**Azure Data Lake SDK and Python: Automating Data Ingestion for Seamless Cloud Integration**

As data continues to grow, businesses need reliable ways to store and manage it. Azure Data Lake SDK and Python make it easy to automate moving data from repositories to the cloud. In this article, you'll learn how to use the SDK to create file systems, set up directories, and upload data to Azure Data Lake Storage. By using environment variables and SAS tokens, you can securely and efficiently build a scalable data pipeline that fits seamlessly with your existing setup.

**Step 1: Setting Up an Azure Storage Account**

Before we dive into automating data ingestion, the first step is to create an Azure Storage Account that supports Data Lake functionalities.

**Create a New Storage Account**

1. Navigate to the **Azure Portal** and select **Storage Accounts**.
2. Click **Create** to start the process.
3. Fill in the required details like Subscription, Resource Group, and Storage Account Name. Make sure to select **StorageV2 (general-purpose v2)** for the **Account kind**.
4. **Enable Hierarchical Namespace** under the **Data Lake Storage** option. This is crucial for Data Lake functionality.

Once the account is created, the **Hierarchical Namespace** should be enabled as shown here:

A screenshot of a computer

Description automatically generated

**Step 2: Generating a Shared Access Signature (SAS) Token**

Now that the storage account is set up, you’ll need a SAS token to authenticate the actions performed by the Python script without exposing your primary access key.

Steps to Generate SAS Token:

1. Navigate to your storage account in the Azure Portal.
2. On the left-hand menu, select **Shared Access Signature**.
3. Configure the permissions (you’ll need at least Read, Write, List, and Create permissions).
4. Set the expiry date and allowed protocols, then click **Generate SAS and connection string.**

Once generated, copy the **SAS token** and keep it secure. This will be used in your Python script to authenticate your operations.

A screenshot of a computer

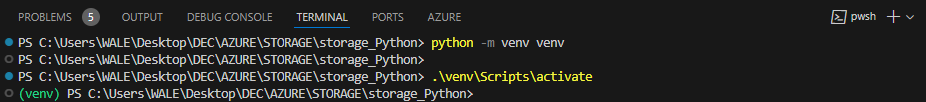
Description automatically generated

**Step 3: Setting Up the Development Environment**

To ensure smooth integration between Python and Azure Data Lake, set up a virtual environment and install the necessary libraries.

**1. Create and Activate a Virtual Environment**

Run the following commands in your terminal:



**2. Install Required Python Packages**

Once the virtual environment is activated, install the Azure Data Lake SDK, Identity libraries, and dotenv to handle environment variables.

A screenshot of a computer program

Description automatically generated

**Step 4: Storing Sensitive Information in Environment Variables**

Instead of hardcoding sensitive information like your SAS token and storage account name, use a **.env file** to store these values securely.

**.env File Configuration:**

Create a .env file in your project root folder

A screen shot of a computer program

Description automatically generated

Add below into the file:

A red line on a black background

Description automatically generated

**Step 5: Writing the Python Script**

**1. Importing Required Libraries**

Your script needs several Azure SDK components to connect and interact with Azure Data Lake:

A screen shot of a computer

Description automatically generated

**2. Loading Environment Variables**

Next, load the sensitive information stored in the .env file:

A screen shot of a computer

Description automatically generated

**3. Connecting to Azure Data Lake Storage**

Create a service client using the **DataLakeServiceClient** class to connect to the Azure Data Lake Storage account:

A screen shot of a computer program

Description automatically generated

**Step 6: Creating a Container (File System)**

A **container** (also referred to as a **file system**) is where you store directories and files. This code creates a new container or retrieves the existing one.

A screen shot of a computer code

Description automatically generated

After running this code, the container appears in the Azure portal:

A screenshot of a computer

Description automatically generated

**Step 7: Creating a Directory**

Within your container, you'll need to create directories to organize your files. Here’s how to create a directory within a container:

A screen shot of a computer code

Description automatically generated

Here’s the directory created in the Azure portal:

A screenshot of a computer

Description automatically generated

**Step 8: Uploading the Data File**

Now that both the container and directory are set up, we can upload a file from your local system to the Azure Data Lake.

A screen shot of a computer program

Description automatically generated

In this example, the file Data.csv is uploaded into the source directory in the oyass container. After running this code, the file appears in the Azure portal:

A screenshot of a computer

Description automatically generated

**Step 9: Running the Script**

With everything in place, you can run the Python script from your terminal using

python Azure\_file\_upload.py; Azure\_file\_upload.py being my python file name

Upon execution, you should see an output like this in your terminal, confirming successful operations:

A black screen with white text

Description automatically generated

**Conclusion**

By combining **Azure Data Lake SDK** and **Python**, you can efficiently manage data ingestion processes, making your pipelines scalable, secure, and easy to maintain. This project demonstrates how to automate the creation of containers, directories, and file uploads using environment variables and SAS tokens for secure access.

Full Code:

A computer screen shot of text

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated