Nitya CloudTech

Dream.Achieve.Succeed

Azure

Data Engineering Interview Questions and Answers

1. How would you design a scalable ETL pipeline using Azure Data Factory for processing terabytes of data daily?

Answer:

- **Data Ingestion:** Use Azure Data Factory pipelines to ingest data in parallel from multiple sources (on-premise databases, APIs, files, etc.) to Azure Data Lake Storage Gen2. For large files, enable chunk-based copying and compression formats like Parquet or Avro.
- **Staging Area:** Stage the raw data in Azure Data Lake or Blob Storage. Create a naming convention for folders (e.g., date partitions) to organize data for easier processing.
- **Transformation:** Use Mapping Data Flows to process terabytes of data efficiently. Ensure transformations are optimized using partitioned datasets and compute-optimized integration runtimes.
- **Load:** Write transformed data into Azure Synapse Analytics or a Delta Lake for further analysis. Use PolyBase for bulk loading or ADF's "Bulk Copy" mode for SQL databases.
- **Scaling Mechanisms:**
  - Leverage Azure's elasticity to autoscale Integration Runtimes (IR).
  - Optimize copy activities by enabling parallel copy and partitioning.
  - Use separate pipelines for high-priority vs. batch jobs.

2. Explain how you would handle schema drift in Azure Data Factory for dynamic datasets.

Answer:

- **Dynamic Schema Handling:** Enable schema drift in Mapping Data Flows to allow automatic detection of new columns or changes in data structure.
- **Metadata-Driven Pipelines:** Store schema definitions and mapping rules in a metadata table (e.g., Azure SQL or Cosmos DB). Dynamically pass these mappings to pipelines using parameters.
- **Validation and Alerts:** Add pipeline steps for schema validation against expected structure. Use Azure Logic Apps or alerts for discrepancies.
- **Versioning:** Maintain schema versions in a metadata table to track historical changes and apply corrective transformations when necessary.

3. Monitoring and optimizing pipeline performance in Azure Data Factory

- **Monitoring Tools:**
  - Use the ADF Monitor tab to track activity runs, triggers, and Integration Runtime performance.
  - Integrate Azure Monitor for detailed logging and custom dashboards in Log Analytics.
- **Performance Tuning:**
  - Reduce unnecessary shuffles and data movement by optimizing data partitioning.
  - Use "Cache Lookup" in Data Flows to minimize repeated transformations.
  - Compress datasets for faster copy activity.
- **Alerts and Automation:** Set up alerts for pipeline failures or performance bottlenecks using Azure Monitor. Combine with Logic Apps to trigger automatic remediation steps.

4. Design and implementation of SCD (Slowly Changing Dimensions) using ADF

- **SCD Type 1:**
  - Use the "Alter Row" transformation in Mapping Data Flows to identify and update matching records.
  - Overwrite rows in the destination table.
- **SCD Type 2:**
  - Identify changes by comparing source and destination datasets using "Surrogate Key" or unique identifiers.
  - Insert updated records with new keys and "Effective From" and "Effective To" columns.
  - Use a watermark column to maintain historical versions.

- **Optimization:** Implement Synapse SQL pools with caching and proper partitioning for high-performance queries.

5. Best practices for large-scale joins and aggregations in PySpark

- **Optimizing Joins:**
  - Use broadcast() for smaller datasets to perform efficient broadcast joins.
  - Partition data on join keys to reduce shuffles.
- **Aggregations:**
  - Use groupByKey() sparingly; prefer reduceByKey() for better efficiency.
  - Repartition datasets before aggregation if default partitions are insufficient.

6. Architectural considerations for integrating Azure Synapse Analytics and Azure Databricks

- **Layered Architecture:**
  - Ingest data into Azure Data Lake using Databricks for preprocessing.
  - Store processed data in Delta Lake, maintaining ACID compliance and performance optimization.
  - Use Synapse Analytics for BI and querying.
- **Integration Points:**
  - Use Synapse Link to query Delta Lake data directly.
  - Use ADF to orchestrate workflows between Databricks and Synapse.
- **Optimization:** Implement Synapse SQL pools with caching and proper partitioning for high-performance queries.

