ETL Strecke vom DWH zum FHIR Server mit Dagster

Eine Einführung

Ralf Lützkendorf

www.med.uni-magdeburg.de

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

UNIVERSITÄTSMEDIZIN MAGDEBURG

Kolloquium
Medizininformatik für Forschung und Versorgung

DIZ Daten Integrations Zentrum

Inhalt

- Einführung: Daten- und Workflow-Management
- Entwicklung der Orchestrierungs-Tools
- Airflow vs. Dagster
- Dagster im Detail: Architektur und Features
- Dagster: Mini Tutorial
- Fazit und Ausblick

Datenmanagement

DATEN

- Aktivitäten
- Prozesse
- Technologien

- Erfassung
- Speicherung
- Organisation
- Integration
- Bereinigung
- Sicherung
- Bereitstellung

Aufgabenbereiche

- Datenintegration
- Datenqualität
- Datenmodellierung
- Datenbankmanagement
- Datensicherheit
- Datenarchivierung

Workflowmanagment

Aufgaben

- Systematische Planung
- Koordination
- Ausführung

- Bestimmte Reihenfolge
- Bestimmte Ziele

Aufgabenbereiche

- Aufgabenplanung
- Ressourcenmanagement
- Abhängigkeitsmanagement
- Überwachung und Steuerung
- Automatisierung
- Reporting

Software

Daten- und Workflow-Management

- Verbindung von Daten- und Workflow-Management
- Ziele der Verbindung
- Besondere Anwendungsfälle
- Klare Strukturen
- Schlüsselwerkzeuge

Nutzen

- Workflow-Definition
- Scheduling
- Monitoring
- Fehlerbehandlung
- Wiederaufnahme
- Skalierung
- Flexibilität

Privat

- Backup
- Privates Daten sammeln (Fotos, Finanzen, Gesundheit Familie,…)
- Automatisierung im Haushalt/Smart Home/IoT Geräte
- Hobby Projekte (Routine Aufgaben)

DWH zum FHIR

Lesen/Schreiben der Staging Datenbank

Lesen des DWH

Datenanalyse

Delta Load

Data Cleaning

FHIR Transformation

Spezifikation für Mapping

Lesen/Schreibe des FHIR Server

Open Source Top 8 Tools

1. **Luigi:**
2. **Prefect:**
3. **Apache NiFi:**
4. **Celery:**
5. **Taskflow:**
6. **Kubeflow Pipelines:**
7. **Oozie:**
8. **Jenkins:**

Apache
Airflow

Airflow Überblick

- Seit 2014 (Airbnb)
- 2015 unter Apache Lizenz 2.0
- 2016 Top Level Projekt Apache Software Foundation
- 2018 Airflow 1.10
- 2020 Airflow 2.0
- Aktuell 2.10.2 (September 2024)

