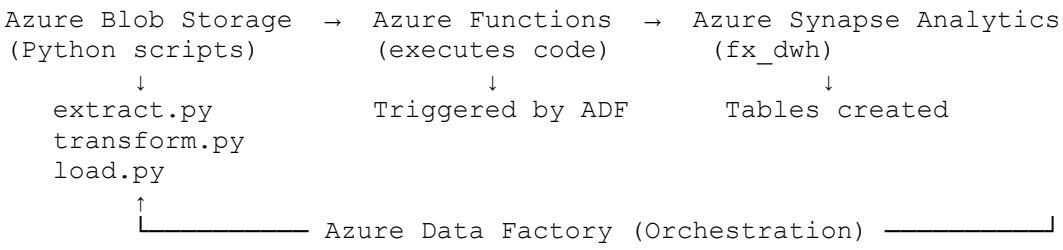
Implementation Steps - Graphical Orchestration with Azure Data Factory

1. Azure Resources

- ☑ Azure Subscription (free account acceptable)
- ☑ Resource Group
- ☑ Azure Data Factory configured (ADF)
- ☑ Azure Synapse Analytics (DWH)
- ☑ Azure Blob Storage (to store Python scripts and temporary CSV files)
- ☑ Integration Runtime (Self-Hosted if MySQL is local, otherwise Azure IR)

📐 Architecture



📋 Prerequisites

☑ Azure Synapse Analytics with tables created
☑ Azure Data Factory configured and linked with ABS, AFA, ASA
☑ Azure Blob Storage with Python scripts

🎯 Solution: Azure Functions + Azure Data Factory

Why this solution?

- ☑ No need for VM or Azure Batch
- ☑ 100% graphical configuration
- ☑ Cost-effective (pay-per-execution)
- ☑ Easy to orchestrate with Data Factory

📝 STEP 1: Prepare Azure Blob Storage

1.1 Organize files

In your **Azure Storage Account**, create this structure:

```
Container: fx-pipeline-scripts
│
├── config/
│   └── config.py
│
├── scripts/
│   ├── extract.py
│   ├── transform.py
│   └── load.py
│
├── requirements.txt
│
└── temp/ (empty folder for temporary CSV files)
```

1.2 Create requirements.txt file

In Blob Storage, add a requirements.txt file:

```
requests==2.31.0
pandas==2.1.4
pymysql==1.1.0
sqlalchemy==2.0.23
python-dotenv==1.0.0
azure-storage-blob==12.19.0
```

🔧 STEP 2: Modify Python Scripts for Azure

2.1 Modify config.py for Azure

Replace the content of config/config.py with Azure-compatible configuration.

2.2 Update requirements.txt

Replace content with:

```
requests==2.31.0
pandas==2.1.4
pymssql==2.2.11
sqlalchemy==2.0.23
azure-storage-blob==12.19.0
azure-functions==1.18.0
```

⚡ STEP 3: Create Azure Functions (Graphical Interface)

3.1 Create a Function App

1. **Go to Azure Portal** → https://portal.azure.com
2. Click **"Create a resource"**
3. Search for **"Function App"** → Click **"Create"**

Configuration:

- **Subscription**: Your subscription
- **Resource Group**: Your project resource group (or create new)
- **Function App name**: fx-pipeline-functions (globally unique)
- **Runtime stack**: **Python**
- **Version**: **3.11** (or 3.10)
- **Region**: Same region as your SQL Database
- **Operating System**: **Linux**
- **Plan type**: **Consumption (Serverless)**

4. Click **"Review + Create"** → **"Create"**

⏳ Wait 2-3 minutes for the Function App to be created.

3.2 Configure Application Settings

1. Once created, go to your **Function App** → fx-pipeline-functions
2. In the left menu → **Settings** → **Configuration**
3. Click **"+ New application setting"** to add each variable:

Name	Value	Description
DB_HOST	your-server.database.windows.net	Your Azure SQL Server
DB_USER	your_username	SQL Username
DB_PASSWORD	your_password	SQL Password
DB_NAME	fx_dwh	Database name
DB_PORT	1433	Azure SQL Port

Name	Value	Description
START_DATE	2024-01-01	Start date
AZURE_STORAGE_CONNECTION_STRING	DefaultEndpointsProtocol=https;AccountName=...	Storage connection string
BLOB_CONTAINER_NAME	fx-pipeline-scripts	Container name

To get the Storage Connection String:

- Go to your **Storage Account → Access keys**
- Copy **"Connection string"** under **key1**

4. Click **"Save"** at the top
5. Restart the Function App

3.3 Create the 3 Functions (HTTP Trigger)

Function 1: ExtractFX

1. In your Function App → **Functions → + Create**
2. **Development environment**: **Develop in portal**
3. **Template**: **HTTP trigger**
4. **New Function**: ExtractFX
5. **Authorization level**: **Function**
6. Click **Create**

