Data Engineering System Design – Interview Master Guide

Audience: Senior / Staff / Principal Data Engineer interviews (Atlassian, SEEK, FAANG). This guide explains scalability, reliability, maintainability, efficiency, and optimization across every data engineering stage, with spoken interview answers and Databricks + Spark + Delta Lake mappings.

1. Data Ingestion

Scalability: Use incremental ingestion and CDC so ingestion cost grows with change rate, not data size. Partition-based ingestion enables horizontal parallelism in Spark.

Reliability: Use checkpoints, retries, and atomic writes (Delta transactions). Late data handled via watermarks.

Efficiency & Optimization: Avoid full table scans, bound ingestion windows, compact small files.

Trade-offs: CDC adds operational complexity; incremental loads risk missing late data if not designed carefully.

Databricks Example: Spark Structured Streaming + Delta Change Data Feed (CDF).

Spoken Answer: I scale ingestion by reducing scanned data, using incremental and CDC pipelines with checkpointing.

2. Error Handling

Scalability: Dead-letter tables isolate bad records without stopping pipelines.

Reliability: Windowed deduplication and late-data detection prevent retries from corrupting data.

Maintainability: Centralized error patterns reduce ad-hoc fixes.

Trade-offs: Additional storage and monitoring overhead.

Databricks Example: Bad records written to quarantine Delta tables.

Spoken Answer: At scale, failures are normal; I isolate them instead of amplifying them.

3. Idempotency & Backfills

Scalability: Idempotent writes allow parallel backfills and reprocessing.

Reliability: Pipelines can be rerun safely without duplication.

Efficiency: Merge-based updates instead of full rewrites.

Trade-offs: Requires careful key and state design.

Databricks Example: Delta MERGE INTO with deterministic keys.

Spoken Answer: Idempotency is non-negotiable—every pipeline must be safely rerunnable.

4. Transformations & Business Value

Scalability: Distributed aggregations, shuffle control via partitioning and bucketing.

Efficiency: Column pruning, predicate pushdown, pre-aggregations.

Maintainability: Modular transformations, medallion architecture.

Trade-offs: Complex joins increase state and memory usage.

Databricks Example: Bronze → Silver → Gold using Spark SQL.

Spoken Answer: I design transformations around state size and shuffle cost.

5. Orchestration & Data Flow

Scalability: Fan-out patterns decouple consumers; parallel runners improve throughput.

Reliability: Dependency isolation prevents cascade failures.

Maintainability: Declarative workflows and clear ownership.

Trade-offs: More parallelism increases coordination complexity.

Databricks Example: Databricks Workflows with task dependencies.

Spoken Answer: I scale workflows by isolating dependencies, not chaining everything.

6. Storage Design

Scalability: Horizontal partitioning enables linear growth.

Efficiency: Bucketing and sorting reduce shuffle and query latency.

Reliability: ACID tables ensure consistency.

Trade-offs: Over-partitioning leads to small-file problems.

Databricks Example: Delta Lake partitioned by date, optimized with Z-ORDER.

Spoken Answer: Storage layout is my first scalability lever.

7. Security & Governance

Scalability: Fine-grained access controls avoid data duplication.

Reliability: Column masking and anonymization protect sensitive data.

Maintainability: Central policy management.

Trade-offs: Security layers add query overhead.

Databricks Example: Unity Catalog row/column-level security.

Spoken Answer: Governance must scale without copying data.

8. Data Quality

Scalability: Automated checks scale better than manual validation.

Reliability: Audit–write–publish prevents bad data propagation.

Maintainability: Schema compatibility enforcement.

Trade-offs: Strict checks can delay data availability.

Databricks Example: Expectations in Delta Live Tables.

Spoken Answer: Data without quality guarantees is just noise.

9. Observability

Scalability: Lag and skew detectors scale with time, not volume.

Reliability: Early detection of failures.

Maintainability: Clear SLAs and lineage.

Trade-offs: Metrics collection adds overhead.

Databricks Example: Lakehouse monitoring + custom metrics.

Spoken Answer: Observability is how I detect failure before users do.