**Azure Data Factory Pipeline with CSV file validation**

### **Overview**

This guide will walk you through creating a robust pipeline in Azure Data Factory (ADF) to validate and load large CSV files into a SQL Server database. We will follow the architecture laid out in the document, which is designed for performance, error handling, and scalability.

### **Phase 1: Initial Azure Setup (Prerequisites)**

Before you start building in Azure Data Factory, you need a few things in place. Think of this as preparing your workshop before starting a project.

1. **Azure Subscription**: You need an active Azure subscription.
2. **Resource Group**: Create a resource group to hold all your related resources (like the Data Factory, storage, and database). This keeps everything organized.
3. **Azure Blob Storage**: You'll need a storage account to hold your source CSV files and for staging data. The document recommends using **Premium Storage** with a **Hierarchical Namespace** enabled for better performance.
4. **Azure SQL Database or SQL Server**: This is where your validated data will be loaded. You'll need the server name, database name, username, and password.

### **Phase 2: Create Your Data Factory and Connections**

Now, let's create the Data Factory itself and connect it to your storage and database.

#### **Step 1: Create the Azure Data Factory**

1. In the [Azure Portal](https://portal.azure.com/), click **+ Create a resource**.
2. Search for Data Factory and click **Create**.
3. Select your subscription and resource group.
4. Give your Data Factory a unique name (e.g., My-CSV-Validation-ADF).
5. Choose a region.
6. For the version, select **V2**.
7. Click **Review + create**, and then **Create**.

#### **Step 2: Launch Azure Data Factory Studio**

1. Once your Data Factory is deployed, navigate to it in the Azure Portal.
2. Click on the **Launch Studio** button. This will open the Azure Data Factory visual interface in a new tab.

#### **Step 3: Create Linked Services (Connections)**

Linked Services are the connection strings that ADF uses to connect to external resources. We'll create two, as described in the "Implementation Checklist".

1. In the ADF Studio, go to the **Manage** tab (wrench icon on the left).
2. Click on **Linked Services**, then **+ New**.

**A. Create the Azure Blob Storage Linked Service:**

1. Search for Azure Blob Storage and select it.
2. Give it a name, like AzureBlobStorage\_Source.
3. Select your Azure subscription and the storage account you created earlier.
4. Click **Test connection** to ensure it works, then click **Create**.

**B. Create the Azure SQL Database Linked Service:**

1. Click **+ New** again.
2. Search for Azure SQL Database and select it.
3. Give it a name, like AzureSqlDatabase\_Target.
4. Enter your SQL Server and database details (server name, database name, username, and password). The document recommends using **Azure Key Vault** to store these secrets securely.
5. **Test connection**, and then **Create**.

### **Phase 3: Set Up a SQL Table for Logging**

The pipeline needs to log errors. Let's create the table for that.

1. Connect to your Azure SQL Database using a tool like Azure Data Studio or SQL Server Management Studio (SSMS).
2. Open a new query window and execute the following SQL script from the document to create the validation\_log table:

SQL

CREATE TABLE validation\_log (

log\_id INT IDENTITY(1,1) PRIMARY KEY,

file\_name VARCHAR(255),

validation\_step VARCHAR(100),

error\_message TEXT,

error\_count INT,

processing\_date DATETIME,

pipeline\_run\_id VARCHAR(100)

);

You will also need a table to log file processing history. Run this SQL as well:

SQL

-- This schema is inferred from the document's SQL queries

CREATE TABLE file\_processing\_log (

log\_id INT IDENTITY(1,1) PRIMARY KEY,

file\_name VARCHAR(255),

table\_name VARCHAR(255),

row\_count BIGINT,

processing\_date DATETIME,

status VARCHAR(50),

pipeline\_run\_id VARCHAR(100)

);

-- Create an index for faster lookups as recommended

CREATE INDEX IX\_processing\_date ON file\_processing\_log (processing\_date);

### **Phase 4: Build the Main Pipeline**

Now for the main event: building the pipeline.

#### **Step 1: Create a New Pipeline**

1. In the ADF Studio, go to the **Author** tab (pencil icon).
2. Hover over **Pipelines** and click the three dots (...), then select **New pipeline**.
3. Give your pipeline a descriptive name, like PL\_CSV\_Validation\_and\_Load.

#### **Step 2: Add Pipeline Parameters**

Parameters make your pipeline reusable. We'll add the ones from the document.

1. With your new pipeline open, click on the **Parameters** tab at the bottom.
2. Click **+ New** and add the following parameters from the Pipeline Parameters section of the document:
   * **Name**: sourceFilePath, **Type**: String
   * **Name**: targetTableName, **Type**: String
   * **Name**: expectedSchema, **Type**: String
   * **Name**: validationMode, **Type**: String (Default value: strict)
   * **Name**: chunkSize, **Type**: Int (Default value: 100000)

#### **Step 3: Add Pipeline Variables**

Variables are used to store temporary values within a pipeline run.

1. Next to the "Parameters" tab, click the **Variables** tab.
2. Click **+ New** to add the variables from the document:
   * **Name**: currentRowCount, **Type**: Integer, **Default value**: 0
   * **Name**: previousRowCount, **Type**: 'Integer', **Default value**: 0
   * **Name**: validationErrors, **Type**: Array
   * **Name**: fileSize, **Type**: Integer, **Default value**: 0

#### **Step 4: Implement the Large File Processing Strategy**

The document outlines a strategy where large files are split into smaller chunks and processed in parallel. This is done using a ForEach activity.

