Cloud Data Engineering Lab Solutions (Questions 1-10)

Shashi Kumar C

November 14, 2025

Q1: Generate and Test a SAS URL

Lab Task: Using Azure Storage Explorer, create a new container and upload a CSV file. Generate a SAS URL with read-only access valid for 30 minutes and test in a browser.

Conceptual Basis

A **Shared Access Signature (SAS)** is a URI that grants delegated, granular access to Azure Storage resources. It's superior to sharing an access key because it restricts permissions (e.g., read-only), scope (e.g., a single container), and time (e.g., 30 minutes).

Step-by-Step Solution

1. **Create Container:** In Azure Storage Explorer, navigate to your Storage Account > Blob Containers. Right-click and select "Create Blob Container". Name it `lab-data`.
2. **Upload File:** Select the `lab-data` container, click "Upload", and upload your CSV file.
3. **Generate SAS:** Right-click the `lab-data` container (not the file) and select "Get Shared Access Signature...".
4. **Configure SAS:**
    - **Permissions:** Uncheck all boxes, then check only **Read** and **List**.
    - **Expiry time:** Set the "End time" to be 30 minutes from the current time.
5. **Create and Copy:** Click "Create". Storage Explorer will generate the SAS. Copy the full **Query String** or **URI** to your clipboard.
6. **Test:** Paste the generated URI into a new browser tab. You should see a list of the files in the container (due to "List" permission). If you click your CSV file, the browser will display its contents (due to "Read" permission).

Q2: Connect with Access Key vs. SAS Token

Lab Task: Connect Azure Storage Explorer using an Access Key and again using a SAS Token. Upload a file in each mode and explain the security differences.

Conceptual Basis

- **Access Key:** This is the "root" password for your storage account. It grants **full administrative privileges** (read, write, delete, manage permissions) over *all* services in that account (Blobs, Files, Queues, Tables).

- **SAS Token:** This is a temporary, delegated "visitor pass". Its permissions are **strictly limited** to what was defined during its creation (e.g., read/write access to a single container for 1 hour).

Step-by-Step Solution

1. **Connect with Access Key:**
   - In Storage Explorer, click the "Connect" icon.
   - Select "Storage account or service".
   - Select "Account name and key".
   - Go to Azure Portal > Storage Account > Access keys. Copy the "Display name" (account name) and one of the keys.
   - Paste them into Explorer and connect.
   - **Result:** You have full admin rights. You can upload, delete, and even generate new SAS tokens.
2. **Connect with SAS Token (using the URI from Q1):**
   - Disconnect from the account, then click "Connect" again.
   - Select "Blob container".
   - Select "Shared access signature (SAS)".
   - Paste the SAS URI you generated in Q1.
   - **Result:** You can *only* see the `lab-data` container. If you try to upload a file, it will **fail** because the token only has "Read" permission. If you generated a SAS with "Write" permission, the upload would succeed for that container only.

Security Differences

The primary difference is **scope** and **control**. An access key is a high-privilege, non-expiring secret that gives total account control. A SAS token is a low-privilege, temporary credential that provides granular control over specific resources. **Best Practice:** Never use access keys in applications; always use SAS tokens.

Q3: Rotate Primary Access Key

Lab Task: Rotate the primary access key for a storage account and reconnect your client application. Explain the purpose of key regeneration.

Conceptual Basis

Key Rotation (or regeneration) is a critical security-hygiene practice. It invalidates an old access key and issues a new one. This is done to mitigate the risk of a key being compromised (e.g., leaked in code, stolen by an attacker). By periodically rotating keys, you limit the window of opportunity for an attacker to use a stolen key.

Step-by-Step Solution

1. **Verify Connection:** Ensure Storage Explorer is connected using the primary key (key1).
2. **Rotate Key:** In the Azure Portal, navigate to your Storage Account > **Access keys**.
3. You will see two keys: "key1" and "key2". This redundancy is specifically for rotation.
4. Click the "Rotate" (regenerate) icon next to **key1**. Confirm the warning.

5. **Test Old Connection:** Go back to Storage Explorer and try to refresh the container list. The connection will **fail** and you will get an authentication error. The old key is now invalid.
6. **Reconnect Client:**
   - In the portal, copy the *new* value for **key1**.
   - In Storage Explorer, find your account, right-click, and select "Properties".
   - Update the "Account Key" field with the new key and save.
   - Alternatively, detach the account and re-add it using the new key.
7. **Verification:** The connection will now succeed.

Q4: Enable Hierarchical Namespace (ADLS Gen2)

Lab Task: Enable hierarchical namespace on an existing storage account to upgrade it to Data Lake Gen2. Upload a nested folder structure and verify it via Azure CLI.

Conceptual Basis

Hierarchical Namespace (HNS) is the feature that distinguishes ADLS Gen2 from standard Blob Storage. It enables true directory-level operations (rename, move, ACLs) which is critical for big data analytics.

