Airflow at UniCredit
Our journey from mainframe scheduling to modern data processing

Jan Pawłowski
Jędrzej Matuszak
Airflow at UniCredit

Speakers

Jan Pawłowski
Murex Reporting Team Head

Jędrzej Matuszak
Python Developer

Our Warsaw team manages Front Office data feeds across the UC Group.

Scale of daily activity:
• 8k batch tasks executed
• 5k data files dispatched
• 2.5bn data points processed
Agenda

1. Legacy issues – why we moved to Airflow
2. How we got here
3. Moving to Airflow - the challenges
4. The target solution
5. What we gained
Mainframe scheduler

- Difficult maintenance: manual change process involving multiple teams
  (e.g. team A misreads request from team B and removes a task instead of moving it)

- Limited number of available environments/instances

- Only one run scheduled per day – no test flexibility

- No CI process – no version control/automated testing

- Long time to market: minimum 1 day for simple changes

- Poor resource control due to rigid scheduling model
Legacy issues: why we moved to Airflow

The mainframe scheduler's most painful issue: no dependency visualization.

Thousands of tasks with many-to-many dependencies created a huge dependency network.
Airflow at UniCredit
How we got here

- Python scheduler script: a major scheduling upgrade vs mainframe
- We later decided to shift to an existing solution: too much effort to keep upgrading in-house
- This led us to finding Airflow

May 2020
40% of tasks migrated to in-house scheduler

May 2022
first Airflow deploy in Production
5% of tasks migrated

April 2023
90% of tasks migrated

August 2023
Airflow upgrade

October 2021
Airflow POC

October 2022
50% of tasks migrated

July 2023
92% of tasks migrated

[End of 2024]
100% of tasks migrated

- After the POC, initiatives to shift our scheduling into Airflow were launched
- Production deployments followed in 2022 and 2023
Moving to Airflow: the challenges

1. Understanding our own **scheduling dependencies** to be able to redefine and simplify them -> months of analytical work
2. **Production** environment **requirements** (eg. incident management requiring a mainframe task crash in the event of a Production failure)
3. **Integration** with overall IT landscape (other systems to continue using mainframe scheduling)
4. Understanding the concept of Airflow **DAG dependencies** – how to trigger a DAG upon another DAG's success?
5. Airflow resets DAGs after crash + restart, instead of resuming their execution – how to speed up **failure recovery**?
Airflow at UniCredit

The „main DAG” concept – our final solution

Structure: one main DAG with multiple sub-DAGs

Scope: > 8k batch tasks executed daily
Airflow at UniCredit

The target solution - plugins

- **ResumeDagRunOperator** - evolution of the TriggerDagRunOperator functionality

- **DAG task search**

- **Static time predecessors**

- **Sensor** to provide incident management (checks for failed tasks and crashes if any are detected)
Example: Our 100-task DAG `sample_5` crashes at 90 tasks. To save time, it is better to resume the DAG run, rather than rerunning the whole scope.

Clearing `sample_5` will restart the DagRun (rerunning 100 tasks) or raise a DagRunAlreadyExists error.

Clearing `sample_5` will resume the DagRun (rerunning only 10 tasks).
Airflow at UniCredit

Custom DateTimeSensor

**Problem:** the parent Airflow process is ran manually* at 9pm, while NY sub-tasks are scheduled for 1am. In case of a severe delay (eg. parent process starts at 3am), NY tasks would start on 1am of the next calendar day.

**Solution:** thanks to **offset_duration**, the custom DateTimeSensor allows dependencies to be met even in this scenario: NY tasks are triggered together with the parent.

*the Airflow execution_date is the same as DagRun start_date*
Problem: While debugging an application crash, the team has to quickly locate the DAG in which the failed task is located.

Solution: A new search option was added to the Airflow toolbar, allowing us to quickly locate tasks in our DAGs.
The target framework – a hybrid approach

We combined Airflow and mainframe scheduling:

- Incident management via Airflow task fail sensor (new feature)
- Downstream dependencies to other systems via mainframe
Airflow at UniCredit

What we gained

- Automated testing (CI with Jenkins)
- Test capacity increased, enabling multiple runs per day across many environments
- Versioning (eg. possibility of defining branching strategies for parallel projects)
- Scheduling as code
- Scalability
- Task latency reduced from 10-60s to 3-10s
- Dependency visualization
- Block approach for tasks
- POC for Airflow in UniCredit - we're open to discuss best practices for usage at scale

- 4 main DAGs
- >250 sub DAGs
- 480 daily DAG runs
- ~8000 daily task runs
Questions?

Reach us at:  
Jan.Pawlowski@unicredit.eu  
Jedrzej.Matuszak@unicredit.eu