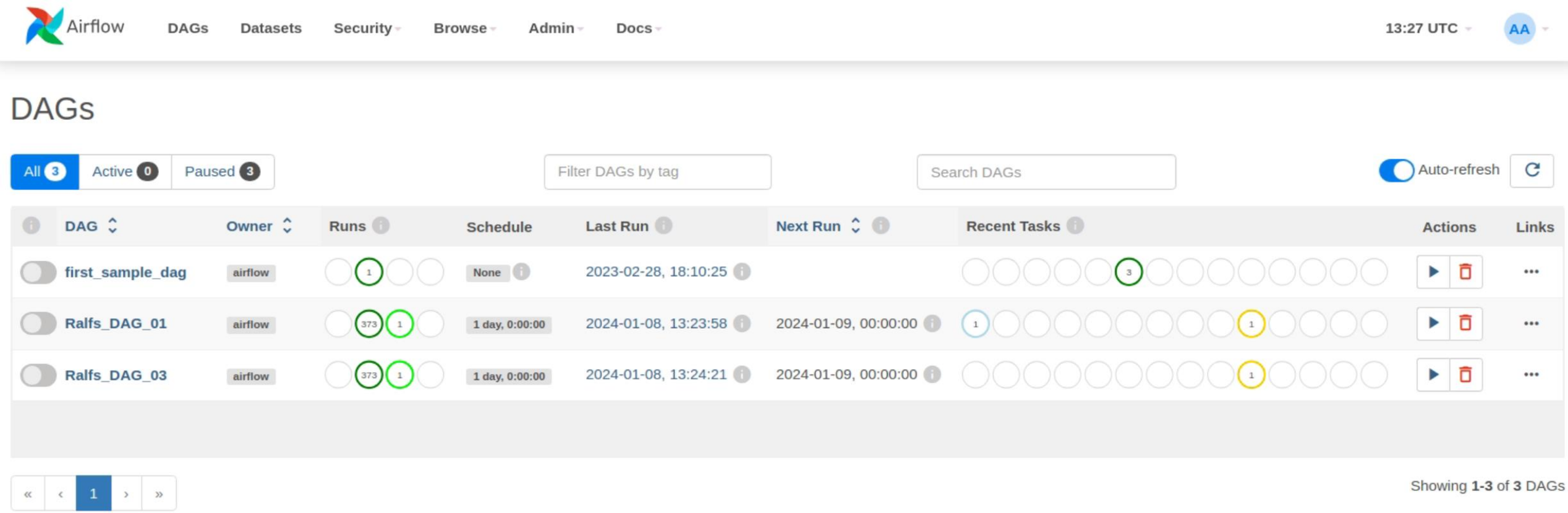
Apache Airflow Releases

- Airflow 2.9.1 (2024-05-03)
- Airflow 2.9.0 (2024-04-08)
- Airflow 2.8.4 (2024-03-25)
- Airflow 2.8.3 (2024-03-11)
- Airflow 2.8.2 (2024-02-26)
- Airflow 2.8.1 (2024-01-19)
- Airflow 2.8.0 (2023-12-18)
- Airflow 2.7.3 (2023-11-06)
- Airflow 2.7.2 (2023-10-12)
- Airflow 2.7.1 (2023-09-07)
- Airflow 2.7.0 (2023-08-18)
- Airflow 2.6.3 (2023-07-10)
- Airflow 2.6.2 (2023-06-17)
- Airflow 2.6.1 (2023-05-16)
- Airflow 2.6.0 (2023-04-30)
- Airflow 2.5.3 (2023-04-01)
- Airflow 2.5.2 (2023-03-15)
- Airflow 2.5.1 (2023-01-20)
- Airflow 2.5.0 (2022-12-02)
- Airflow 2.4.3 (2022-11-14)
- Airflow 2.4.2 (2022-10-23)
- Airflow 2.4.1 (2022-09-30)
- Airflow 2.4.0 (2022-09-19)
- Airflow 2.3.4 (2022-08-23)

