

Orchestrierung mittels Airflow

vorgelegt am 19.01.2025

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2022F

von

MEHMET KARACA

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	1
1 Einleitung	1
1.1 Hintergrund und Motivation	1
1.2 Automatisierung und Orchestrierung	1
1.3 Architektur und Funktionsweise von Apache Airflow	1
1.4 Komponenten von Apache Airflow	3
1.5 Die Weboberfläche von Apache Airflow	5
1.6 Vorteile von Apache Airflow	7
1.7 Anwendungsbeispiele	7
2 Installation und Umsetzung	8
2.1 Vorbereitung	8
2.2 Initialisierung der Airflow-Datenbank	8
2.3 Einrichtung der Datenbank	9
2.4 Starten von Airflow	9
2.5 Überprüfung des Workflows	10
3 Codebeispiel	11
3.1 Einführung in den Workflow	11
3.2 Definition des DAGs	11
3.3 Die SQL-Skripte	13
Literaturverzeichnis	15

Abbildungsverzeichnis

1	Unterschied zwischen einem DAG und einem zyklischen Graphen. ¹	2
2	Überblick über alle beteiligten Komponenten in Airflow. ²	4
3	Login-Seite der Weboberfläche von Apache Airflow. ³	5
4	Hauptseite von Airflow. ⁴	6
5	Codeausschnitt von <code>example_dag.py</code> .	11
6	Codeausschnitt von <code>aggregate_data_dag.py</code> .	12
7	Codeausschnitt von <code>create_and_insert.sql</code> .	13
8	Codeausschnitt von <code>aggregate_and_store.sql</code> .	14

1 Einleitung

1.1 Hintergrund und Motivation

Apache Airflow ist die am häufigsten verwendete Open-Source-Plattform zur Workflow Orchestrierung. Das Projekt wurde Ende 2014 als internes Projekt bei Airbnb initiiert, um eine Lösung für das Problem der zunehmend komplexen Workflows zu finden. Das Ziel bestand in der Schaffung einer zentralen Plattform zur Erstellung und Planung von Aufträgen über eine einfache Benutzeroberfläche. Im Wesentlichen definiert ein Job seinen eigenen Zeitplan und seine Konfiguration innerhalb eines einfachen Python-Skripts. Dieser codegesteuerte Ansatz für die Auftragsplanung ermöglicht eine umfassende Anpassung, die mit einer statischen Konfiguration allein nur schwer zu erreichen ist. Es sei darauf hingewiesen, dass dieser Ansatz für Unternehmen der Größenordnung von Airbnb vorteilhaft sein kann. Die Frage, die sich in diesem Zusammenhang stellt, ist, warum man sich für Airflow oder einen anderen automatisierten Workflow-Orchestrator entscheiden sollte.⁵

1.2 Automatisierung und Orchestrierung

In Phasen zunehmender Arbeitsbelastung ist die Bewältigung neuer Aufgaben durch Teams häufig nur durch das Abgeben oder Automatisieren bestehender Tätigkeiten möglich. Während kleinere Teams solche Aufgaben zunächst manuell verwalten können, wird langfristig die Automatisierung als entscheidender Faktor für den Erfolg hervorgehoben. Die Automatisierung von repetitiven und wenig erfüllenden Arbeiten ermöglicht die Freisetzung von Ressourcen, die anderweitig eingesetzt werden können. Dies kann die Erreichung neuer Ziele fördern und die Leistungsfähigkeit des Teams steigern, ohne dass eine proportionale Vergrößerung des Teams notwendig wäre. Sobald jedoch auch automatisierte Prozesse einer Verwaltung bedürfen, werden Orchestrierungswerkzeuge wie Apache Airflow eingesetzt, um diese effizient zu steuern.⁶

1.3 Architektur und Funktionsweise von Apache Airflow

Airflow wurde konzipiert, um den Herausforderungen der zeitgenössischen Datenverarbeitung zu begegnen, die durch die Zusammenführung, Verarbeitung und Analyse großer Datenmengen aus diversen Quellen charakterisiert ist. Die Plattform zeichnet sich durch eine flexible und skalierbare Architektur aus, welche die Definition von Workflows als Code ermöglicht. Dieser

⁵Vgl. Haines [2022](#), 256 f.

⁶Vgl. Haines [2022](#), 256 f.

Ansatz erleichtert nicht nur die Wiederverwendbarkeit und Wartung, sondern fördert auch die Zusammenarbeit zwischen Teams.⁷

An dieser Stelle sei Airflow erwähnt. Bei Airflow handelt es sich um ein Open-Source-Tool zur Orchestrierung von Workflows, das speziell für die Verwaltung und Automatisierung komplexer Datenpipelines entwickelt wurde. Es ermöglicht die Planung, Überwachung und Verwaltung von Prozessen, die aus mehreren voneinander abhängigen Aufgaben bestehen. Die Organisation dieser Aufgaben erfolgt mittels *Directed Acyclic Graphs (DAGs)*, welche die sequentielle Abfolge und Abhängigkeiten der einzelnen Schritte eines Workflows definieren.⁸

- **Directed Acyclic Graphs (DAGs):** Ein Directed Acyclic Graph (DAG) ist eine spezielle Art von Graph, der aus Knoten (Tasks) und gerichteten Kanten (Abhängigkeiten) besteht und keine Zyklen enthält. In der Abbildung wird der Unterschied zwischen einem DAG und einem zyklischen Graphen anhand eines Beispiels illustriert.

