Data Orchestration Platform

Comprehensive Implementation Summary

Apache Airflow & Prefect Integration Guide

Document Date: December 16, 2025

Document Version: 1.0

Scope: Production Deployment Configuration

Table of Contents

1. Executive Summary

Overview: This document provides a comprehensive guide for implementing data orchestration pipelines using two leading platforms: Apache Airflow and Prefect. The project focuses on building an automated currency exchange rate acquisition and processing system.

Key Outcomes:

✓ Successfully implemented currency acquisition pipeline using Prefect 3.6.5

✓ Hybrid execution model combining Cloud monitoring with local Windows Task Scheduler automation

✓ Resolved critical infrastructure challenges (entrypoint configuration, Python path resolution)

✓ Established reliable monthly execution schedule (17th of each month at 11:00, 11:30, 12:00 Europe/Zurich)

✓ Production-ready configuration with comprehensive monitoring and logging

2. Apache Airflow Overview & Windows Platform Considerations

2.1 What is Apache Airflow?

Apache Airflow is an open-source workflow orchestration platform that allows users to define, schedule, and monitor complex workflows through Python code. It uses Directed Acyclic Graphs (DAGs) to represent data pipelines and provides a rich web UI for monitoring and management.

2.2 Windows Compatibility Issues

Issue 1: Unix-Specific Dependencies

Apache Airflow depends on the fcntl module, which is exclusively available on Unix-based systems (Linux, macOS). This module handles file locking and Unix-specific I/O operations. Windows lacks this module, resulting in ModuleNotFoundError: No module named 'fcntl' errors when attempting direct installation on Windows.

Issue 2: File Path Handling Differences

Airflow's codebase uses Unix-style file paths extensively. Windows uses backslashes (\\) for path separation, while Unix uses forward slashes (/). This inconsistency can cause import failures and configuration errors when running Airflow directly on Windows.

2.3 Recommended Solution: Docker Containerization

The recommended approach for running Apache Airflow on Windows is through **Docker Desktop**, which provides a Linux containerization layer. This solution:

• Provides a complete Linux environment inside a container

• Eliminates Windows compatibility issues entirely

• Allows seamless DAG development and testing

• Maintains consistency with production environments (typically Linux)

• Enables easy scaling and deployment to cloud platforms

2.4 Docker Implementation Steps

Step 1: Install Docker Desktop

Download and install Docker Desktop from the official Docker website. After installation, ensure the Docker daemon is running.

Step 2: Pull and Run Airflow Container

Execute the following command in CMD/PowerShell:

```
docker run -it --rm --name airflow-devel -p 8080:8080 apache/airflow airflow standalone
```

This command:
• -it: Interactive terminal mode
• --rm: Automatically removes container on exit
• -p 8080:8080: Maps port 8080 for web UI access
• airflow standalone: Runs in standalone mode (all-in-one setup)

Step 3: Access the Airflow Web UI

Once the container is running, open http://localhost:8080 in your browser to access the Airflow UI.

Step 4: Upload DAG Files

Use Docker's cp command to transfer DAG files into the container:

```
docker cp quick_test_dag.py airflow-devel:/opt/airflow/dags
```

Step 5: Test DAG Execution

Enter the container shell and test DAG execution:

```
airflow dags test quick_test_dag
```

Step 6: View Execution Logs

After the DAG runs, inspect execution results and logs through the web UI or container terminal.

3. Prefect Platform: Architecture & Implementation

3.1 Prefect Overview

Prefect is a modern, open-source workflow orchestration platform designed to be flexible and user-friendly. Unlike Airflow's DAG model, Prefect uses a more Pythonic approach with tasks and flows, emphasizing developer experience and observability.

Key Differences from Airflow:

• Native Python functions as tasks (no special syntax required)

• Better error handling with built-in retries and timeouts

• First-class cloud support (Prefect Cloud) with UI hosted in the cloud

• Simpler worker management without complex broker setup

• Improved task dependency management

3.2 Project Implementation: Currency Exchange Rate Pipeline

This implementation demonstrates Prefect's capabilities with a real-world currency acquisition pipeline.

Architecture Overview:

Component	Technology	Purpose
Flows	Python 3.11.9	Define data pipeline workflows
Deployments	Prefect 3.6.5	Package and configure flows
Schedules	Prefect Cloud	Trigger execution at specified times
Workers	Windows Task Scheduler	Execute flows locally
Storage	Local File System	Store pipeline data
Monitoring	Prefect Cloud UI	View execution history and logs

3.3 Core Implementation Components

Flow 1: Currency Acquisition Flow

Purpose: Fetches current exchange rates from external APIs and stores them locally.
Deployment: currency-acquisition
Schedule: Monthly on 17th at 11:00 Europe/Zurich
Output: CSV file with exchange rates (exchange_rates_YYYY_MM.csv)

Flow 2: Prepare Batch Flow

Purpose: Prepares data for batch processing by formatting and organizing exchange rate data.
Deployment: prepare-batch
Schedule: Monthly on 17th at 11:30 Europe/Zurich