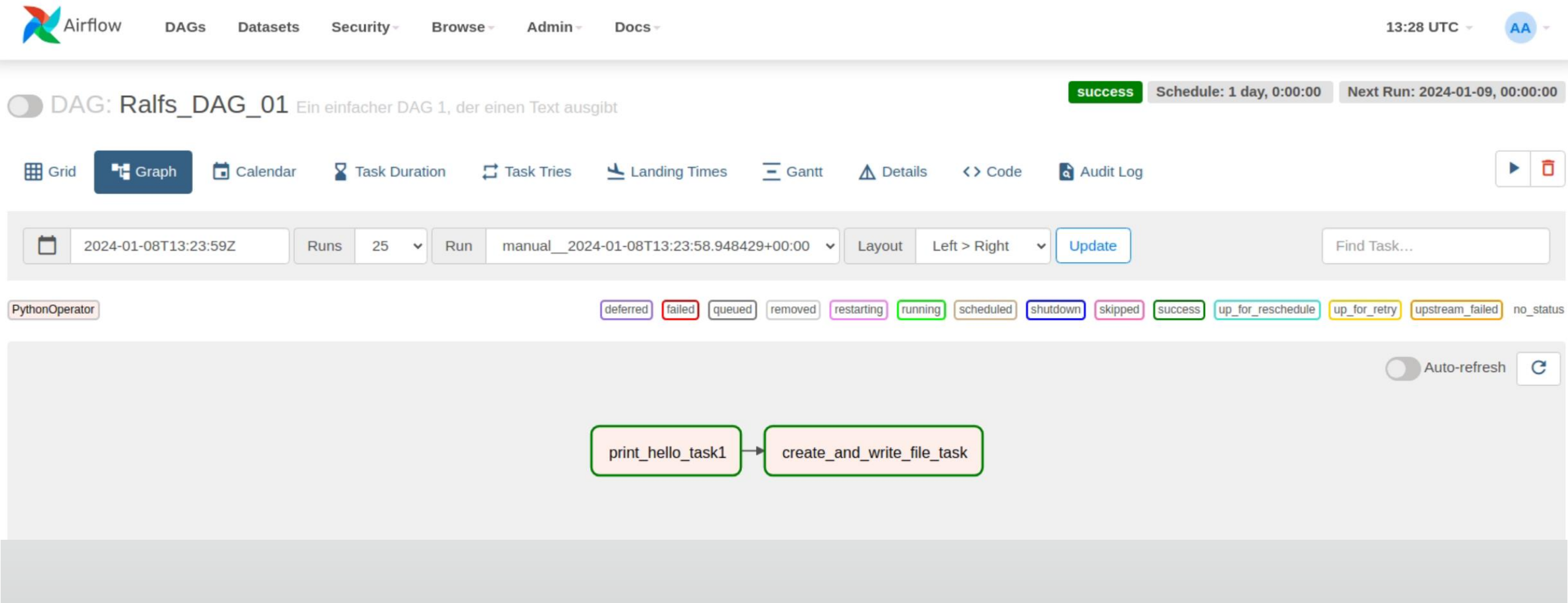
Airflow Überblick

1.Airbnb: Wie bereits erwähnt, wurde Airflow ursprünglich von Airbnb entwickelt und ist integraler Bestandteil ihrer Dateninfrastruktur.
2.Adobe: Nutzt Airflow zur Automatisierung von Datenworkflows in verschiedenen Geschäftsbereichen.
3.Twitter(X): Setzt Airflow ein, um komplexe Datenpipelines zu managen, die große Mengen an Tweet-Daten verarbeiten.

Airflow Überblick



Airflow Überblick




dagster

Dagster Überblick

- Seit 2019
- 2020 extreme Verbesserungen und Weiterentwicklungen
- Aktuelle Version 1.7.4 / 0.21.14 libraries (Mai 2024)

Version	Datum der Veröffentlichung
1.7.4 (core) / 0.23.4 (libraries)	2024-04-04
1.7.3 (core) / 0.23.3 (libraries)	2024-03-27
1.7.2 (core) / 0.23.2 (libraries)	2024-03-20
1.7.1 (core) / 0.23.1 (libraries)	2024-03-13
1.7.0 (core) / 0.23.0 (libraries)	2024-04-04
1.6.14 (core) / 0.22.14 (libraries)	2024-02-29
1.6.13 (core) / 0.22.13 (libraries)	2024-02-22
1.6.12 (core) / 0.22.12 (libraries)	2024-02-15
1.6.11 (core) / 0.22.11 (libraries)	2024-02-08
1.6.10 (core) / 0.22.10 (libraries)	2024-02-01

