks that might arise in the ETL process, particularly when dealing with large volumes of data. How would you optimize the ETL pipeline to ensure efficient data processing?

- **Parallel Processing:** Break down tasks into smaller, parallelizable units.
- **Optimize Data Transformations:** Use efficient data processing techniques (e.g., vectorization, parallel processing).
- **Database Optimization:** Index frequently queried columns and use efficient query patterns.
- **Batch Processing:** Process data in batches to reduce load on systems.
- **Caching:** Cache frequently accessed data to reduce API calls and database queries.

Section 9: Documentation and Collaboration

How important is documentation in the context of ETL pipeline development? Describe the components you would include in documentation to ensure seamless collaboration with other team members and future maintainers of the pipeline.

- **Data and Table Lineage:** Document the flow of data from source to target.
- **Pipeline Architecture/Resources Details:** Document the overall pipeline design and components.
- **Data Transformations:** Document complex transformations and their rationale.
- **Error Handling:** Document error handling strategies and recovery procedures.
- **Best Practices:** Document coding standards, testing procedures, and deployment processes.
- **Version Control:** Use version control (e.g., Git) to track changes to code and configuration.

- **Business Logics:** Business logics used in the pipeline are very crucial and must be part of the documentation
- **Load Schedule:** ETL pipeline run load schedule should be part of the documentation

Section 10: Real-world Scenario

You have been given a scenario where CleverTap's API structure has changed, affecting your ETL pipeline. Explain the steps you would take to adapt your existing pipeline to accommodate this change while minimizing disruptions.

1. **Analyze the API Change:** Understand the impact of the change on existing data extraction and transformation logic.
2. **Update Extraction Logic:** Modify the data extraction scripts to accommodate the new API structure.
3. **Test Thoroughly:** Conduct unit and integration tests to ensure the updated pipeline works correctly.
4. **Deploy Gradually:** Deploy the changes in a controlled manner, perhaps using feature flags or A/B testing.
5. **Monitor Closely:** Monitor the pipeline closely after deployment to identify and address any issues.
6. **Document Changes:** Update documentation to reflect the changes made to the pipeline.

Assignment - 2

Github Link - https://github.com/Shobhit2526/assigment/tree/main/Assignment-2

Part 1: Data Source Integration

1. Extracting Data from Facebook Ads and Google Ads APIs

Steps to Extract Data:

1. **API Documentation Review**: Understand the endpoints, parameters, and response formats for Facebook Ads (Marketing API) and Google Ads (Google Ads API).
2. **Authentication**:
    - **Facebook Ads**: Use OAuth 2.0. Obtain an access token by creating an app in the Facebook Developer Console, and authenticate the app for required permissions.
    - **Google Ads**: Use OAuth 2.0. Create a project in the Google Cloud Console, enable the Google Ads API, and obtain the access token.
3. **Rate Limit Handling**:
    - Implement retry logic using exponential backoff when rate limits are hit.
    - Use response headers like `X-RateLimit-Limit` and `X-RateLimit-Remaining` (Facebook Ads) to monitor limits.
4. **Pagination**:
    - Use cursor-based pagination for Facebook Ads.
    - Use `page_token` for Google Ads.
5. **Data Extraction**:
    - Use APIs to fetch required metrics, dimensions, and performance data.
    - Store raw responses in a staging layer (e.g., S3 or Azure Blob Storage).

Considerations:

- Leverage batch processing for large data pulls.
- Secure API keys and tokens using environment variables or secret managers (e.g., AWS Secrets Manager).

---

2. ETL Process for Relational Databases (RDS)

Steps:

1. **Database Connectivity**:
    - Use tools like JDBC/ODBC connectors or Python libraries like `psycopg2` (PostgreSQL) and `PyMySQL` (MySQL).

2. **Incremental Load**:
    - Use timestamp columns or auto-incrementing primary keys to pull only new/updated rows.
3. **Data Extraction**:
    - Run SQL queries optimized for large datasets (e.g., indexed columns, partitioned queries).
    - Stream data in chunks using batch size.
4. **Reliability**:
    - Use transactions to ensure data consistency.
    - Monitor and log query execution times.
5. **Storage**:
    - Store extracted data in Parquet or ORC format for efficient downstream processing.

Considerations:

- Avoid locking the database with heavy queries.
- Use CDC (Change Data Capture) for real-time updates.

---

3. Extracting and Transforming CleverTap Data

Approach:

1. **Data Extraction**:
    - Use the CleverTap REST API to fetch event data.
    - Authenticate using CleverTap Account ID and Passcode.
    - Extract events in JSON format.
2. **Transformation**:
    - Parse JSON to flatten nested structures (e.g., using Python or Spark).
    - Enrich data by adding derived columns like event timestamps or user cohorts.
3. **Pipeline Components**:
    - **Extraction**: Python scripts or custom API integration tools.
    - **Transformation**: Use PySpark or Pandas for data cleansing and structuring.
    - **Loading**: Save transformed data to a cloud data warehouse (e.g., AWS Redshit).

