VISION BOARD

AZURE DATA ENGINEERING INTERVIEW QUESTIONS PART-1

DEVIKRISHNA R

LinkedIn: @Devikrishna R Email: visionboard044@gmail.com

1. Reading and Processing 10 Million CSV Files in ADLS Gen2 using Azure Data Factory

To handle a large number of CSV files in ADLS Gen2 using Azure Data Factory (ADF):

- **Step 1: Use Wildcard File Path**
  Use ADF's **Copy Data** or **Data Flow** activity to define a wildcard pattern to match the files in the ADLS Gen2 container. This avoids the need to list each file explicitly.
- **Step 2: Batch Processing**
  Enable **File Path Partitioning** to process files in parallel. Use the maxConcurrency setting in activities like Copy Data to increase throughput.
- **Step 3: Optimize Data Flow**
  Use **Mapping Data Flow** to transform and aggregate data as needed. Leverage **partitioning options** in Data Flow for parallel processing.
- **Step 4: Iterative Processing with ForEach Activity**
  If each file requires specific processing, use the **Get Metadata** activity to retrieve file names, and iterate over them with the **ForEach** activity.
- **Step 5: Performance and Cost Optimization**
  - Enable **compression** on CSVs to reduce size.
  - Use **polybase or external tables** if the destination is Azure Synapse Analytics.
  - Use **Integration Runtime (IR)** to scale the compute environment.

---

2. What is Integration Runtime?

Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory to provide the data integration capabilities. Types of IR:

- **Azure IR**: For cloud-based activities like copying data between cloud sources.
- **Self-Hosted IR**: For accessing on-premises or private network data securely.
- **Azure-SSIS IR**: For executing SQL Server Integration Services (SSIS) packages.

---

3. Variables and Parameters in ADF

- **Variables**:
  - Used to store temporary values during the pipeline run.
  - Scope: Pipeline-level.
  - Use: Modify values dynamically during execution using **Set Variable** activity.
- **Parameters**:
  - Used to pass values into a pipeline or data flow at runtime.
  - Scope: Read-only and defined at pipeline or dataset levels.
  - Use: Customize pipeline runs without modifying the pipeline.

---

4. Activities in ADF

Activities perform tasks in a pipeline. Common types:

- **Copy Data Activity**: Copies data between sources/destinations.
- **Data Flow Activity**: Enables data transformation at scale.
- **Lookup Activity**: Retrieves data from a source (used for condition-based workflows).
- **ForEach Activity**: Iterates through a collection of items.
- **Filter Activity**: Filters data based on conditions.
- **Set Variable/Append Variable Activity**: Updates variable values.
- **Execute Pipeline Activity**: Executes a child pipeline.
- **Web Activity**: Calls a REST endpoint.
- **Delete Activity**: Deletes files or datasets.
- **Wait Activity**: Adds a delay to the pipeline.

---

5. Automating Failure Notifications

To automate email notifications on pipeline failure:

- Add a **Failure dependency** in the pipeline using the **OnFailure** trigger.
- Use the **Web Activity** to call a Logic App or Azure Function that sends an email using **SendGrid** or **Office 365 SMTP**.
- Alternatively, use Azure Monitor Alerts to monitor pipeline status and send emails on failure.

---

6. Handling Exceptions in ADF

- Use **Error Handling Activities** like:
  - **Try-Catch blocks** with OnFailure dependencies.
  - **Execute Pipeline** for reattempt or fallback operations.
- **Custom Logging**: Log errors using **Azure Log Analytics** or store them in a database.
- **Retry Policies**: Configure retries on activities in the activity settings.

---

7. Fixing Slow ADF Pipeline

- **Analyze Bottlenecks**:
  - Use ADF **monitoring tools** to identify slow activities.
  - Optimize data movement by using proper partitioning.
- **Increase Parallelism**:
  - Increase maxConcurrency in activities.
  - Optimize partitioning in Data Flows.
- **Use High-Performance Resources**:
  - Use **Premium Integration Runtime** for intensive operations.
  - Upgrade the target database (e.g., Azure Synapse) if it is the bottleneck.
- **Optimize Source/Destination**:
  - Enable compression and indexing.

- Use incremental loading or delta processing for large datasets.

---

8. Blob Storage vs. ADLS Gen2

Feature	Blob Storage	ADLS Gen2
Hierarchy	Flat Namespace	Hierarchical Namespace
Performance	Suitable for general workloads	Optimized for analytics workloads
Security	Role-based access	POSIX-like ACLs, more granular
Integration	Good for general use	Better for Big Data and analytics

Why ADLS Gen2 is Required?

- Supports **big data analytics** scenarios like processing petabytes of data.
- Offers **better performance** due to hierarchical namespace.
- Allows **granular security controls** with ACLs.
- Optimized for integration with **Azure Synapse** and **Data Lake Analytics**.

9. Connecting ADLS Gen2 with Databricks

To connect ADLS Gen2 with Databricks:

1. **Use Azure Active Directory (AAD)**: Authenticate Databricks to access ADLS Gen2 via a Service Principal or Managed Identity.