In Google Tabellen exportieren

Dagster Überblick

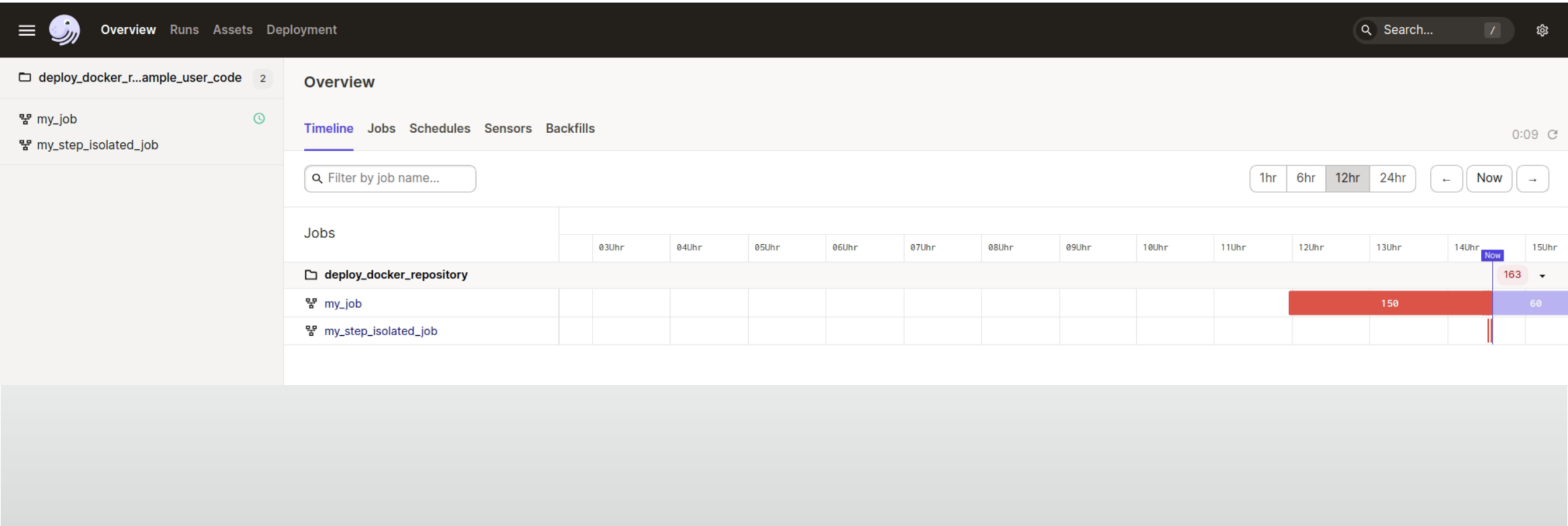
Unternehmen, die Dagster nutzen

1.Elementl: Das Unternehmen hinter Dagster nutzt natürlich seine eigene Software zur Datenpipeline-Orchestrierung.

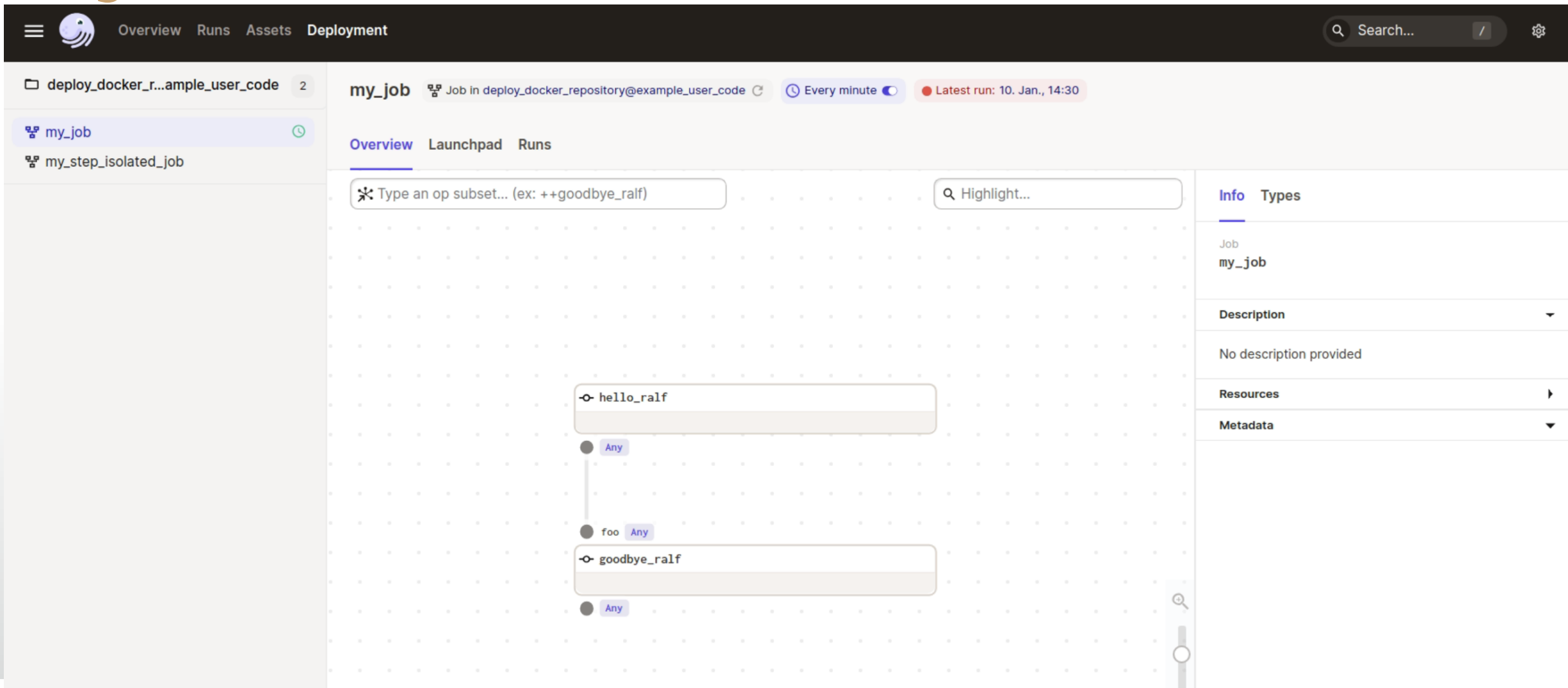
2.GoodRx: Nutzt Dagster, um ihre Datenpipelines zu managen, was entscheidend ist, um erschwingliche Medikamentenpreise anzubieten.

