Design for Scale: TikTok Data Pipeline Architecture

1. Scaling to Millions of Creators and 100M+ Videos

To handle the jump from dozens to millions of creators, the pipeline must transition from pandas to Apache Spark.
Spark distributes data across many worker nodes, enabling parallel computation.

- A 100-node Spark cluster can process billions of records in hours instead of days.
- Each executor processes a partition independently, dramatically reducing runtime.
- Move data storage into cloud object storage (S3 / GCS / Azure Blob) combined with a warehouse (BigQuery / Databricks).
- Object storage offers near-infinite capacity and durability.
- Parquet/Delta formats provide 80–90% compression savings over CSV.
- Querying last week's data scans only 7 partitions, instead of the entire dataset.
- Processing distributes evenly across partitions, improving throughput.
- Materialized views: Pre-compute high-value aggregates like daily creator metrics.
- Indexes & clustering: Optimize common filters (creator_id, category, date).

2. How would you ?

1 Incremental Updates

- Here we will take advantage of CDC (typ-2), use updated_at timestamps to detect only changed creators/videos.
- Typically only 1–5% of records change per day.
- Use ACID upserts to update existing creator/video records reliably.
- Ensures consistency even in distributed environments.
- Only reprocess partitions containing changed data.
- Reduces unnecessary computation.
- High-priority creators update every 15–30 minutes.
- Others update hourly or daily.

2. Historical Data Tracking

- Maintain daily snapshots stored per date partition.

- Query historical versions using version numbers or timestamps.
- Enables point-in-time queries like:
  "What was creator X's follower count on January 15th?"
- Hot (0–30 days), warm (30–365 days), cold (>1 year in Glacier).

- Reduces storage costs by up to 90%.

3. Idempotency

- Ensure all transformations give identical results for identical input.
- Persist pipeline progress so failures resume from the last checkpoint.
- Use Delta Lake or transactional databases to ensure atomic writes.
- Prevent duplicate writes across retries or re-runs with the use of composite keys.

4. Data Quality Strategy

- Enforce strict field types, required fields, and formats.
- Reject invalid rows and log violations.
- Validate follower counts ≥ 0, engagement rates between 0–1.
- Ensure videos reference valid creators.
- Use Z-scores to detect sudden spikes (e.g., suspicious follower jumps).
- Track null percentage and enforce thresholds (e.g., alert if >5%).
- Validate aggregations (sum of video views ≈ creator total_views).
- Composite scoring (0–100) for schema, completeness, and consistency.
- Fix common issues automatically (median imputation, type casting).

5. Monitoring, Failures & Alerting

- Implement full observability using Promethus, ELK, and Distributed tracing system
- Logs : Record processing rates, CPU/memory utilization, Data quality scores, Pipeline execution time, Cost per run
- Critical failures → PagerDuty (on-call)
- Warnings → Email Alert
- Daily summaries → Email (Down the road we can reduce it to weekly or monthly)
- Retry transient failures (1 min → 5 min → 15 min)

6. Metadata and Lineage

Use a central metadata platform (DataHub / Apache Atlas / AWS Glue/ Open metadata) to manage:

- Table schemas + evolution
- Data owners & update frequency
- PII classification
- Data dictionary (business definitions)