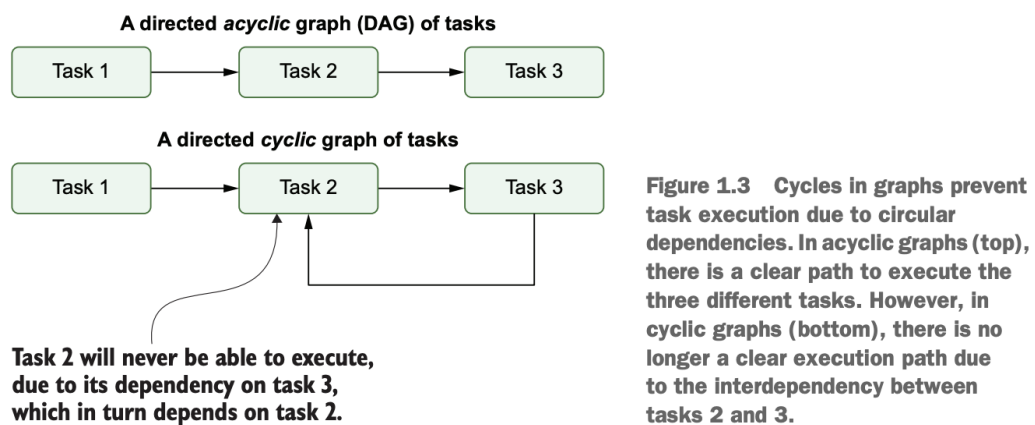


Abb. 1: Unterschied zwischen einem DAG und einem zyklischen Graphen.⁹

Im oberen Teil der Abbildung wird ein DAG dargestellt, in dem die Tasks 1, 2 und 3 in einer klaren Reihenfolge ausgeführt werden können. Die Pfeile zwischen den Tasks zeigen die Abhängigkeiten an: Task 1 muss abgeschlossen sein, bevor Task 2 starten kann, und Task 2 muss abgeschlossen sein, bevor Task 3 ausgeführt wird. Da es keine Zyklen gibt, gibt es immer einen eindeutigen Pfad zur Ausführung der Aufgaben.

Im unteren Teil der Abbildung wird ein zyklischer Graph gezeigt, bei dem eine zyklische Abhängigkeit zwischen den Tasks 2 und 3 besteht. Task 2 hängt von Task 3 ab, und Task 3 hängt wiederum von Task 2 ab. Diese gegenseitige Abhängigkeit führt zu einem Deadlock: Keiner der beiden Tasks kann ausgeführt werden, da beide auf die Fertigstellung des jeweils anderen warten. Solche Zyklen machen die Ausführung eines Workflows unmöglich.

⁷Vgl. Nara/Shaikh/Pradhan 2023.

⁸Vgl. Nara/Shaikh/Pradhan 2023.

⁹Vgl. Harensak/de Ruiter 2021, 6 f.

Die Literatur betont die Bedeutung von DAGs in der Workflow-Orchestrierung, insbesondere in Systemen wie Apache Airflow. Ein DAG stellt sicher, dass Workflows effizient und ohne logische Inkonsistenzen ausgeführt werden können. Durch die Vermeidung von Zyklen können Aufgaben parallelisiert werden, wo dies möglich ist, was die Ausführungszeit verkürzt und Ressourcen optimal nutzt.¹⁰

1.4 Komponenten von Apache Airflow

Ein DAG definiert den Workflow als eine Sammlung von Aufgaben (*Tasks*), die in einer bestimmten Reihenfolge ausgeführt werden müssen. Die Struktur des DAGs stellt sicher, dass keine zyklischen Abhängigkeiten entstehen. Jede Aufgabe im Workflow wird durch einen Task repräsentiert. Die Tasks werden wie folgt beschrieben:¹¹

- **Operators:** *Operators* sind vorgefertigte Bausteine, die spezifische Aktionen ausführen, wie z. B. das Ausführen eines Python-Skripts (`PythonOperator`) oder eines Shell-Befehls (`BashOperator`).
- **Scheduler:** Der *Scheduler* liest die DAG-Dateien aus, analysiert die Abhängigkeiten und plant die Ausführung der Tasks basierend auf den definierten Zeitplänen. Er überprüft kontinuierlich, ob Tasks ausgeführt werden müssen, und fügt sie der Warteschlange hinzu.
- **Executor:** Der *Executor* führt die geplanten Tasks aus. Airflow unterstützt verschiedene Executors wie den `LocalExecutor` für lokale Ausführung oder den `CeleryExecutor` für verteilte Umgebungen.
- **Workers:** Die geplanten Tasks werden an Airflow-Worker weitergeleitet, die für deren Ausführung verantwortlich sind. Die Worker führen die Aufgaben aus und speichern die Ergebnisse sowie Statusinformationen.
- **Metadatenbank:** Alle Informationen über DAGs, Task-Zustände und Logs werden in einer zentralen Metadatenbank gespeichert. Diese Datenbank ist das Herzstück von Airflow und ermöglicht eine konsistente Verwaltung aller Workflows.
- **Webserver:** Der Webserver bietet eine benutzerfreundliche Oberfläche zur Überwachung und Verwaltung von Workflows. Benutzer können DAGs visualisieren, deren Status prüfen und Logs einsehen.

¹⁰Vgl. Abbildung 1.

¹¹Vgl. Nara/Shaikh/Pradhan 2023.

Interaktion zwischen den Komponenten

Die hier genannten Komponenten wirken zusammen, und die folgende Abbildung zeigt die Hauptkomponenten von Airflow sowie ihre Wechselwirkungen bei der Ausführung solcher Pipelines:

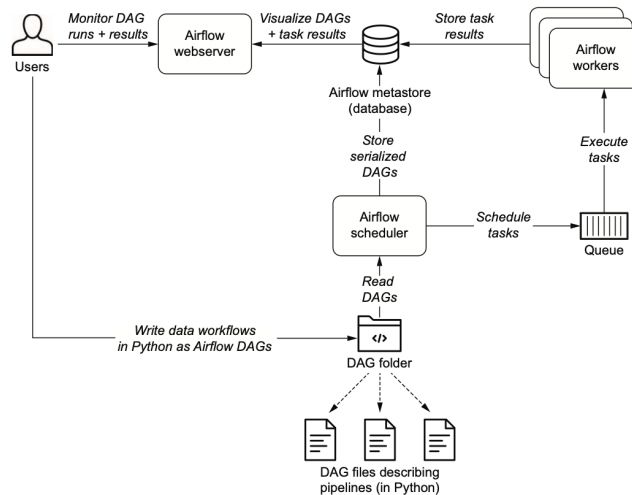


Figure 1.8 Overview of the main components involved in Airflow (e.g., the Airflow webserver, scheduler, and workers)

Abb. 2: Überblick über alle beteiligten Komponenten in Airflow.¹²

Ein DAG in Airflow beschreibt eine Datenpipeline als eine Sammlung von Tasks (Aufgaben), die in einer bestimmten Reihenfolge ausgeführt werden müssen. Benutzer schreiben diese Workflows in Python und speichern sie in einer speziellen DAG-Ordnerstruktur. Der Airflow Scheduler liest diese DAG-Dateien aus und plant die Ausführung der Tasks basierend auf ihren definierten Abhängigkeiten und Zeitplänen.¹³

Die geplanten Tasks werden dann an eine Warteschlange übergeben, wo sie von Airflow-Workern ausgeführt werden. Diese Worker sind für die tatsächliche Verarbeitung der Aufgaben zuständig. Während der Ausführung speichern sie Ergebnisse und Statusinformationen in der Metadatenbank von Airflow. Diese Datenbank dient als zentrales Repository für den Zustand und Verlauf aller DAGs und Tasks.¹⁴

Der Airflow-Webserver bietet eine Benutzeroberfläche, über die Nutzer ihre DAGs visualisieren, überwachen und analysieren können. Hier können sie den Status der Tasks einsehen, Fehler diagnostizieren und Logs überprüfen. Durch diese Visualisierung wird es einfacher, komplexe Workflows zu verstehen und zu verwalten.¹⁵

¹²Vgl. Harensak/de Ruiter 2021 S. 12.

¹³Vgl. Harensak/de Ruiter 2021 12 f.

¹⁴Vgl. Harensak/de Ruiter 2021 12 f.

¹⁵Vgl. Harensak/de Ruiter 2021 12 f.

1.5 Die Weboberfläche von Apache Airflow

Die Weboberfläche von Apache Airflow ist ein zentrales Werkzeug zur Überwachung und Verwaltung von Workflows. Sie bietet eine benutzerfreundliche Möglichkeit, DAGs (Directed Acyclic Graphs) zu visualisieren, deren Status zu prüfen und detaillierte Informationen zu einzelnen Tasks einzusehen.

Beim ersten Zugriff auf die Weboberfläche wird der Benutzer mit einer Login-Seite begrüßt, wie in Abbildung 3 dargestellt. Hier können Benutzername und Passwort eingegeben werden, um Zugang zur Plattform zu erhalten. Standardmäßig wird bei der Einrichtung von Airflow ein Benutzer mit den Anmeldedaten „admin“ für Benutzername und Passwort erstellt, was für erste Tests nützlich ist.

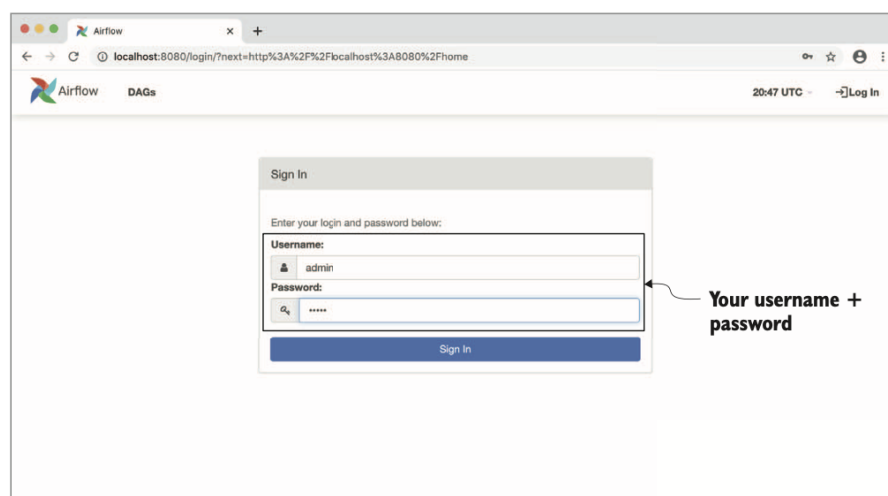


Figure 1.10 The login page for the Airflow web interface. In the code examples accompanying this book, a default user “admin” is provided with the password “admin.”

Abb. 3: Login-Seite der Weboberfläche von Apache Airflow.¹⁶

Nach dem erfolgreichen Login gelangt man zur Startseite der Weboberfläche (siehe Abbildung 4). Diese Seite bietet eine Übersicht über alle registrierten DAGs (Directed Acyclic Graphs), die in Airflow definiert wurden. Für jeden DAG werden wichtige Informationen angezeigt, darunter:

- **Name des DAGs:** Der Titel des Workflows, der in der Konfiguration definiert wurde.
- **Zeitplan (Schedule):** Gibt an, wie häufig der Workflow ausgeführt wird, z. B. täglich oder wöchentlich.
- **Status der letzten Ausführungen:** Zeigt den Zustand der Tasks aus den letzten Durchläufen an (z. B. erfolgreich, fehlgeschlagen oder laufend).

¹⁶Vgl. Harensak/de Ruiter 2021, S. 14.

Die Startseite ermöglicht es Benutzern, Workflows zu aktivieren oder zu pausieren sowie spezifische DAGs manuell auszuführen. Über die bereitgestellten Filter- und Suchoptionen können Benutzer zudem gezielt nach bestimmten Workflows suchen.

Die intuitive Gestaltung der Weboberfläche erleichtert es Entwicklern und Administratoren, komplexe Workflows effizient zu überwachen und zu verwalten. Sie stellt sicher, dass alle wichtigen Informationen leicht zugänglich sind und bietet Funktionen zur Fehlerdiagnose und Optimierung von Workflows.

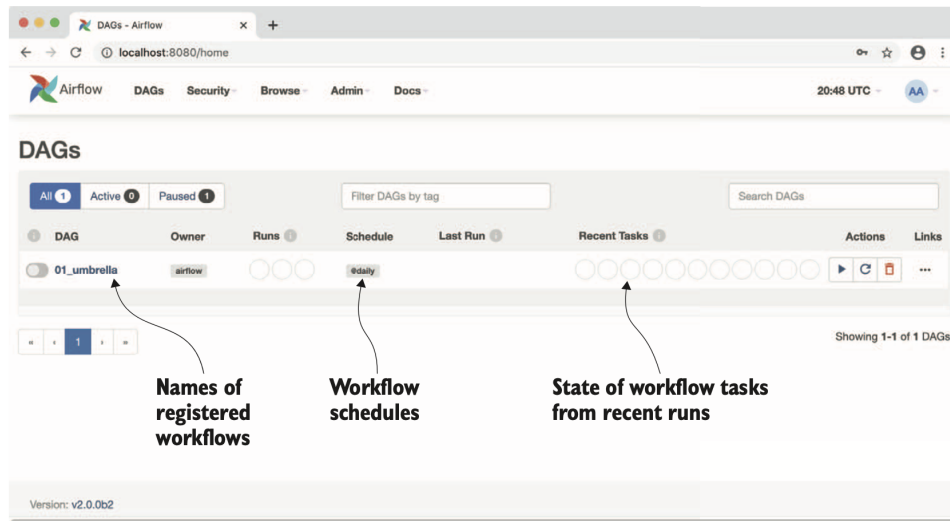


Figure 1.11 The main page of Airflow's web interface, showing an overview of the available DAGs and their recent results

Abb. 4: Hauptseite von Airflow.¹⁷

Die Weboberfläche bietet zudem verschiedene Ansichten für einzelne DAGs:

- **Graph View:** Zeigt die Struktur des DAGs als Diagramm an, wobei die Abhängigkeiten zwischen den Tasks durch Pfeile dargestellt werden.
- **Tree View:** Stellt die Ausführungshistorie eines DAGs in einer tabellarischen Form dar, wobei jede Spalte eine Ausführung repräsentiert.
- **Logs:** Ermöglicht den Zugriff auf detaillierte Protokolle einzelner Tasks, um Fehlerdiagnosen durchzuführen oder den Fortschritt nachzuvollziehen.

Die intuitive Gestaltung der Weboberfläche macht sie zu einem unverzichtbaren Werkzeug für Entwickler und Administratoren, um komplexe Workflows effizient zu verwalten und zu überwachen.

¹⁷Vgl. Harensak/de Ruiter 2021, S. 14.

1.6 Vorteile von Apache Airflow

Ein zentraler Vorteil von Apache Airflow ist seine Fähigkeit zur Planung und Automatisierung von Aufgaben. Benutzer können Zeitpläne für wiederkehrende Prozesse definieren, wie z. B. tägliche Datenaktualisierungen oder monatliche Berichte. Der integrierte Scheduler von Airflow sorgt dafür, dass Aufgaben gemäß den definierten Zeitplänen oder durch externe Trigger ausgeführt werden. Diese Automatisierungsfähigkeit reduziert manuelle Eingriffe und minimiert potenzielle Fehlerquellen.^[18]

Ein weiteres herausragendes Merkmal ist das Monitoring und Debugging über eine webbasierte Benutzeroberfläche. Diese bietet Einblicke in den Status der Workflows, zeigt potenzielle Fehler an und ermöglicht es den Benutzern, detaillierte Protokolle einzelner Aufgaben einzusehen.^[19]

Zusätzlich unterstützt Airflow eine Vielzahl von Operatoren und Plugins, die die Integration mit externen Systemen wie Datenbanken, APIs oder Cloud-Diensten ermöglichen.

1.7 Anwendungsbeispiele

Die Anwendungsmöglichkeiten von Apache Airflow sind breit gefächert. Es wird häufig für die Verwaltung von Datenpipelines verwendet (z. B. für ETL-Prozesse), aber auch für das Monitoring solcher Pipelines sowie für komplexe Datenverarbeitungsaufgaben.^[20]

Ein Beispiel zeigt den Einsatz von Airflow zur Verwaltung einer Web-Scraping-Pipeline für *MS-MEs* (*Micro-, Small- and Medium Enterprises*): Hierbei wurde eine automatisierte Datenextraktion implementiert.^[21] Diese Pipeline sammelt Daten aus verschiedenen Quellen, bereinigt sie und speichert sie in einem Data Warehouse, um eine bessere Entscheidungsfindung zu ermöglichen.

Ein weiterer häufiger Anwendungsfall von Apache Airflow ist die Orchestrierung von ETL-Prozessen (Extrahieren, Transformieren, Laden), bei denen Daten aus verschiedenen Quellen wie APIs, Datenbanken oder Cloud-Speichern extrahiert, transformiert und in ein Data Warehouse geladen werden. Dies erleichtert die effiziente Aufbereitung von Daten für Analysen und Berichte.^[22]

Zudem wird Airflow oft für maschinelles Lernen und die Automatisierung von Dashboards und Berichten genutzt. Es ermöglicht die Erstellung von Workflows, die Trainingsdaten sammeln, Modelle trainieren und bereitstellen sowie regelmäßige Berichte generieren oder Dashboards mit aktuellen Daten versorgen. Zusätzlich können Benachrichtigungen bei Anomalien oder Fehlern in den Daten automatisiert versendet werden.^[23]

¹⁸Vgl. Nara/Shaiikh/Pradhan 2023.

¹⁹Vgl. Nara/Shaiikh/Pradhan 2023.

²⁰Vgl. Yasmin u. a. 2024.

²¹Vgl. Riyadi u. a. 2024.

²²Vgl. Intorf/Storey/van Doorn 2024.

²³Vgl. Yasmin u. a. 2024.

2 Installation und Umsetzung

Zur Umsetzung eines Beispiels, wurde die offizielle Dokumentation von Apache genutzt.²⁴ Die Installation von Apache Airflow zur Orchestrierung eines Workflows erfolgt in mehreren Schritten, die nachfolgend detailliert beschrieben werden.

2.1 Vorbereitung

Zunächst wird das Projekt-Repository auf den lokalen Rechner geklont, um die benötigten Dateien und Verzeichnisse bereitzustellen. Dies geschieht über den Befehl:

```
git clone <repository-url>
```

gefolgt von einem Wechsel in das Projektverzeichnis mittels:

```
cd <repository-name>
```

Anschließend werden die erforderlichen Python-Bibliotheken installiert, die in der Datei `requirements.txt` definiert sind. Dies erfolgt mit dem Befehl:

```
pip install -r requirements.txt
```

2.2 Initialisierung der Airflow-Datenbank

Nach der Vorbereitung der Abhängigkeiten wird die Airflow-Datenbank initialisiert. Dieser Schritt umfasst die Einrichtung einer Datenbank sowie die Erstellung eines Admin-Benutzers. Die Initialisierung erfolgt durch den Befehl:

```
airflow db init
```

während der Benutzer mit folgendem Befehl angelegt wird:

```
airflow users create
```

Dabei werden Parameter wie Benutzername, Vorname, Nachname, Rolle und E-Mail-Adresse angegeben, um die Konfiguration abzuschließen.

²⁴Vgl. Apache Software Foundation [2025](#).

2.3 Einrichtung der Datenbank

Ein weiterer zentraler Schritt ist die Einrichtung der Datenbank. Das Projekt verwendet PostgreSQL, kann aber auch mit anderen Datenbanken wie MySQL oder SQLite verwendet werden. Die Konfiguration kann in der Datei `airflow.cfg` geändert werden.

Zur Einrichtung der Datenbank wird ein SQL-Skript verwendet, das Tabellen und Benutzer anlegt. Dieses Skript wird über den Befehl:

```
psql -U postgres -f scripts/setup_postgres.sql
```

ausgeführt.

Optional können Docker-Container verwendet werden, um die Airflow- und PostgreSQL-Komponenten zu orchestrieren. Die Container werden mit dem Befehl:

```
docker-compose up -d
```

gestartet, wobei die Airflow-Weboberfläche anschließend unter <http://localhost:8080> zugänglich ist.

2.4 Starten von Airflow

Im nächsten Schritt wird Airflow selbst gestartet. Dazu werden sowohl der Webserver als auch der Scheduler aktiviert. Der Webserver wird mit dem Befehl:

```
airflow webserver --port 8080
```

gestartet, während der Scheduler mit:

```
airflow scheduler
```

ausgeführt wird. Nach dem Start von Airflow können die definierten DAGs (Directed Acyclic Graphs) in der Benutzeroberfläche aktiviert werden. In diesem Projekt sind dies die DAGs `example_dag` und `aggregate_data_dag`, die nach Aktivierung manuell gestartet werden können.

2.5 Überprüfung des Workflows

Die Ergebnisse des Workflows können schließlich direkt in der PostgreSQL-Datenbank überprüft werden. Dazu werden SQL-Abfragen ausgeführt, um die generierten Tabellen einzusehen. Beispielsweise kann mit dem Befehl:

```
psql -U airflow -d airflow -c "SELECT * FROM example_table;"
```

der Inhalt einer Tabelle abgefragt werden.

Für die Konfiguration ist es wichtig zu beachten, dass die Zugangsdaten zur PostgreSQL-Datenbank sowohl in der Datei `setup_postgres.sql` als auch in der Konfigurationsdatei von Airflow (`airflow.cfg`) definiert sind. Letztere enthält den Eintrag `sql_alchemy_conn`, über den die Verbindung zur Datenbank hergestellt wird.

Im Rahmen des Workflows werden öffentlich verfügbare Daten durch SQL-Skripte verarbeitet, die im Verzeichnis `dags/sql/` abgelegt sind. Diese Skripte dienen dazu, Daten zu erstellen, einzufügen und zu aggregieren und werden automatisch durch die definierten DAGs ausgeführt.

3 Codebeispiel

Nachdem Apache Airflow vorgestellt wurde und die Installation sowie Ausführung des Codes erfolgt ist, sollen im nächsten Schritt die einzelnen Codebestandteile erläutert werden.

3.1 Einführung in den Workflow

Der Code der Datei `example_dag.py` definiert einen einfachen Workflow in Apache Airflow, der die Ausführung eines SQL-Skripts in einer PostgreSQL-Datenbank automatisiert. Der Workflow wird als Directed Acyclic Graph (DAG) beschrieben, was bedeutet, dass er aus einer Reihe von Aufgaben besteht, die in einer bestimmten Reihenfolge ausgeführt werden und keine zyklischen Abhängigkeiten aufweisen.

```
dags > example_dag.py > ...
1  from airflow import DAG
2  from airflow.providers.postgres.operators.postgres import PostgresOperator
3  from datetime import datetime
4
5  # Standardargumente für den DAG
6  default_args = {
7      'owner': 'airflow', # Besitzer des DAGs
8      'start_date': datetime(2025, 1, 8), # Startdatum des DAGs
9      'retries': 1, # Anzahl der Wiederholungsversuche bei Fehlern
10 }
11
12 # Definition des DAGs
13 with DAG(
14     dag_id='example_postgres_dag', # ID des DAGs
15     default_args=default_args, # Standardargumente
16     schedule='@daily', # Zeitplan für die Ausführung (täglich)
17     catchup=False, # Keine nachträgliche Ausführung verpasster Intervalle
18 ) as dag:
19
20     # Task: SQL-Skript ausführen
21     execute_sql = PostgresOperator(
22         task_id='execute_sql_script', # ID des Tasks
23         postgres_conn_id='postgres_default', # Verbindung zu PostgreSQL
24         sql='sql/create_and_insert.sql', # Pfad zum auszuführenden SQL-Skript
25     )
26
```

Abb. 5: Codeausschnitt von `example_dag.py`.

3.2 Definition des DAGs

Zu Beginn wird ein Wörterbuch namens `default_args` definiert, das Standardparameter für den DAG enthält. Diese Parameter umfassen den Besitzer des Workflows (`owner`), das Startdatum (`start_date`) und die Anzahl der Wiederholungsversuche (`retries`), falls eine Aufgabe

fehlschlägt. Das Startdatum ist hier auf den 8. Januar 2025 gesetzt. Diese Argumente werden verwendet, um allgemeine Einstellungen für alle Aufgaben im DAG zu definieren.

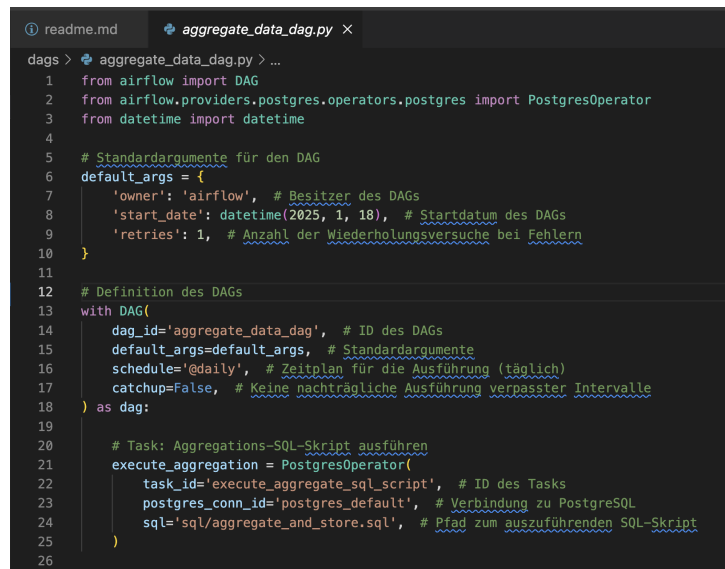
Der DAG selbst wird mithilfe des DAG-Objekts erstellt und mit der ID `example_postgres_dag` benannt. Der Parameter `schedule='@daily'` gibt an, dass der Workflow täglich ausgeführt werden soll. Mit `catchup=False` wird festgelegt, dass keine rückwirkenden Ausführungen für vergangene Zeitpunkte durchgeführt werden sollen.

Innerhalb des DAGs wird eine Aufgabe namens `execute_sql` definiert, die ein SQL-Skript ausführt. Diese Aufgabe verwendet den `PostgresOperator`, einen vorgefertigten Operator in Airflow, der speziell für die Interaktion mit PostgreSQL-Datenbanken entwickelt wurde. Der Operator benötigt drei wesentliche Parameter:

- `task_id`: Ein eindeutiger Bezeichner für die Aufgabe, hier `execute_sql_script`.
- `postgres_conn_id`: Die ID der PostgreSQL-Verbindung, die in Airflow konfiguriert ist (hier `postgres_default`).
- `sql`: Der Pfad zum SQL-Skript, das ausgeführt werden soll (hier: `sql/create_and_insert.sql`).

Das SQL-Skript wird ausgeführt, sobald der DAG aktiviert und geplant wird. Der gesamte Workflow ist so konzipiert, dass er einfach zu verstehen und zu erweitern ist.

Im Beispielprojekt gibt es noch eine weitere DAG-Datei (`aggregate_sql_dag.py`), die ähnlich aufgebaut ist wie die beschriebene Datei, jedoch andere Werte für die Parameter enthält.



```
1 from airflow import DAG
2 from airflow.providers.postgres.operators.postgres import PostgresOperator
3 from datetime import datetime
4
5 # Standardargumente für den DAG
6 default_args = {
7     'owner': 'airflow', # Besitzer des DAGs
8     'start_date': datetime(2025, 1, 18), # Startdatum des DAGs
9     'retries': 1, # Anzahl der Wiederholungsversuche bei Fehlern
10 }
11
12 # Definition des DAGs
13 with DAG(
14     dag_id='aggregate_data_dag', # ID des DAGs
15     default_args=default_args, # Standardargumente
16     schedule='@daily', # Zeitplan für die Ausführung (täglich)
17     catchup=False, # Keine nachträgliche Ausführung verpasster Intervalle
18 ) as dag:
19
20     # Task: Aggregations-SQL-Skript ausführen
21     execute_aggregation = PostgresOperator(
22         task_id='execute_aggregate_sql_script', # ID des Tasks
23         postgres_conn_id='postgres_default', # Verbindung zu PostgreSQL
24         sql='sql/aggregate_and_store.sql', # Pfad zum auszuführenden SQL-Skript
25     )
26
```

Abb. 6: Codeausschnitt von `aggregate_data_dag.py`.

3.3 Die SQL-Skripte

Durch die Datei `example_dag.py` wird ein SQL-Skript ausgeführt, welches zwei Befehle enthält:

```

dags > sql > create_and_insert.sql
1 | -- Erstellen einer neuen Tabelle namens example_table, falls sie nicht bereits existiert
2 | CREATE TABLE IF NOT EXISTS example_table (
3 |     id SERIAL PRIMARY KEY, -- Primärschlüssel, der automatisch inkrementiert wird
4 |     name VARCHAR(50),      -- Spalte für Namen mit einer maximalen Länge von 50 Zeichen
5 |     value INT              -- Spalte für Werte vom Typ Integer
6 | );
7 |
8 | -- Einfügen von Beispiel-Daten in die Tabelle example_table
9 | INSERT INTO example_table (name, value)
10 | VALUES
11 |     ('Test1', 100), -- Einfügen eines Datensatzes mit dem Namen 'Test1' und dem Wert 100
12 |     ('Test2', 200); -- Einfügen eines Datensatzes mit dem Namen 'Test2' und dem Wert 200
13 |

```

Abb. 7: Codeausschnitt von `create_and_insert.sql`.

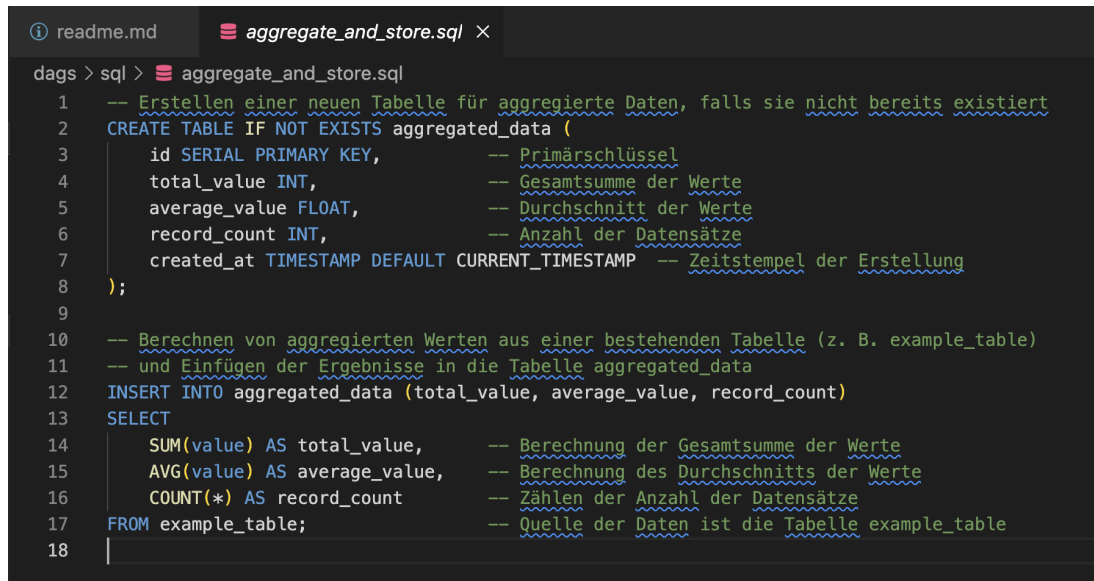
- **CREATE TABLE:-Befehl**

Der erste Befehl, `CREATE TABLE IF NOT EXISTS`, definiert eine Tabelle namens `example_table`. Dabei wird sichergestellt, dass die Tabelle nur erstellt wird, falls sie noch nicht existiert. Die Tabelle enthält drei Spalten: - `**id**`: Ein automatisch inkrementierender Primärschlüssel (`SERIAL PRIMARY KEY`), der für jede Zeile einen eindeutigen Wert generiert. - `**name**`: Ein Textfeld mit einer maximalen Länge von 50 Zeichen (`VARCHAR(50)`). - `**value**`: Eine Ganzzahl (`INT`) zur Speicherung numerischer Werte.

- **INSERT INTO-Befehl**

Der zweite Befehl, `INSERT INTO`, fügt zwei Datensätze in die Tabelle ein. Die Werte für die Spalten `**name**` und `**value**` werden explizit angegeben, während die Spalte `**id**` automatisch durch die Datenbank generiert wird. Die eingefügten Datensätze haben die Werte `('Test1', 100)` und `('Test2', 200)`. Diese Befehle zusammen bilden eine einfache Grundlage für das Arbeiten mit relationalen Datenbanken: Sie definieren zunächst die Struktur der Daten und fügen anschließend Beispielwerte ein.

Darüber hinaus existiert ein weiteres SQL-Skript, das der Aggregation und Speicherung von Daten dient. Da es sich hierbei um ein einfaches SQL-Skript handelt, wird auf eine detaillierte Erläuterung verzichtet.



```

dags > sql > aggregate_and_store.sql
1  -- Erstellen einer neuen Tabelle für aggregierte Daten, falls sie nicht bereits existiert
2  CREATE TABLE IF NOT EXISTS aggregated_data (
3      id SERIAL PRIMARY KEY,          -- Primärschlüssel
4      total_value INT,                -- Gesamtsumme der Werte
5      average_value FLOAT,            -- Durchschnitt der Werte
6      record_count INT,               -- Anzahl der Datensätze
7      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Zeitstempel der Erstellung
8  );
9
10 -- Berechnen von aggregierten Werten aus einer bestehenden Tabelle (z. B. example_table)
11 -- und Einfügen der Ergebnisse in die Tabelle aggregated_data
12 INSERT INTO aggregated_data (total_value, average_value, record_count)
13 SELECT
14     SUM(value) AS total_value,        -- Berechnung der Gesamtsumme der Werte
15     AVG(value) AS average_value,      -- Berechnung des Durchschnitts der Werte
16     COUNT(*) AS record_count         -- Zählen der Anzahl der Datensätze
17 FROM example_table;                 -- Quelle der Daten ist die Tabelle example_table
18

```

Abb. 8: Codeausschnitt von aggregate_and_store.sql.

Literaturverzeichnis

- Apache Software Foundation (2025):** Apache Airflow Documentation. Accessed: 2025-01-19. URL: <https://airflow.apache.org/docs/>.
- Haines, S. (2022):** Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications. New York, NY: Apress. ISBN: 978-1-4842-7452-1. URL: <https://link.springer.com/book/10.1007/978-1-4842-7452-1>.
- Harenslak, B./de Ruiter, J. R. (2021):** Data Pipelines with Apache Airflow. For more information, visit <https://www.manning.com/>. Shelter Island, NY, USA: Manning Publications Co. ISBN: 9781617296901.
- Intorf, D./Storey, D./van Doorn, K. (2024):** Apache Airflow Best Practices: A Practical Guide to Orchestrating Data Workflow with Apache Airflow. Birmingham, UK: Packt Publishing Ltd, S. 188. ISBN: 9781805129332.
- Nara, M./Shaikh, A./Pradhan, R. (2023):** Managing Data Pipeline with Apache Airflow. In: *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)* 3.2, S. 1–5. URL: <https://ijarsct.co.in/A12134.pdf>.
- Riyadi, M. u. a. (2024):** Big Data Pipeline Infrastructure Design in MSME E-Commerce Systems with a Focus on Data Source Processing Using Orchestration Tools. In: *Transmisi: Jurnal Ilmiah Teknik Elektro*. URL: <https://ejournal.undip.ac.id/index.php/transmisi/article/view/59003>.
- Yasmin, J./Wang, J. A./Tian, Y./Adams, B. (2024):** An Empirical Study of Developers' Challenges in Implementing Workflows as Code: A Case Study on Apache Airflow. In: *Journal of Systems and Software* 219, S. 112248. DOI: [10.1016/j.jss.2024.112248](https://doi.org/10.1016/j.jss.2024.112248).

Erklärung zur Verwendung generativer KI-Systeme

Bei der Erstellung der eingereichten Arbeit habe ich die nachfolgend aufgeführten auf künstlicher Intelligenz (KI) basierten Systeme benutzt:

1. Perplexity
2. DeepL

Ich erkläre, dass ich

- mich aktiv über die Leistungsfähigkeit und Beschränkungen der oben genannten KI-Systeme informiert habe.²⁵
- die aus den oben angegebenen KI-Systemen direkt oder sinngemäß übernommenen Passagen gekennzeichnet habe,
- überprüft habe, dass die mithilfe der oben genannten KI-Systeme generierten und von mir übernommenen Inhalte faktisch richtig sind,
- mir bewusst bin, dass ich als Autorin bzw. Autor dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Die oben genannten KI-Systeme habe ich wie im Folgenden dargestellt eingesetzt:

Arbeitsschritt in der wissenschaftlichen Arbeit	Eingesetzte(s) KI-System(e)	Beschreibung der Verwendungsweise
Erklärung und Verständnisfragen	Perplexity	Erklärungen
Literatursuche	DeepL	Übersetzungen
Finalisierung	DeepL	Rechtschreibkorrektur

²⁵U.a. gilt es hierbei zu beachten, dass an KI weitergegebene Inhalte ggf. als Trainingsdaten genutzt und wiederverwendet werden. Dies ist insb. für betriebliche Aspekte als kritisch einzustufen.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Orchestrierung mittels Airflow* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum)

stuttgart, 19.01.2025

(Unterschrift)

Mehmet Karaca