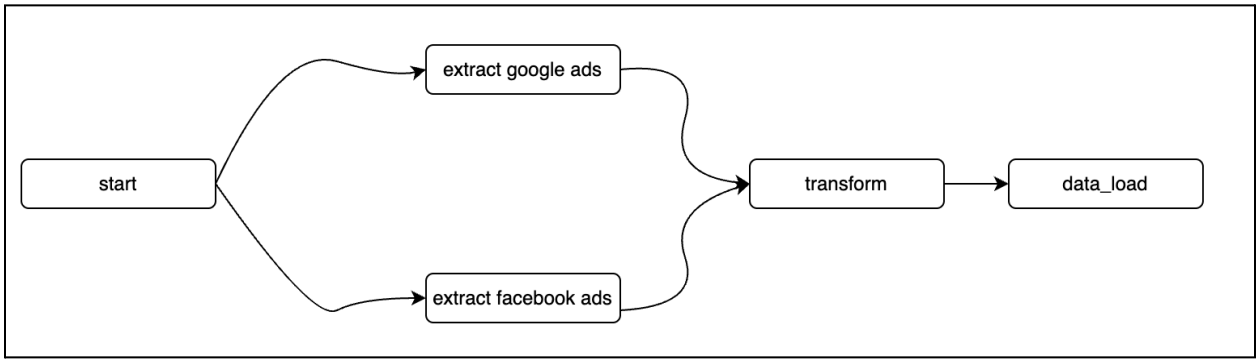
Key Considerations:

- Handle API rate limits using batch processing.
- Ensure high throughput by parallelizing API calls.

Part 2: ETL Pipeline Development

4. Apache Airflow Overview and Example

Overview: Apache Airflow is an open-source workflow orchestrator for scheduling and monitoring ETL pipelines. It uses Directed Acyclic Graphs (DAGs) to define workflows.



Example DAG:

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime

def extract_data():
    pass

def transform_data():
    pass

def load_data():
    pass

with DAG(
    dag_id='clevertap_etl_pipeline',
    start_date=datetime(2023, 1, 1),
    schedule_interval='@daily',
    catchup=False,
) as dag:
    extract = PythonOperator(task_id='extract_data', python_callable=extract_data)
    transform = PythonOperator(task_id='transform_data', python_callable=transform_data)
    load = PythonOperator(task_id='load_data', python_callable=load_data)

extract >> transform >> load
```

5. Role of Kubernetes

Enhancement via Kubernetes:

- **Scalability**: Automatically scale containers based on ETL load.
- **Fault Tolerance**: Ensure high availability of ETL jobs using container replicas.
- **Resource Optimization**: Allocate specific CPU and memory for containerized ETL jobs.

Containerization Benefits:

- **Portability**: Run ETL jobs on any Kubernetes-compatible environment.
- **Isolation**: Avoid dependency conflicts by packaging libraries in Docker images.

6. Data Transformation Challenges

Challenges:

- **Data Normalization**: Aligning disparate formats (e.g., dates, currencies).
- **Missing Data**: Imputing missing values.
- **Schema Drift**: Handling changes in source schemas.

Examples:

- Convert nested JSON to a flat structure.
- Aggregate raw event data into time-series metrics.
- Perform joins across multiple datasets for enriched views.

Part 3: Scalability and Monitoring

7. Designing for Scalability

Techniques:

- Use distributed frameworks like Apache Spark for processing.
- Partition data by time or regions for parallel processing.
- Store intermediate results in distributed storage (e.g., HDFS or S3).

8. Monitoring and Logging

Strategies:

- **Airflow Monitoring**: Use Airflow's UI for DAG status.
- **Alerts**: Set up alerts for job failures or SLA breaches.

Error Handling:

- Implement retries with exponential backoff.
- Capture detailed error logs for troubleshooting.

Part 4: Best Practices and Security

9. Data Security Measures

- Implement role-based access control (RBAC).

10. Documentation Best Practices

- Create pipeline flow diagrams.
- Document data sources, transformations, and destination schemas.
- Maintain a version-controlled repository with pipeline metadata.

Part 5: Practical Scenario

11. High-Level Architecture

Architecture Components:

- **Data Sources**: APIs (Facebook, Google, CleverTap), RDS.
- **Orchestration**: Apache Airflow.
- **Containerization**: Kubernetes.
- **Data Transformation**: PySpark jobs.
- **Storage**: Cloud data warehouse (e.g., AWS Redshift).

Data Flow:

1. Extract data via APIs and database connectors.
2. Transform data using Spark jobs.
3. Load data into a cloud data warehouse.

Tools:

- **ETL Orchestration**: Airflow.
- **Transformation**: Spark.
- **Container Management**: Kubernetes.