1. **File Splitting (Conceptual)**: The document shows a Python script for splitting files. In a real-world ADF pipeline, you would implement this logic using an **Azure Function** or **Azure Batch** activity. For a beginner, a simpler approach is to assume the files are already chunked or to use a **Data Flow** which can handle large files internally. For this guide, we will focus on the validation steps that can be performed without custom code.
2. **Add a ForEach Activity for Parallel Processing**:
   * In the **Activities** toolbox, search for ForEach and drag it onto the pipeline canvas.
   * Go to the **Settings** tab for the ForEach activity.
   * The document specifies using @activity('SplitFile').output.chunk\_files for the **Items** field. This assumes a previous activity splits the file. If you are processing a folder of files, you could use a **Get Metadata** activity to get a list of files and pass that to the ForEach activity.
   * Check the **Is sequential** box to *uncheck* it for parallel processing.
   * Set the **Batch count** to 8 as suggested for parallelism.

#### **Step 5: Add Validation Activities Inside the ForEach**

Now, click on the pencil icon on the ForEach activity to edit the activities that will run on each file/chunk.

**A. Data Flow for Schema and Quality Validation:**

The document suggests using a **Data Flow** for distributed validation. This is a powerful way to perform complex validations on large datasets without writing code.

1. Inside the ForEach activity, drag a **Data Flow** activity onto the canvas.
2. Go to the **Settings** tab. Click **+ New** to create a new Data Flow.
3. Give the Data Flow a name, like DF\_Validate\_CSV\_Chunk.

**Inside the Data Flow:**

1. **Add Source**:
   * Click **Add Source**.
   * For the source dataset, create a new one: select Azure Blob Storage, then DelimitedText (for CSV).
   * Point this dataset to your source file's location. Use a parameter for the filename to make it dynamic.
2. **Add Assert Transformation for Validation**:
   * Click the + icon next to your source, and select **Assert** under "Schema modifiers".
   * This is where you'll implement the schema and data quality checks.
   * Go to the **Settings** tab for the Assert transformation.
   * Set the **Assert type** to whatever is appropriate (e.g., Assert true).
   * In **Assert conditions**, you can add the rules mentioned in the document, like !isNull(key\_column) or data type checks. You'll need to use the [ADF expression language](https://learn.microsoft.com/en-us/azure/data-factory/control-flow-expression-language-functions) here.

**B. Row Count Validation (using a Lookup activity):**

1. Go back to the main pipeline canvas. After the ForEach loop, you can compare row counts.
2. Drag a **Lookup** activity onto the canvas.
3. **Settings**:
   * Configure it to run a SQL query against your database to get the historical row count.
   * Use the SQL query provided in the document under Historical Comparison with Indexing. Remember to replace placeholders with ADF dynamic content expressions (e.g., '@{pipeline().parameters.targetTableName}').

### **Phase 5: Implement Data Loading**

After validation, the pipeline loads the data.

#### **Step 1: Create Staging and Target Tables**

1. Connect to your SQL database again.
2. Run the SQL scripts from the document to create the staging\_table and your final target\_table. The document also suggests creating a **Clustered Columnstore Index** on the staging table for performance.

#### **Step 2: Add the Copy Data Activity**

1. On your pipeline canvas (likely inside the ForEach or after it, depending on your strategy), drag a **Copy Data** activity.
2. **Source**: Set the source to your CSV dataset.
3. **Sink**: Set the sink to a new dataset pointing to your SQL staging\_table.
4. **Settings**:
   * The document suggests enabling staging for high performance (enableStaging: true). This uses Blob Storage as a temporary stage, which is much faster for bulk loads.
   * Set **Write behavior** to Insert.

#### **Step 3: Add a Stored Procedure Activity for Final Load**

1. First, create the stored procedure in your SQL database using the sp\_transform\_and\_load script from the document. This procedure will handle the logic of moving data from staging to the final target table (often called an "upsert").
2. In your ADF pipeline, after the **Copy Data** activity, drag a **Stored Procedure** activity onto the canvas.
3. **Settings**:
   * Select the **Linked Service** for your SQL database.
   * Select the sp\_transform\_and\_load stored procedure you just created.

### **Phase 6: Finalizing the Pipeline**

#### **Step 1: Configure Triggers**

Triggers start your pipeline automatically.

1. In the ADF Studio, go to the **Manage** tab.
2. Click on **Triggers**, then **+ New**.
3. You can choose a:
   * **Schedule Trigger**: To run the pipeline at a specific time (e.g., daily).
   * **Storage Event Trigger**: To run the pipeline whenever a new file is uploaded to your storage account.

#### **Step 2: Monitoring and Alerting**

1. To monitor pipeline runs, go to the **Monitor** tab (the speedometer icon). You can see successful and failed runs, check their duration, and troubleshoot errors.
2. To set up alerts (e.g., for failures), click on **Alerts & Metrics** and then **New alert rule**. You can configure it to send an email if a pipeline run fails.

### **Implementation Checklist Summary**

To ensure you've covered the key points from the document:

* [ ] **Create Linked Services**: You've connected to your Storage Account and SQL Server.
* [ ] **Create Datasets**: You've defined your CSV source and SQL sinks.
* [ ] **Create Pipeline with Activities**: You've built the flow with validation and loading steps.
* [ ] **Set Up Error Handling**: The SQL logging table is ready to catch errors.
* [ ] **Configure Monitoring**: You know where to monitor runs and set up alerts.
* [ ] **Test**: Run your pipeline with sample data to ensure it works end-to-end.
* [ ] **Deploy**: Once tested, your pipeline is live.

This step-by-step guide provides a simplified path to implementing the complex pipeline from your document. As you get more comfortable with the Azure UI, you can explore the more advanced features mentioned, such as custom activities with Python, data flow partitioning, and auto-scaling integration runtimes.