3.Wolt: Dieses Lieferdienst-Unternehmen verwendet Dagster zur Verbesserung ihrer Datenverarbeitungsprozesse, was ihnen hilft, Lieferungen effizienter zu gestalten.

Dagster Überblick



Dagster Überblick



Dagster versus Airflow

Vergleich

- **Projektanforderungen:**
  - Spezifische Anforderungen des Projekts
- **Präferenzen:**
  - Persönliche und Teampräferenzen
- **Infrastruktur:**
  - Integration mit vorhandener Infrastruktur
- **Community und Ressourcen:**
  - Community-Support und Ressourcenverfügbarkeit
- **Lernkurve:**
  - Lernkurve für das Team
- **Flexibilität:**
  - Flexibilität bei der Definition von Workflows
- **Erweiterbarkeit und Integration:**
  - Erweiterbarkeit und Integration mit anderen Tools

Vergleich – Apache Airflow

Vorteile:

- **Community-Support:**
  - Breite Community und weit verbreitet in vielen Unternehmen.
  - Zugriff auf eine reiche Ressourcenbasis für Support und Erweiterungen.
- **Flexibilität:**
  - Flexibilität bei der Definition von DAGs und Tasks.
  - Gut geeignet für verschiedene Anforderungen in der Datenverarbeitung.
- **Scheduler:**
  - Der Scheduler ermöglicht die zeitgesteuerte Ausführung von Aufgaben.
  - Nützlich für periodische ETL-Jobs und zeitkritische Workloads.
- **Erweiterbarkeit:**
  - Airflow ist erweiterbar und unterstützt die Integration von Plugins.
  - Unterstützt Integrationen für verschiedene Anwendungen und Plattformen.

Nachteile:

- **Etwas mehr Boilerplate-Code:**
  - Im Vergleich zu Dagster - Airflow etwas mehr Boilerplate-Code
  - Höherer Initialisierungsaufwand

Vergleich - Dagster

Vorteile:

- **Datenqualität:**
  - Starker Fokus auf Überwachung und Sicherung der Datenqualität.
  - Besonders relevant in medizinischen Kontexten.
- **Declarative Definition:**
  - Datenpipelines werden deklarativ definiert.
  - Verbessert Lesbarkeit und Wartbarkeit des Codes.
- **Assets und Metadaten-Management:**
  - Ermöglicht das Management von Assets.
  - Nützlich für die Verwaltung von medizinischen Daten und deren Metadaten.
- **Integration mit Analyse-Tools:**
  - Geeignet für Datenanalyse und Machine Learning.
  - Gute Integration mit verschiedenen Analyse-Tools.