7. Architecting a Delta Lake solution in Azure Databricks for batch and streaming workloads

- **Delta Lake Overview:** Use Delta Lake for unified batch and streaming pipelines. It ensures schema enforcement, ACID transactions, and time travel.
- **Batch:** Use spark.write.format("delta").save() to write batch jobs. Partition data by relevant columns for faster access.
- **Streaming:** Use Spark Structured Streaming with Delta Lake as the sink and source. Implement watermarking for late-arriving data and checkpointing for fault tolerance.
- **Optimization:** Enable Delta Lake Z-Order indexing and optimize the table regularly using the OPTIMIZE command.

8. Implementing a data lakehouse architecture in Azure

- Use Azure Data Lake Gen2 for storage and Delta Lake for unified batch and streaming pipelines.
- Leverage Synapse Analytics for query acceleration and visualization.
- Implement metadata and governance using Azure Purview.

9. Securing and managing access to pipelines

- Use Azure Active Directory (AAD) for RBAC.
- Secure credentials using Azure Key Vault.
- Implement auditing and logging for access monitoring.

10. Optimizing Azure Synapse Analytics for analytical queries

- Resource Tuning: Assign workloads to appropriate resource classes.
- Query Optimization: Use distribution keys and indexed tables for complex joins and large datasets.
- Caching: Use Materialized Views to precompute and cache frequent query results.

11. Spark Structured Streaming for real-time pipelines

- **Role:** Processes real-time data streams in Azure Databricks. Supports data transformations, joins, and aggregations in real time.
- **Implementation:**
  - Define input sources (e.g., Kafka, Event Hubs).
  - Use watermarking and window operations for stateful processing.
  - Write streams to Delta tables or Synapse Analytics for further analysis.

12. Handling skewed data in PySpark

- **Skew Analysis:** Use Spark UI to identify skew in stages.
- **Solutions:**
  - **Apply salting to distribute keys evenly.**
  - **Increase shuffle partitions to reduce data congestion.**

13. Architecting a multi-region disaster recovery solution for a data platform

- **Primary & Secondary Regions:** Use Azure paired regions for high availability.
  - Primary region hosts the production environment (Data Lake, Synapse, Databricks).
  - Secondary region is the failover site, configured with read-access replicas and DR pipelines.
- **Data Replication:**
  - Use Geo-redundant storage (GRS) for Azure Data Lake.
  - Enable cross-region replication for Synapse Analytics using Active Geo-Replication.
- **Failover Plan:**
  - Implement automated failover with Azure Traffic Manager to redirect workloads.
  - Keep Databricks clusters synchronized using notebooks stored in Git and metadata replicated to secondary storage.
- **Testing:** Conduct regular DR drills to validate failover readiness.

14. Key considerations for designing a metadata-driven ETL framework in Azure

- **Central Metadata Repository:** Store pipeline configurations, schema mappings, and transformation rules in Azure SQL or Cosmos DB**.**
- **Dynamic Pipelines:**
  - Use parameters to pass runtime configurations dynamically.
  - Read metadata at runtime to control flow logic, source/target paths, and transformations.
- **Error Handling:** Use metadata to log errors and implement retry logic.
- Extensibility: Design the framework to handle new datasets or transformations by updating metadata rather than code.

15. Approach to data modeling in a modern Azure architecture

- **OLTP Systems:** Normalize schemas to reduce redundancy and optimize transactional performance. Use Azure SQL Database.
- **OLAP Systems:** Denormalize data into star or snowflake schemas for faster query performance. Use Azure Synapse Analytics for analytical workloads.
- **Hybrid Approach:** Leverage Data Vault modeling for flexibility in combining transactional and analytical requirements.