Dependency: Runs after currency-acquisition flow

Flow 3: Process Batch Flow

Purpose: Processes the prepared batch data and generates final output files.
Deployment: process-batch
Schedule: Monthly on 17th at 12:00 Europe/Zurich
Dependency: Runs after prepare-batch flow

4. Comparative Analysis: Airflow vs Prefect

Aspect	Apache Airflow	Prefect
Learning Curve	Moderate to Steep	Gentle
Windows Support	Requires Docker	Native Support
Execution Model	DAGs	Flows & Tasks
Error Handling	Basic retry mechanism	Advanced with better defaults
Cloud Solution	Third-party (Astronomer)	Prefect Cloud (built-in)
Dependency Management	Complex (Celery/RabbitMQ)	Simplified with Workers
Development Experience	Good	Excellent
Monitoring UI	Self-hosted	Cloud-hosted (free tier available)
Cost	Free (self-hosted)	Free tier + paid plans
Production Readiness	Very High	Very High

4.1 Selection Rationale

For this project, **Prefect was selected over Apache Airflow** for the following reasons:

1. **Native Windows Compatibility:** No Docker required for local development and testing
2. **Simpler Setup:** Minimal configuration required to get started
3. **Better Developer Experience:** Pythonic API makes development faster
4. **Cloud-Ready:** Prefect Cloud integration provides monitoring without additional infrastructure
5. **Cost Efficiency:** Free tier suitable for this use case

5. Currency Exchange Rate Pipeline - Prefect Implementation

5.1 System Architecture

The implementation uses a hybrid execution model combining cloud monitoring with local execution:

Layer	Component	Status	Purpose
Cloud	Prefect Cloud Schedules	Active (monitoring)	Track execution history
Cloud	Prefect Cloud UI	Active	View flow execution logs
Local	Windows Task Scheduler	Active (execution)	Trigger flows at scheduled times
Local	Python Interpreter	Active	Execute flow code
Local	Data Storage	Active	Store exchange rate data

5.2 Critical Implementation Decisions

Decision 1: Hybrid Execution Model (Cloud + Local)

Initially attempted to run flows directly in Prefect Cloud, but encountered a limitation: Cloud's prefect:managed work pool cannot access local code files stored on the user's computer. Solution: Configure Windows Task Scheduler to trigger local Python execution while maintaining Cloud monitoring. This provides the best of both worlds: reliable execution + centralized logging.

Decision 2: Windows Task Scheduler Over Prefect Cloud Scheduler

While Prefect Cloud provides scheduling capabilities, the local execution model necessitates Windows Task Scheduler as the actual trigger mechanism. Cloud schedules are maintained for monitoring and historical record-keeping.

Decision 3: Absolute Python Path in Batch Files

Critical fix: All batch files must use the absolute Python path (C:\Program Files\Python311\python.exe) rather than relative python command. Task Scheduler doesn't inherit the system PATH, causing execution failures with relative paths.

6. System Configuration & Deployment

6.1 Environment Specifications

Component	Value
Operating System	Windows 11
Python Version	3.11.9
Prefect Version	3.6.5
Timezone	Europe/Zurich (UTC+1/+2)
Work Pool	Yichen_Test (prefect:managed)

6.2 Deployment Configuration

Deployment	Entrypoint	Schedule	Timezone
currency-acquisition	flows/currency_acquisition_flow.py: currency_acquisition_flow	Monthly 17th at 11:00	Europe/Zurich
prepare-batch	flows/prepare_batch_flow.py: prepare_batch_flow	Monthly 17th at 11:30	Europe/Zurich
process-batch	flows/process_batch_flow.py: process_batch_flow	Monthly 17th at 12:00	Europe/Zurich

6.3 Configuration Files Overview

prefect.yaml: Defines deployments with entrypoint specifications

run_flows_locally.py: Local execution wrapper that imports and runs all flows

run_Prefect-*.bat (3 files): Batch files triggered by Task Scheduler with full Python path

requirements.txt: Python dependencies (Prefect, pandas, etc.)

check_status.py: Verification script to check deployment and schedule status

6.4 Key Configuration Parameters

Python Installation Path: C:\\Program Files\\Python311\\python.exe
Project Root: C:\\Users\\yli\\Desktop\\Prefect_Project
Data Output Directory: C:\\Users\\yli\\Desktop\\Prefect_Project\\data\\
Flows Directory: C:\\Users\\yli\\Desktop\\Prefect_Project\\flows\\
Utilities Directory: C:\\Users\\yli\\Desktop\\Prefect_Project\\utils\\

7. Operational Procedures & Monitoring

7.1 Daily Operational Checks

Prefect Cloud UI Status: Verify that Cloud is accessible and showing current deployment status

Windows Task Scheduler: Confirm all 3 tasks are enabled and showing correct schedule

Data Directory: Check for presence of latest exchange rate files

System Logs: Review Windows Event Viewer for any Task Scheduler errors

7.2 Monitoring Dashboard

Access Prefect Cloud at https://app.prefect.cloud to view:
• Real-time flow execution status
• Historical execution logs and run times
• Performance metrics and error tracking
• Schedule status and next scheduled execution times