2. **Steps**:
    - Assign **Storage Blob Data Contributor** or **Storage Blob Data Owner** role to the Service Principal or Databricks workspace managed identity in the Azure portal for the ADLS Gen2 storage account.
    - Configure access in Databricks by adding the credentials (e.g., OAuth token or client secret).

Role assignments are configured in **Azure Portal > Storage Account > Access Control (IAM)**.

---

10. Using Service Principal to Connect ADLS from Databricks

Steps to connect using Service Principal:

1. **Create a Service Principal**:
    - Register an app in Azure AD.
    - Generate a client secret.
2. **Assign Roles**:
    - Assign **Storage Blob Data Contributor** to the Service Principal for the ADLS Gen2 account.
3. **Configure Databricks**:
    - Add Service Principal details as secret scopes in Databricks.

Code Example:

```
# Set up configurations
configs = {
    "fs.azure.account.auth.type": "OAuth",
    "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs
        .azurebfs.oauth2.ClientCredsTokenProvider",
    "fs.azure.account.oauth2.client.id": "<client_id>",
    "fs.azure.account.oauth2.client.secret": "<client_secret>",
    "fs.azure.account.oauth2.client.endpoint": "https://login
        .microsoftonline.com/<tenant_id>/oauth2/token"
}

# Mount ADLS Gen2
dbutils.fs.mount(
    source="abfss://<container>@<storage_account_name>.dfs.core
        .windows.net/",
    mount_point="/mnt/<mount_name>",
    extra_configs=configs
)
```

---

11. Why Use Service Principal? How to Create It?

Why Use Service Principal?

- Provides a secure way to authenticate Databricks without using user credentials.
- Supports automation and application-level authentication.
- Enables fine-grained access control with role assignments.

Steps to Create Service Principal:

1. Go to **Azure Active Directory > App Registrations**.
2. Click **New Registration**, name the app, and register it.
3. Generate a **client secret** under **Certificates & Secrets**.

4. Assign the Service Principal a role under **Access Control (IAM)** for the target resource.

---

12. What is Databricks Runtime?

The **Databricks Runtime** is a pre-configured environment with optimized libraries for Apache Spark, Delta Lake, and other big data analytics tools.

Why We Need It:

- Offers optimized performance and scalability.
- Ensures compatibility with Spark APIs and machine learning libraries.
- Provides ready-to-use integrations with Azure services.

---

13. What are Workflows?

Workflows in Databricks allow you to create, schedule, and monitor pipelines of jobs.

- A workflow can chain multiple jobs together with dependencies.
- Supports retry policies and alerts for monitoring.
- Enables triggering with APIs or schedules.

---

14. Medallion Architecture

Medallion Architecture organizes data in three layers:

1. **Bronze Layer**: Raw, unprocessed data stored in a data lake.
2. **Silver Layer**: Cleaned and enriched data.

3. **Gold Layer**: Aggregated, analytics-ready data.

Benefits: Provides structured, incremental processing for large datasets.

---

15. Delta File Format

Delta Lake is an open-source storage format built on top of Parquet.

- **Features**:
  - ACID transactions.
  - Schema evolution.
  - Time travel (historical queries).
  - Scalable performance with optimized reads and writes.

---

16. Why Delta File Format in High Write Scenarios?

Delta format is essential for scenarios like Facebook's transactional logs:

- Handles **small file problems** by merging them into larger files during optimization.
- Provides **ACID guarantees**, ensuring consistency.
- Includes **data compaction** (optimize command), reducing the impact on performance.
- Enables efficient querying with **z-order clustering** and caching.

---

17. Debugging a Slow Job in Databricks

Steps to address slow jobs:

1. **Check Spark UI**: Identify slow stages or tasks.
2. **Skewed Data**: Use partitioning and bucketing to balance the load.
3. **Cluster Configuration**: Use autoscaling clusters or increase node size.
4. **Optimize Storage**: Use Delta Lake and compact files.
5. **Caching**: Cache intermediate results to avoid re-computation.

---

18. Optimization Techniques

1. **Storage Optimization**: Delta format, partitioning, and file compaction.
2. **Query Optimization**:
    - Use predicate pushdown.
    - Optimize joins with broadcast hints.
3. **Cluster Tuning**:
    - Adjust executor and driver memory.
    - Use autoscaling.
4. **Data Skew Handling**: Partition data by key.
5. **Pipeline Optimization**: Use caching, avoid shuffles, and tune Spark configurations.

---

19. Memory Management Optimization

1. **Memory Allocation**: Increase executor memory (spark.executor.memory) and configure off-heap memory if needed.
2. **Broadcast Variables**: Use for small lookup tables to reduce shuffle operations.
3. **Garbage Collection**: Tune JVM GC settings for optimal performance.
4. **Cache Management**:
    - Persist frequently accessed datasets using persist() or cache().
    - Release unused cached data.
5. **Shuffle Optimization**: Reduce shuffles by proper partitioning and using repartition() or coalesce().