16. Processing and analyzing petabytes of unstructured data in Azure Data Lake Gen2

- **Data Ingestion:** Use Azure Data Factory or Databricks to ingest unstructured data into Data Lake Gen2. Compress data using Avro or Parquet.
- **Processing:** Use Databricks with Delta Lake for efficient querying and transformations.
- **Analytics:**
  - Use Synapse Serverless SQL Pools to query unstructured data.
  - Implement machine learning on top of processed data using Azure ML.
- **Optimization:** Partition data by relevant attributes and use indexing for faster query performance.

17. Architectural considerations for a near-real-time dashboard with Synapse and Power BI

- **Ingestion:** Use Event Hubs or Azure Stream Analytics to capture real-time data.
- **Processing:** Stream data into Synapse Analytics using Spark or SQL pools.
- **Modeling:** Use Materialized Views or pre-aggregated tables for optimized query performance.
- **Visualization:** Connect Power BI to Synapse Analytics with DirectQuery mode for real-time updates.

18. Partitioning strategy for Azure Data Lake

- **Criteria for Partitioning:**
  - Use high-cardinality attributes like Year/Month/Day or specific business keys.
  - Avoid over-partitioning, which increases metadata overhead.
- **Tooling:** Use Databricks or Synapse SQL to define and manage partitions.
- **Optimization:** Regularly compact small files into larger ones to reduce I/O operations.

19. Challenges and solutions for maintaining data consistency in Azure Databricks

- **Challenges:**
  - Distributed nature of Spark jobs may cause skew or partial updates.
  - Concurrent writes or schema changes can lead to inconsistencies.
- **Solutions:**
  - Use Delta Lake's ACID properties for transactional consistency.
  - Implement optimistic concurrency control for concurrent writes.
  - Use checkpointing and idempotent pipelines to handle retries gracefully.

20. Implementing row-level security in Azure Synapse Analytics

- **Steps:**
  a. Define roles and policies in Synapse Analytics.
  b. Use SQL security predicates (CREATE SECURITY POLICY) to define access rules.
  c. Apply filters based on tenant IDs or user roles.
- **Management:** Integrate AAD for role-based access control (RBAC).

21. Architecting a hybrid data solution integrating on-premises and Azure

- **Ingestion:** Use ADF or Azure Data Gateway to connect on-premise databases to Azure services.
- **Processing:**
  - Process data in Azure Databricks for scalability.
  - Use ADF to orchestrate ETL workflows across hybrid environments.
- **Storage:** Synchronize data between on-premises and Azure Blob Storage or Synapse Analytics.
- **Connectivity:** Use ExpressRoute for secure, low-latency connections.

22. Migrating Hadoop/Spark workloads to Azure Databricks

- **Steps:**
  a. Export data from HDFS to Azure Data Lake.
  b. Convert Hive scripts and Spark jobs to Databricks notebooks.
  c. Test workloads in a staging Databricks cluster.
- **Minimizing Downtime:** Perform migration in phases, using ADF to copy incremental data during cutover.

23. Optimizing Spark jobs in Azure Databricks

- **Strategies:**
  - Use appropriate cluster sizing and auto-scaling for workloads.
  - Optimize joins with broadcast and repartitioning strategies.
  - Cache frequently accessed data to reduce re-computation.
- **Cost Management:** Use Spot Instances for transient clusters.

24. Orchestrating complex workflows across Azure services with ADF

  - Use a pipeline-centric approach to orchestrate data flows between Blob Storage, Databricks, and Synapse.
  - Implement activities like Web, Copy, and Notebook for service integration.
  - Enable dependency handling with conditional branching and retry policies.

25. Designing a lineage and auditing system in an Azure data platform

- **Lineage:** Use Azure Purview to track data flow and lineage across services.
- **Auditing:**
  - Log pipeline activities and errors in Log Analytics.
  - Use Databricks' delta.history() to trace changes in Delta tables.
- **Visualization:** Build dashboards in Power BI to monitor lineage and audit logs.

Nitya CloudTech

Dream.Achieve.Succeed