Python Code (**init**.py):

```
import logging
import azure.functions as func
import sys
import os

# Import your extract script
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('ExtractFX function started')

    try:
        # Import and execute extract
        from scripts import extract

        start_date = req.params.get('start_date', '2024-01-01')
        end_date = req.params.get('end_date')
```

```
        result = extract.main(start_date=start_date, end_date=end_date)

        if result == 0:
            return func.HttpResponse(
                "☑ Extraction successful",
                status_code=200
            )
        else:
            return func.HttpResponse(
                "❌ Extraction failed",
                status_code=500
            )

    except Exception as e:
        logging.error(f"Error: {e}")
        return func.HttpResponse(
            f"❌ Error: {str(e)}",
            status_code=500
        )
```

Function 2: TransformFX

Repeat the same steps:

- **New Function**: TransformFX
- **Code**:

```
import logging
import azure.functions as func
import sys
import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('TransformFX function started')

    try:
        from scripts import transform

        result = transform.main()

        if result == 0:
            return func.HttpResponse(
                "☑ Transformation successful",
                status_code=200
            )
        else:
            return func.HttpResponse(
                "❌ Transformation failed",
                status_code=500
            )
```

```
    except Exception as e:
        logging.error(f"Error: {e}")
        return func.HttpResponse(
            f"❌ Error: {str(e)}",
            status_code=500
        )
```

Function 3: LoadFX

- **New Function**: LoadFX
- **Code**:

```
import logging
import azure.functions as func
import sys
import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('LoadFX function started')

    try:
        from scripts import load

        result = load.main()

        if result == 0:
            return func.HttpResponse(
                "☑ Loading successful",
                status_code=200
            )
        else:
            return func.HttpResponse(
                "❌ Loading failed",
                status_code=500
            )

    except Exception as e:
        logging.error(f"Error: {e}")
        return func.HttpResponse(
            f"❌ Error: {str(e)}",
            status_code=500
        )
```

🔄 STEP 4: Create the Pipeline in Azure Data Factory (Graphical)

4.1 Open Azure Data Factory Studio

1. Go to your **Azure Data Factory**
2. Click **"Launch Studio"**

4.2 Create Linked Services

Linked Service 1: Azure Functions

1. In ADF Studio → **Manage** (briefcase icon) → **Linked Services**
2. Click **"+ New"**
3. Search for **"Azure Function"** → Select → **Continue**
4. **Configuration**:
    - **Name**: AzureFunctionLinkedService
    - **Function App URL**: https://fx-pipeline-functions.azurewebsites.net
    - **Authentication type**: **Function key**
    - **Function key**:
        - Go to your Function App → **Functions** → **App keys**
        - Copy the **default** key
        - Paste in ADF
5. **Test connection** → Click **Create**

Linked Service 2: Azure SQL Database (if not already done)

1. **+ New** → Search for **"Azure SQL Database"** → **Continue**
2. **Configuration**:
    - **Name**: AzureSQLDatabase
    - **Server name**: your-server.database.windows.net
    - **Database name**: fx_dwh
    - **Authentication**: **SQL authentication**
    - **User name** / **Password**
3. **Test connection** → **Create**

4.3 Create the FX Pipeline

1. In ADF Studio → **Author** (pencil icon) → **Pipelines** → **+ (New Pipeline)**
2. **Name**: FX_ETL_Pipeline

Activity 1: ExtractFX

1. In the **Activities** toolbox → **Azure Function** → Drag onto canvas
2. **General tab**:
    - **Name**: Extract_FX_Data
3. **Azure Function tab**:
    - **Azure Function linked service**: AzureFunctionLinkedService
    - **Function name**: ExtractFX
    - **Method**: POST
    - **Body**: `{"start_date": "@{pipeline().parameters.StartDate}"}`

Activity 2: TransformFX

1. Drag a new **Azure Function** activity
2. **General**:
   - **Name**: Transform_FX_Data
3. **Azure Function**:
   - **Linked service**: AzureFunctionLinkedService
   - **Function name**: TransformFX
   - **Method**: POST
4. **Connect Extract → Transform**:
   - Click the **green arrow** from Extract_FX_Data
   - Drag to Transform_FX_Data
   - This creates a dependency: Transform executes **only if** Extract succeeds

Activity 3: LoadFX

1. Drag a 3rd **Azure Function** activity
2. **General**:
   - **Name**: Load_To_SQL
3. **Azure Function**:
   - **Linked service**: AzureFunctionLinkedService
   - **Function name**: LoadFX
   - **Method**: POST
4. **Connect Transform → Load**

Visual Result:

4.4 Add a Pipeline Parameter

1. Click on the **white background** of the pipeline (not on an activity)
2. Bottom → **Parameters tab**
3. **+ New**:
   - **Name**: StartDate
   - **Type**: String
   - **Default value**: 2024-01-01

4.5 Publish the Pipeline

1. Top → Click **"Publish all"**
2. Click **"Publish"** to confirm

⏰ STEP 5: Create an Automatic Trigger (Graphical Interface)

5.1 Create a Schedule Trigger

1. In your pipeline FX_ETL_Pipeline
2. Top → **Add trigger** → **New/Edit**
3. **Choose trigger** → **+ New**
4. **Configuration**:
    - **Name**: DailyFXTrigger
    - **Type**: **Schedule**
    - **Start date**: Today at 17:00
    - **Time zone**: **Europe/Paris** (or your timezone)
    - **Recurrence**: **Every 1 Day**
    - **At these hours**: 17 (5 PM)
    - **At these minutes**: 0
5. **Activated**: ☑ Yes
6. Click **OK** → **OK**
7. **Publish** the trigger

Result: The pipeline will automatically execute every day at 5 PM.

🧪 STEP 6: Test the Pipeline Manually

6.1 Manual Execution

1. In your pipeline FX_ETL_Pipeline
2. Top → **Add trigger** → **Trigger now**
3. **Pipeline run parameters**:
    - **StartDate**: 2024-01-01
4. Click **OK**

6.2 Monitor Execution

1. In ADF Studio → **Monitor** (chart icon) → **Pipeline runs**
2. See your pipeline in progress:
    - 🟡 **In Progress**
    - 🟢 **Succeeded**
    - 🔴 **Failed**
3. Click on the **pipeline name** to see details of each activity

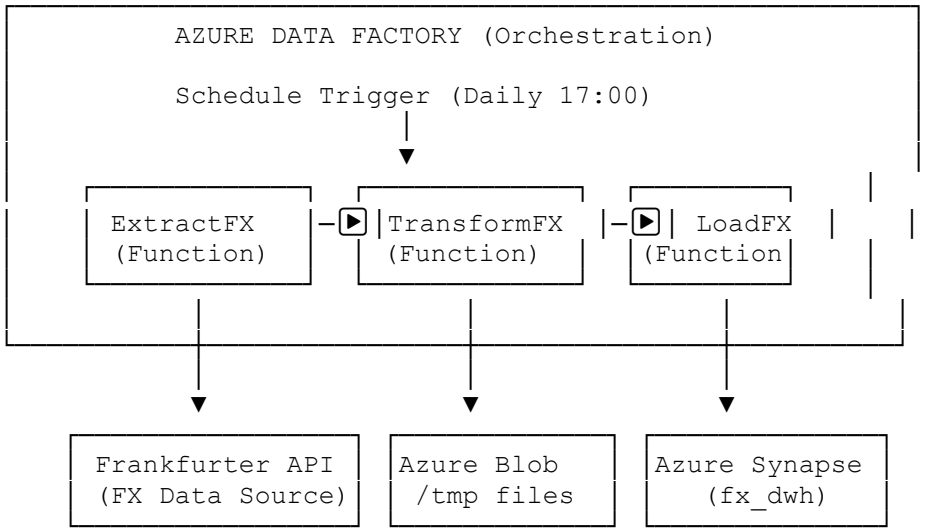
STEP 7: Verify Results in Azure SQL

1. Open **Azure Data Studio** or **SQL Server Management Studio**
2. Connect to your Azure SQL Database
3. Execute:

```
-- Verify data
SELECT COUNT(*) FROM fact_fx_rates_daily;
SELECT COUNT(*) FROM fact_fx_rates_ytd;

-- View pipeline logs
SELECT * FROM pipeline_execution_log
ORDER BY execution_date DESC;
```

Final Visual Diagram



Complete Checklist

- [ ] **Azure Blob Storage**: Scripts uploaded (config, extract, transform, load)
- [ ] **Function App**: Created and configured
- [ ] **Application Settings**: Environment variables added
- [ ] **3 Azure Functions**: ExtractFX, TransformFX, LoadFX created
- [ ] **Linked Services**: Azure Function + Azure Synapse configured
- [ ] **ADF Pipeline**: 3 activities connected
- [ ] **Trigger**: Daily schedule at 5 PM
- [ ] **Manual test**: Pipeline executed successfully
- [ ] **SQL Verification**: Data present in tables

🎯 Final Result

- ☑ **Automated pipeline** that executes every day at 5 PM
- ☑ **Graphical orchestration** in Azure Data Factory
- ☑ **3 serverless Azure Functions** (cost-effective)
- ☑ **Monitoring** via Azure portal
- ☑ **No server management required** (fully managed)