Critical Correction: The lab premise is flawed. You **cannot** enable HNS on an *existing* storage account. HNS must be enabled **at the time of creation**. This is a one-way decision. This solution answers the corrected task of creating a new ADLS Gen2 account.

Step-by-Step Solution

1. **Create Account:** In the Azure Portal, create a new "Storage Account".
   - **Basics:** Use a "Standard" performance tier and "StorageV2" account kind.
   - **Advanced Tab:** This is the key step. Check the box for **"Enable hierarchical namespace"**.
2. **Create Container:** Once created, go to the account > "Containers" and create a container (e.g., `datalake`).
3. **Upload (via CLI):** Open Azure Cloud Shell or your local CLI.
4. **Create Nested Structure:**

```
# Create directories locally
mkdir -p my-nested-data/raw/invoices
echo "data1" > my-nested-data/raw/invoices/inv-01.csv
echo "data2" > my-nested-data/raw/invoices/inv-02.csv

# Upload the entire structure (using --account-name)
# Note: 'az storage blob upload-batch' works for ADLS Gen2
az storage blob upload-batch -s "my-nested-data" \
  -d "datalake" --account-name <YourStorageAccountName>
```

5. **Verify (via CLI):** Use the ADLS-specific ‘az storage fs‘ commands.

```
# List the directories in the 'datalake' file system
az storage fs directory list -f "datalake" \
  --account-name <YourStorageAccountName>

# List the files in the nested directory
```

```
az storage fs file list -f "datalake" -p "raw/invoices" \
  --account-name <YourStorageAccountName>
```

Q5: ADF Pipeline - Copy Multiple CSVs (Wildcard)

Lab Task: Create an ADF pipeline to copy multiple CSV files from Blob to a consolidated file using wildcards and column mapping.

Conceptual Basis

This is a common ETL pattern. ADF's **Copy data** activity can use **wildcard paths** to read multiple files at once. The **Sink** configuration then allows these files to be **merged** into a single output file.

Step-by-Step Solution

1. **Setup:** Ensure you have a Blob container (`source-files`) with multiple CSVs (e.g., `sales-01.csv`, `sales-02.csv`) and an empty container (`output-files`).
2. **Create Datasets:**
   - **Source_Dataset (DelimitedText):** Point to the `source-files` container. *Do not* specify a file name.
   - **Sink_Dataset (DelimitedText):** Point to the `output-files` container. Specify the output file name, e.g., `consolidated.csv`.
3. **Create Pipeline:** Create a new ADF Pipeline and drag a **Copy data** activity.
4. **Configure Source:**
   - **Dataset:** Select `Source_Dataset`.
   - **File path type:** Select **Wildcard file path**.
   - **Wildcard path:** Set to `source-files`.
   - **Wildcard file name:** Set to `*.csv`.
5. **Configure Sink:**
   - **Dataset:** Select `Sink_Dataset`.
   - **Copy behavior:** Select **Merge files**.
6. **Configure Mapping:** Go to the "Mapping" tab. Click "Import schemas". ADF will inspect the files and map columns by name. You can adjust or remove mappings here if needed.
7. **Run:** Debug the pipeline. It will read all `*.csv` files from the source, merge them, and write a single `consolidated.csv` to the sink.

Q6: ADF Pipeline - Parameterized SQL Copy

Lab Task: Use a parameterized ADF pipeline to copy data from one SQL table to another dynamically by passing table names as parameters.

Conceptual Basis

Parameterization is a core ADF principle for building reusable, dynamic pipelines. We pass parameters from the *Pipeline* to the *Dataset*. The Dataset then uses these parameters to dynamically construct its properties (like the table name).

Step-by-Step Solution

1. **Create Pipeline Parameters:** In your ADF pipeline (outside the activity), go to the "Parameters" tab.
   - Create `SourceTableName` (String).
   - Create `SinkTableName` (String).
2. **Create Generic SQL Dataset:**
   - Create *one* new Dataset (Azure SQL Database). Name it `Generic_SQL_Table`.
   - Go to its "Parameters" tab and create a parameter named `DSetTableName`.
   - Go to its "Connection" tab. **Do not** select a table. Instead, check "Edit" and type `@dataset().DSetTableName` into the "Table" text box.
3. **Configure Copy Activity:**
   - **Source:** Select the `Generic_SQL_Table` dataset. You will see its parameter `DSetTableName`.
   - Set its value to the *pipeline* parameter: `@pipeline().SourceTableName`.
   - **Sink:** Select the *same* `Generic_SQL_Table` dataset.
   - Set its `DSetTableName` value to the *pipeline* parameter: `@pipeline().SinkTableName`.
   - In the Sink settings, set "Table option" to "Auto create table" if the sink table doesn't exist.
4. **Run:** When you Debug the pipeline, ADF will prompt you to enter values for `SourceTableName` and `SinkTableName`.

Q7: Data Flow - Filter Transformation

Lab Task: Use Filter transformation to extract transactions where Amount > 10000. Export the output to Blob storage.

Conceptual Basis

A **Filter** transformation in a Mapping Data Flow evaluates a condition for every row. If the condition is *true*, the row passes through. If *false*, the row is dropped.

Step-by-Step Solution

1. **Create Data Flow:** Create a new **Mapping Data Flow**.
2. **Add Source:** Add a `Source` transformation pointing to your transactions data (e.g., a source CSV in Blob).
3. **Add Filter:** Click the `+` icon next to the source and select the **Filter** transformation.
4. **Configure Filter:**
   - **Filter on:** Click the "Expression builder" box.
   - **Expression:** Find your amount column (e.g., `Amount`) and write the condition: `Amount > 10000`.
   - Note: You may need to cast the column if it's read as a string: `toInteger(Amount) > 10000`.
5. **Add Sink:** Click the `+` icon next to the filter and add a **Sink** transformation. Configure it to write to a new CSV file in Blob storage.
6. **Create Pipeline:** Create a new ADF Pipeline and drag your **Data Flow** activity onto it.
7. **Run:** Debug the pipeline. Check the output file; it should only contain rows where the amount exceeds 10000.

Q8: Data Flow - Derived Column Transformation

Lab Task: Use Derived Column transformation to create a calculated field `FinalPrice = Price - Discount`. Apply string functions to format text columns.

Conceptual Basis

A **Derived Column** transformation is a data-flow "workbench". It doesn't add or remove rows, but it allows you to *modify* existing columns or *add* new columns based on expressions, calculations, and functions.

Step-by-Step Solution

1. **Create Data Flow:** In a Mapping Data Flow, add your `Source`.
2. **Add Derived Column:** Click the `+` icon and select **Derived Column**.
3. **Configure Columns:** In the "Derived column's settings" pane:
   - **Calculated Field:**
   - **Column:** Select "Add new column" and name it `FinalPrice`.
   - **Expression:** `Price - Discount`. (Use `toFloat(Price) - toFloat(Discount)` if type-casting is needed).
   - **String Function:**
   - **Column:** Select an existing text column (e.g., `ProductName`).
   - **Expression:** `upper(trim(ProductName))`. This will remove leading/trailing whitespace and convert the name to uppercase.
4. **Add Sink:** Add a `Sink` and run the Data Flow from a pipeline to see the results.

Q9: Data Flow - Aggregate Transformation

Lab Task: Implement Aggregate transformation to find average sales per region and total number of transactions.

Conceptual Basis

An **Aggregate** transformation performs calculations across groups of data, similar to a SQL `GROUP BY` clause. It has two parts: the *grouping key* (what defines the group) and the *aggregate functions* (the calculations to perform on each group).

Step-by-Step Solution

1. **Create Data Flow:** In a Mapping Data Flow, add your `Source` (e.g., sales data).
2. **Add Aggregate:** Click the `+` icon and select **Aggregate**.
3. **Configure Group By:**
   - In the "Group by" tab, select the column that defines your groups, e.g., `Region`.
4. **Configure Aggregates:**
   - Switch to the "Aggregates" tab.
   - **Column 1:** Name it `AverageSales`.

- **Expression 1:** `avg(SalesAmount)` (or `avg(toInteger(SalesAmount))`).
- **Column 2:** Add another aggregate. Name it `TotalTransactions`.
- **Expression 2:** `count(TransactionID)` (or simply `count(1)` to count rows).

5. **Add Sink:** Add a `Sink`. The output will no longer be row-level data; it will be a summary table with columns: `Region`, `AverageSales`, `TotalTransactions`.

Q10: Data Flow - Inner Join Transformation

Lab Task: Perform an Inner Join between Employee and Department tables on `DeptID` and display matching results.

Conceptual Basis

A **Join** transformation combines data from two different input streams (sources) based on a matching condition. An **Inner Join** specifically outputs *only* the rows that have a matching key in *both* streams.

Step-by-Step Solution

1. **Create Data Flow:** In a Mapping Data Flow, add two **Source** transformations.
   - `Source_Employee`: Pointing to the Employee table.
   - `Source_Department`: Pointing to the Department table.
2. **Add Join:** Click the `+` icon after `Source_Employee` and select **Join**.
3. **Configure Join:**
   - **Right stream:** Select `Source_Department`.
   - **Join type:** Select **Inner**.
   - **Join conditions:**
   - **Left column:** `DeptID` (from Employee stream).
   - **Right column:** `DeptID` (from Department stream).
4. **Add Sink:** Add a `Sink`. The output will be a "wide" table containing columns from both Employee and Department, but only for employees who belong to a valid, existing department.