****Section 1: Data Source Understanding****

1. 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 Source: Focuses on using social media profiles and engagement metrics.
- ➔ Data Structure
  - Store the campaigns id, list of ads, particular ads details.
  - Quantitative Performance Metrics i.e. Impression, Clicks, Cost, Click-Through Rate, Cost Per Click, Actions taken by user after viewing ads.
  - Qualitative Data i.e. Campaign name, ads name, ad set name, demographic data (age, gender location).
- ➔ API access
  - We can access it using Graph API, a restful api provided by Facebook to interact with various facebook services like ads.
  - Authentication is using OAuth 2.0, requiring access token to retrieve the data.
  - Supports various endpoints:
    - ‘/campaigns’- fetch details about campaigns.
    - ‘/ad sets’ - fetch details about ad sets.
    - ‘/ads’ - Fetch details about ads.
    - ‘/insights’ - Fetch performance data and metrics.
- ➔ Data Types
  - Metrics: Numerical (clicks, impressions, cost).
  - Dimensions: Categorical (ad names, campaign names).
  - Dates: Timestamped data for tracking performance over time by created at, updated at.

Google Ads

- ➔ Data Source: Focuses on using search and browsing data.
- ➔ Data Structure
  - Store the campaigns id, ads group, ads, keywords.
  - Quantitative Performance Metrics i.e. Impression, Clicks, Cost, Click-Through Rate, Cost Per Click, Actions taken by user after viewing ads.
  - Qualitative Data i.e. Campaign name, ads group name, ad text.
- ➔ API access
  - A gRPC-based API provided by Google for managing Google Ads accounts and retrieving data.
  - Allows for programmatic access to manage and retrieve data related to campaigns.
  - Authentication is using OAuth 2.0, requiring access tokens to retrieve the data.
  - Google Ads Query Language (GAQL): SQL-like query data, allowing specifying fields and filters to customize the data retrieved,
  - Supports pagination for large datasets.

➔ Data Types
- Metrics: Numerical (clicks, impressions, cost).
- Dimensions: Categorical (ad names, campaign names).
- Dates: Timestamped data for tracking performance over time by created at, updated at.

RDS (Relational Database Service)

➔ Data Structure:
- Relational: Structured data stored in tables with defined schemas.
- Schemas: Define the structure of tables (columns, data types, constraints).

➔ API Access:
- SQL Queries: Accessed via SQL (Structured Query Language).
- Connection Protocols: JDBC, ODBC, or native database connectors.
- Authentication: Varies by database (user/password, IAM roles for AWS RDS).

➔ Data Types:
- Numerical: Integers, floats.
- Categorical: Text, enums.
- Dates: Timestamps, dates.
- Other: JSON, BLOBs, depending on the database.

CleverTap

➔ Data Source: Utilize in-app behavior and lifecycle data.

➔ Data Structure:
- Event-Driven: Captures user engagement events (app launches, purchases, etc.).
- User Profiles: Stores data related to user properties and segments.

➔ API Access:
- REST API: Allows for event ingestion and data retrieval.
- Authentication: API keys for authentication.
- Data Retrieval: Endpoints for fetching user profiles, events, and cohorts.

➔ Data Types:
- Events: JSON structures with various attributes (event name, properties).
- User Properties: Categorical (user demographics, preferences).
- Metrics: Event counts, engagement metrics.
- Dates: Timestamped events for tracking user activity over time.

****Section 2: ETL Pipeline Design****

2. 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:**
  - In extraction of data, using facebook and google ads API.
  - Airflow triggers the extraction jobs at scheduled intervals.
  - Custom scripts connect to Facebook Ads and Google Ads APIs to fetch the latest data.
  - Data is stored temporarily in a staging area (e.g., S3 bucket, local storage).
- **Frequency:**
  - We can perform batch extraction where we run the process at a regular interval (e.g. hourly, daily) to keep the data up-to-date.
  - Also only fetch the data that has changed since the previous extraction to minimize the requests and processing time.
- **Data Transformation:**
  - Cleaning up the data with duplicates records or any missing values.
  - Ensure that data fetched from different sources should have standard formats.
  - Aggregate the fields as per the business requirements.
  - Create a transformation script to perform data cleaning, normalization and aggregation.
  - Also define airflow tasks to execute the transformation scripts.
  - Airflow triggers transformation script after extraction.
- **Data Loading:**
  - To load the data into targeted RDS i.e. PostgreSQL, MySQL hosted on AWS.
  - For larger dataset, load the transformed data into batches.
  - Use libraries (e.g., psycopg2 for PostgreSQL) to connect and load data into the RDS.
  - Define Airflow tasks to execute the load operations.
- **Error Handling:**
  - Making sure there is proper error handling is done by logging of errors and setting up alerts for any data extraction, transform and loading issue.
  - Implement retry mechanism for transient error. (e.g. API rate limits.)
- **Scalability:**
  - **For Larger Dataset:**
    - Need to partition the data based on time or any other dimensions to manage large volumes of data effectively.

        - Ensure the target database tables are indexed appropriately for faster query performance.
    - **Infrastructure:**
        - **Kubernetes:** Deploy Airflow on Kubernetes for better scalability and resource management.
        - **Auto-scaling:** Use auto-scaling groups for the underlying infrastructure to handle varying loads.

****Section 3: Apache Airflow****

3. 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.

Apache Airflow is an open-source tool to programmatically create, schedule, and monitor workflows. It allows for defining workflows as code, ensuring that they are easy to maintain and reproduce.

Apache Airflow facilitate ETL Pipeline Orchestration as:

1. **Task Scheduling:** It helps users to schedule the data to run at specific intervals of time.
2. **Task Dependency Management:** Users can define task dependency to ensure that the tasks are executed in specific order.
3. **Scalability:** Airflow supports parallel execution of tasks, allowing workflow to scale horizontally.
4. **Monitoring and Alerting:** Monitor task execution, retrying on failed tasks and manage workflows. Alerts and notifications can be set up for failures or retries.

****Section 4: Kubernetes Integration****

4. 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 provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts.

By leveraging Kubernetes, ETL pipelines can achieve high availability, scalability and efficient resource utilization, and manage complex data processing workflows.

- **Scalability**: Kubernetes can automatically scale the ETL pods based on the CPU utilization or other metrics. Also it adjusts the size of the cluster by adding or removing nodes based on demands of running pods.
- **Fault Tolerance**: If a pod fails, Kubernetes will restart it or move it to another node.
- **Resource Optimization**: Kubernetes manages resources to ensure efficient usage.

Section 5: Data Transformation

5. 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.

```
import json

import pandas as pd

from datetime import datetime

from sqlalchemy import create_engine



REDSHIFT_USER = 'your_redshift_user'

REDSHIFT_PASSWORD = 'your_redshift_password'

REDSHIFT_HOST = 'your_redshift_host'

REDSHIFT_PORT = '5439'

REDSHIFT_DB = 'your_redshift_db'

REDSHIFT_TABLE = 'your_redshift_table'



engine =
create_engine(f'redshift+psycopg2://{REDSHIFT_USER}:{REDSHIFT_PASSWORD}@{R
EDSHIFT_HOST}:{REDSHIFT_PORT}/{REDSHIFT_DB}')




def transform_facebook_ads_data(json_data):

    ads_data = json.loads(json_data)
```

```
    transformed_data = []


    for ad in ads_data:
        transformed_entry = {
            "id": ad["id"],
            "name": ad["name"],
            "adset_id": ad["adset_id"],
            "campaign_id": ad["campaign_id"],
            "clicks": int(ad["clicks"]),
            "impressions": int(ad["impressions"]),
            "spend": float(ad["spend"]),
            "created_time": datetime.strptime(ad["created_time"],
"%Y-%m-%dT%H:%M:%S%z"),
            "updated_time": datetime.strptime(ad["updated_time"],
"%Y-%m-%dT%H:%M:%S%z")
        }
        transformed_data.append(transformed_entry)


    df = pd.DataFrame(transformed_data)
    return df


json_data = '''
[
    {
```

```
        "id": "123",
        "name": "Ad 1",
        "adset_id": "456",
        "campaign_id": "789",
        "clicks": "100",
        "impressions": "1000",
        "spend": "50.00",
        "created_time": "2023-01-01T00:00:00+0000",
        "updated_time": "2023-01-02T00:00:00+0000"
    },
    {
        "id": "124",
        "name": "Ad 2",
        "adset_id": "457",
        "campaign_id": "790",
        "clicks": "150",
        "impressions": "1500",
        "spend": "75.00",
        "created_time": "2023-01-01T00:00:00+0000",
        "updated_time": "2023-01-02T00:00:00+0000"
    }
]
'''
```

```
df = transform_facebook_ads_data(json_data)

print(df)

df.to_sql(REDSHIFT_TABLE, engine, index=False, if_exists='replace')
```

****Section 6: Error Handling and Monitoring****

6. 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**:
  - Use try-except blocks to catch and handle exceptions.
  - Data validations and quality checks.
  - Implement retries for transient errors.
  - Define fallback procedures for critical failures like keeping a backup database or store temporary data.
  - Log errors for later analysis. Use logging libraries such as Python's `logging` module.
- **Monitoring and Alerting**:
  - Use Airflow's built-in monitoring capabilities.
  - Set up alerts using tools like Prometheus and Grafana.
  - Cloud monitoring services tools like AWS CloudWatch, Google Cloud Monitoring, or Azure Monitor.
  - Monitor logs for errors and performance issues.
  - Implements health checks and heartbeat signals to ensure ETL jobs are running properly.

****Section 7: Security and Compliance****

7. 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.

- **Encryption**: Use HTTPS/TLS to encrypt data in transit and at rest.
- **Access Controls**: Use IAM roles and policies to restrict access.
  - **Role Based Access Control:** Implement RBAC to manage permissions based on user roles.
  - **Multi-Factor Authentication:** Require MFA for accessing systems that handle sensitive data to add an extra layer of security.

- **Compliance**: Ensure compliance with GDPR, CCPA, and other regulations.

****Section 8: Performance Optimization****

8. 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?

- **Efficient Queries**: Optimize SQL queries for faster execution.
- **Batch Processing**: Process data in batches to reduce load.
- **Parallel Processing**: Use parallel processing to speed up ETL jobs.
- **Indexing**: Index the database tables to improve query performance.

****Section 9: Documentation and Collaboration****

9. 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.

Documentation ensures that the ETL pipeline is understandable and maintainable by other team members. Key components include:

- **Overview**: High-level description of the pipeline.
- **Setup Instructions**: How to set up and configure the pipeline.
- **Code Documentation**: Inline comments and docstrings.
- **Error Handling**: How errors are handled and where to find logs.
- **Dependencies**: List of dependencies and how to install them.

****Section 10: Real-world Scenario****

10. 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.

Steps to adapt the ETL pipeline to accommodate API changes:

1. **Identify Changes**: Understand the changes in the API structure.
2. **Update Code**: Modify the extraction code to handle the new API structure.
3. **Test Changes**: Test the modified code to ensure it works as expected.
4. **Deploy Changes**: Deploy the updated code to production with minimal downtime.
5. **Monitor**: Monitor the pipeline to ensure it runs smoothly.