Advanced Data Engineer Interview Q&A Guide

Table of Contents

System Design & Architecture

Q: Design an end-to-end clickstream data pipeline for millions of user events per minute. Assume a web-scale environment.

A: For high-velocity clickstream, I'd start with Kafka (or Kinesis) for ingestion, partitioned by user or region for scalability. Spark Structured Streaming processes sessions using windowed aggregations and watermarking for late data. Data lands in S3 as partitioned Parquet files, cataloged by Glue. Gold aggregations refresh dashboards in Redshift or Snowflake. Monitoring is via CloudWatch, lag tracked with Prometheus, lineage via OpenLineage/Atlas.

Q: How would you handle schema evolution and incompatible changes in a CDC pipeline from MySQL to Snowflake?

A: Use Debezium/Kafka Connect for CDC, Avro/Protobuf schema registry for evolution, with consumers applying version-aware deserialization. Incompatible changes trigger fallback logic: new columns default null, breaking updates staged in side tables. Document all changes via schema history tracking. Data validation at ingest and ELT layer catches drift; alerts trigger incident response.

Q: Design a GDPR-compliant delete pipeline for an analytics lakehouse.

A: All personal data is tagged at ingest with unique row identifiers. For deletes, use ACID-compliant table formats (Delta Lake, Iceberg) to perform soft deletes and time-travel audits. Deletion requests issued via service API; batch jobs propagate updates across all downstream tables. Lineage tools identify all relevant datasets; compliance validated through regular audits.

Q: Build a multi-region data replication pipeline with failover for low-latency analytics.

A: Use Kafka MirrorMaker or AWS MSK cross-region replication. For writes, partition by region and leverage exactly-once semantics where possible. Storage is multi-zone S3 or Snowflake cross-region. Monitoring includes replication lag and failover triggers. Consistency ensured via conflict resolution logic (timestamp ordering), with DR exercises simulated quarterly.

Spark/PySpark Deep Dive

Q: Explain how Spark manages memory during shuffle operations and tuning techniques for large joins.

A: Spark stages shuffle files in off-heap memory, falling back to disk when executor memory limits are hit. For large joins: increase 'spark.sql.shuffle.partitions', utilize broadcast joins for small tables, optimize data serialization (Kryo), ensure even partitioning. Monitor UI for spilled tasks; tune JVM options for GC. Regularly review cluster size and executor configs for optimal utilization.

Q: How do you detect and fix skewed joins in PySpark?

A: Profile join key distributions with DataFrame groupBy, identify skewed keys, apply salting to randomize skewed values, and use 'skewed' hints if supported. Filter high-skew keys for separate processing. Consider pre-aggregating data to reduce partition pressure.

Airflow Advanced Topics

Q: Explain dynamic DAG generation and pitfalls. What patterns do you use for hundreds of pipelines?

A: For dynamic DAGs (e.g., pipeline-per-table), parameterize pipeline logic via Jinja templates or factory functions. Register DAGs programmatically. Pitfalls include excessive DAG parsing time, scheduling bottlenecks, and XCom bloat. Patterns include grouping pipelines, using TaskGroups, Airflow Variables for configuration, and DAG versioning for maintainability.

Q: How do you ensure idempotency and error recovery in Airflow DAGs?

A: Design tasks to check for prior completion, use atomic S3 writes or upsert logic, and maintain checkpoints for batch processes. Error recovery via retries, sensors to await dependencies, and on_failure_callback to reroute failures. Downstream tasks set to skip on upstream errors to avoid partial runs.

Data Modeling & Warehousing

Q: Star schema vs. snowflake schema: advantages, drawbacks, and when to use which?

A: Star schema offers simpler design and faster queries for BI; dimensions are denormalized for performance. Snowflake schema normalizes dimensions, saves storage, enforces consistency, and is best for highly variable or large dimensions (e.g., products). For most reporting, star schema is preferred; use snowflake for complex data relationships and strict normalization.

Q: Implement SCD Type 2 in SQL/Databricks for customer data.

A: Assign row effective dates and expiration dates. On change, insert a new row for updated customer, archive previous record by setting end_date. Use merge/upsert logic in Databricks SQL or Spark. For queries, filter where NOW() between effective and expiration dates.

Advanced SQL & Python

Q: Write a query to find gaps in a sequential order stream (e.g., missing order IDs).

A:

```
SELECT order_id + 1 AS start_gap,
       LEAD(order_id) OVER (ORDER BY order_id) - 1 AS end_gap
FROM orders
WHERE LEAD(order_id) OVER (ORDER BY order_id) - order_id &gt; 1;
```

Q: Python: How do you efficiently process a 100GB CSV file for deduplication?

A: Use Dask or Spark for distributed processing; for on-disk Pandas, chunk with `read_csv(chunksize=...)` and write deduplicated batches. Never load entirely into RAM. Store intermediate keys in HDF5 or SQLite for scalable tracking.

Behavioral (STAR)

Q: Tell me about a failure in a pipeline you designed and what you learned.

A: In a previous project, an Airflow pipeline failed silently due to missed validation checks, causing downstream data quality issues. After identifying missing null checks, I integrated data validation libraries and added automated alerting for anomalies. This improved reliability, reduced manual QA time, and taught me the value of proactive error monitoring for robust pipelines.

Q: Describe a situation where you had to scale a data solution for 10x the expected volume.

A: Faced with a surge in incoming sessions, I optimized Spark jobs by increasing shuffle partitions, switched storage formats to Parquet for compression, and transitioned to a scalable Kinesis stream for ingestion. Refactored downstream aggregations, implemented autoscaling on EMR, and validated pipeline integrity through stress tests. Result: Maintained throughput, reduced latency, and met SLA for reporting.