7.3 Verification Commands

Check Current Status:

```
python check_status.py
```

Displays deployment status, schedule information, and configuration

Manual Flow Execution:

```
python run_flows_locally.py
```

Manually triggers all three flows for testing purposes

Task Scheduler Query:

```
schtasks /query | findstr Prefect
```

Lists all Prefect-related tasks in Windows Task Scheduler

Create Deployment:

```
prefect deployment create flows/currency_acquisition_flow.py:currency_acquisition_flow
--name currency-acquisition
```

Creates a new deployment in Prefect

Create Cloud Schedule:

```
prefect deployment schedule create [deployment/name] --cron "0 11 17 * *" --timezone
"Europe/Zurich"
```

Creates a monthly schedule at 11:00 on the 17th

8. Troubleshooting & Lessons Learned

8.1 Common Issues & Solutions

Issue 1: "Deployment does not have an entrypoint"

Root Cause: Deployment created without specifying entrypoint parameter

Solution: Recreate deployment with full entrypoint: flows/flow_name.py:flow_name

Issue 2: Task Scheduler returns exit code -1073741510

Root Cause: Batch file uses relative python path which is not in Task Scheduler PATH

Solution: Use absolute Python path: C:\\Program Files\\Python311\\python.exe

Issue 3: "ModuleNotFoundError: No module named fcntl" (if using Airflow)

Root Cause: fcntl is Unix-specific and not available on Windows

Solution: Use Docker Desktop to run Airflow in a Linux container

Issue 4: Cloud Schedules not executing

Root Cause: Cloud prefect:managed cannot access local code files

Solution: Use local execution model with Windows Task Scheduler

Issue 5: Cloud UI shows old schedule times

Root Cause: Schedules updated locally but Cloud configuration not synchronized

Solution: Delete old Cloud schedules and create new ones with correct cron expressions

8.2 Lessons Learned

1. **Always use absolute paths in batch files** when using Task Scheduler, as relative paths may not resolve correctly

2. **Synchronize configuration across systems** - when updating schedules, update both Cloud and local configs to maintain consistency

3. **Docker is essential for Airflow on Windows** - native Windows support is unreliable due to Unix-specific dependencies

4. **Prefect's local execution is more flexible** than Cloud execution for scenarios with local code and data

5. **Test thoroughly before production** - manual execution tests revealed issues that wouldn't surface in dry runs

6. **Monitor both Cloud and local execution** - maintain awareness of both systems for complete troubleshooting visibility

7. **Document configuration changes** - track all schedule and deployment changes for audit and recovery purposes

9. Future Roadmap & Recommendations

9.1 Production Configuration Update

Current test configuration (monthly 17th) should be updated to production schedule once validated. Production dates: 15th, 25th, 28th-31st of each month at 11:00, 11:30, 12:00 (Europe/Zurich).

9.2 Enhanced Monitoring

Recommendations for monitoring improvements:
• Implement email notifications for failed executions
• Set up Slack integration for real-time alerts
• Create custom dashboards for data quality metrics
• Implement automated backup verification

9.3 Scalability Considerations

To scale this system:
• **Migrate to Cloud Execution:** Refactor to use Cloud-compatible code structure
• **Implement Database Storage:** Move from CSV files to database for better querying
• **Add Data Validation:** Implement automated data quality checks
• **Enable Concurrent Flows:** Modify dependencies to allow parallel execution where possible
• **Implement Versioning:** Track flow version history and rollback capabilities

9.4 Technology Evolution Path

Short-term (0-6 months):
• Finalize production configuration and validate monthly execution
• Implement comprehensive monitoring and alerting
• Document runbooks for common operations

Medium-term (6-12 months):
• Evaluate cloud migration (Azure, AWS, GCP)
• Implement data warehouse for historical analysis
• Explore Prefect Cloud's advanced features

Long-term (12+ months):
• Consider transitioning to fully cloud-native architecture
• Integrate with business intelligence tools
• Expand to multi-region deployment for redundancy

Appendix: Quick Reference Guide

A.1 Critical File Paths

Python Executable: C:\\Program Files\\Python311\\python.exe
Project Root: C:\\Users\\yli\\Desktop\\Prefect_Project
Flow Files: C:\\Users\\yli\\Desktop\\Prefect_Project\\flows\\
Data Output: C:\\Users\\yli\\Desktop\\Prefect_Project\\data\\
Batch Files: C:\\Users\\yli\\Desktop\\Prefect_Project\\run_Prefect-*.bat

A.2 Essential URLs

Prefect Cloud: https://app.prefect.cloud
Prefect Documentation: https://docs.prefect.io
Apache Airflow: https://airflow.apache.org
Airflow Documentation: https://airflow.apache.org/docs/apache-airflow

A.3 Support Contact Information

Project Owner: Yichen Li
Prefect Cloud Account: Associated with yli@intracen.com
Documentation Location: C:\\Users\\yli\\Desktop\\Prefect_Project*.md

Document Information
Generated: December 16, 2025 at 17:58:09
Version: 1.0
Classification: Technical Documentation