Explanation of the Flow & Component Choices

This architecture follows a modern **ELT (Extract, Load, Transform)** paradigm, prioritizing fast ingestion and leveraging the power of the cloud data warehouse for transformations.

a. Component Choices Rationale

- **Ingestion (S3/GCS & Apache Kafka):** The pipeline uses two entry points to handle different data types.
    - **S3/GCS (for Batch):** The web scraper and manual data uploads are batch processes that produce files (JSON, CSV). A cloud storage bucket is the most cost-effective, durable, and scalable place to land these files.
    - **Apache Kafka (for Streaming):** Chosen for its high-throughput, low-latency, and fault-tolerant capabilities. It acts as a central "buffer" for all real-time event streams, decoupling the data producers (app, database, AI engine) from the consumers (Spark). This prevents data loss and allows processing systems to be taken offline without disrupting data collection. **Debezium** is used with Kafka Connect to efficiently stream every `INSERT`, `UPDATE`, and `DELETE` from the internal sales database using Change Data Capture (CDC), avoiding the need for complex API development.
-
- **Processing (Apache Spark):** Spark is chosen for its powerful unified processing engine. With **Spark Structured Streaming**, it can consume data from both Kafka (streaming) and S3 (batch) using a consistent API. It will be used for light, stateless transformations: parsing, cleaning (e.g., standardizing date formats), and enriching data before loading it into the warehouse's raw layer.
- **Storage (Snowflake):** Snowflake is selected as the cloud data warehouse for several key reasons:
    - **Separation of Storage and Compute:** Allows us to scale compute resources up for heavy transformation jobs (with DBT) and down for normal querying, optimizing costs.
    - **Native Semi-Structured Data Support:** Its `VARIANT` data type can ingest JSON from any source without needing a predefined schema, making it perfect for our raw landing zone and gracefully handling schema evolution.
    - **Performance:** Excellent query performance for analytics and BI.
-
- **Orchestration & Transformation (Apache Airflow & DBT):** This pair forms the modern "control plane" for the data platform.
    - **Airflow:** As the primary orchestrator, Airflow is responsible for scheduling and managing dependencies. It will trigger the web scraping script on a schedule, run

Spark jobs to load data from S3, and trigger DBT runs after new raw data has been loaded.
    - **DBT (Data Build Tool):** DBT handles the "T" (Transform) in our ELT process. It allows data analysts to build, test, and document complex data models inside Snowflake using only SQL. It's perfect for transforming the raw, schemaless data into the clean, structured star schema needed for analytics.
-

b. Handling of Real-time vs. Batch Data

The architecture is designed with two distinct paths that converge in the warehouse:

- **The Near-Real-Time Path:**
    1. User app logs, AI pricing data, and internal sales changes are captured instantly and streamed into Kafka topics.
    2. An always-on Spark Structured Streaming job reads these events in small micro-batches (e.g., every minute).
    3. Spark cleans the data and appends it to the corresponding raw tables in Snowflake.
    4. This ensures data is available for querying in the raw layer within minutes of creation.
-
- **The Batch Path:**
    1. Airflow triggers the Python web scraper on a schedule (e.g., every 6 hours).
    2. The scraper outputs its JSON results to a specific folder in the S3 bucket.
    3. Manual data is uploaded to a different folder in S3.
    4. Airflow triggers a Spark batch job (or uses Snowflake's Snowpipe for auto-ingestion) to load these new files into the raw tables in Snowflake.
-

Finally, an Airflow-scheduled DBT job runs periodically (e.g., every hour) to transform the newly arrived data from *both paths* in the raw layer into the final, clean `Transformed Layer`.

c. Data Quality and Schema Evolution

- **Data Quality:**
    1. **At Ingestion:** Spark jobs can perform initial validation. Malformed records that fail basic parsing can be routed to a "dead-letter queue" or a separate S3 location for later inspection.

2. **In the Warehouse:** DBT is the primary tool for quality enforcement. We can write **DBT tests** to assert conditions on our final models (e.g., `not_null`, `unique`, `referential_integrity`). These tests run automatically with every DBT execution, preventing bad data from propagating to BI dashboards.

- **Schema Evolution:**
    1. **Streaming Sources:** We can use a **Schema Registry** with Kafka. This ensures that any data written to a topic conforms to a predefined schema, and it manages how schemas can evolve over time (e.g., only allowing backward-compatible changes).
    2. **Warehouse:** This is a key strength of using Snowflake's `VARIANT` type in the raw layer. If the scraper starts collecting a new field or a source API adds a new attribute to its JSON payload, the pipeline will not break. The new data will be loaded into the `VARIANT` column, and we can adjust our downstream DBT SQL models to parse and incorporate the new field without any downtime.

d. Partitioning and Performance in Snowflake

In Snowflake, performance is primarily managed through a concept called **Clustering**. Instead of manual partitioning, we define a "cluster key" on our large tables. Snowflake then automatically manages the sorting and organization of data in its micro-partitions based on that key.

- **Fact Table Strategy:** Our largest table will be the fact table (`FCT_PHONE_TRANSACTIONS`). We would define a multi-column cluster key on the columns most frequently used for filtering in analytical queries. A good choice would be:
    - `CLUSTER BY (TO_DATE(TRANSACTION_TIMESTAMP), PRODUCT_ID, LOCATION_ID)`
    - **:** Date is almost always used for filtering, so clustering by it provides excellent query pruning.
    - **and** These are high-cardinality foreign keys that will likely be used in `WHERE` clauses for analysis.

This strategy ensures that when a user queries for a specific product in a specific location over the last month, Snowflake can scan a minimal amount of data, dramatically improving query performance.