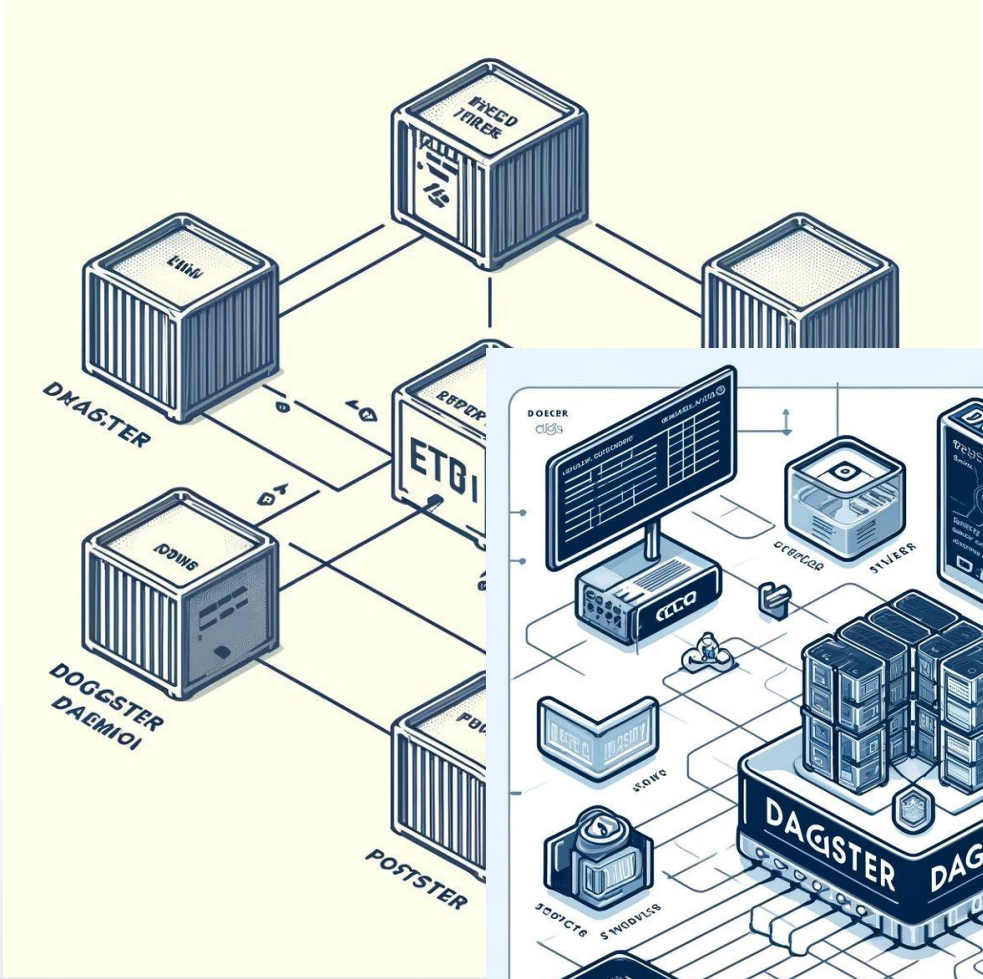
Nachteile:

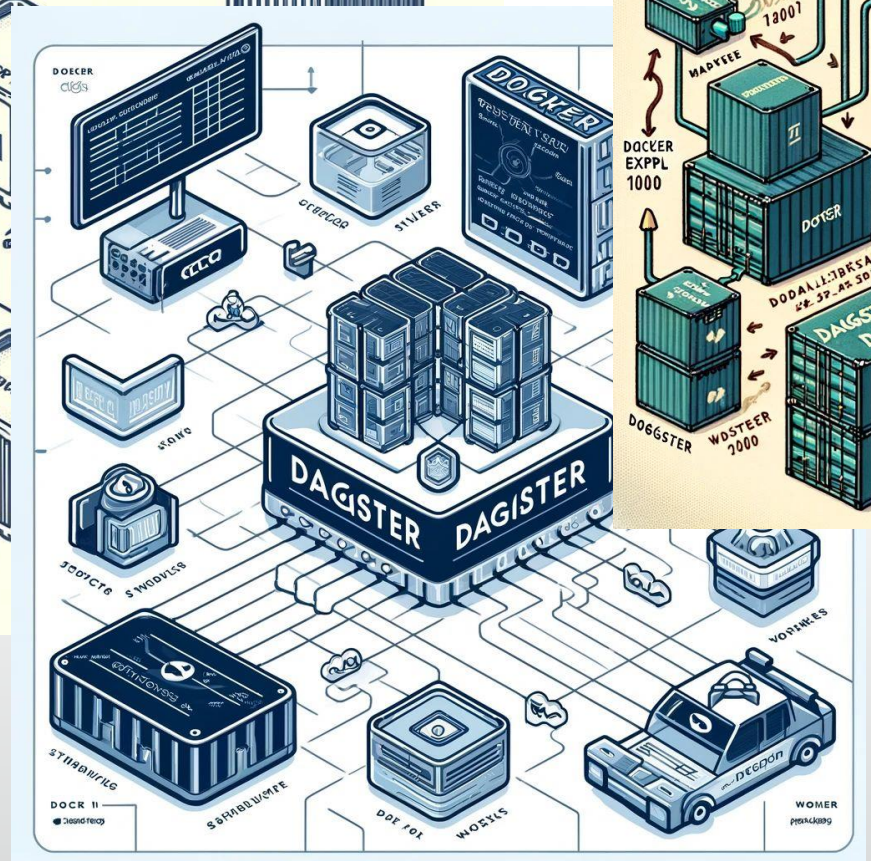
- **Lernkurve:**
  - Das Konzept von Jobs und Assets erfordert möglicherweise eine gewisse Einarbeitungszeit.
  - Kann für Programmieranfänger hohe Einstiegshürde.

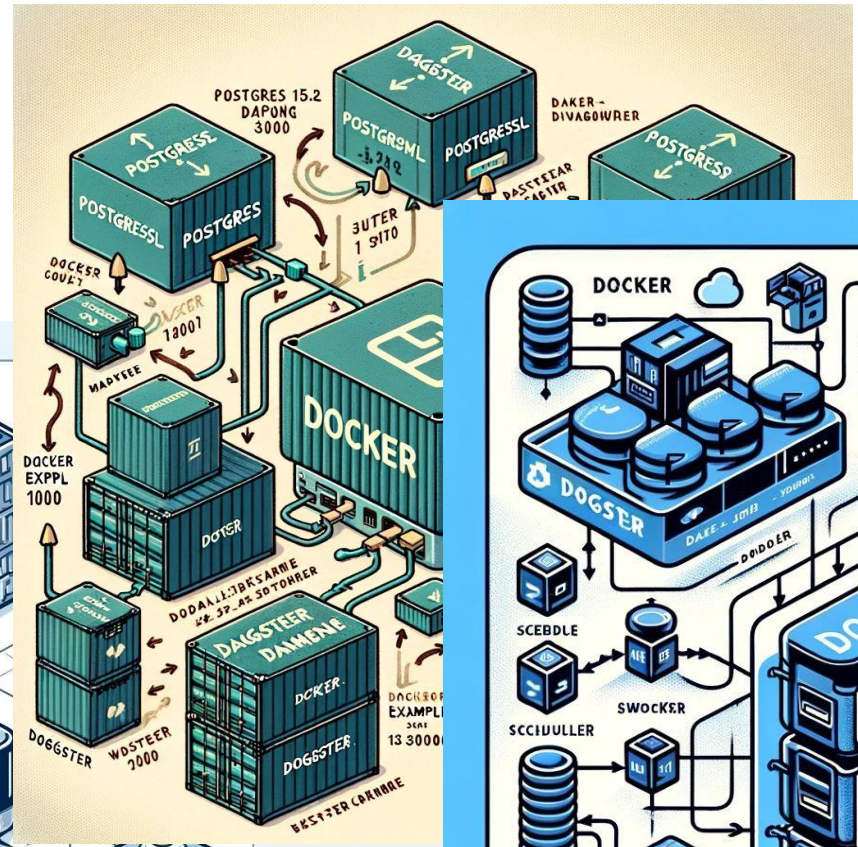
Vergleich

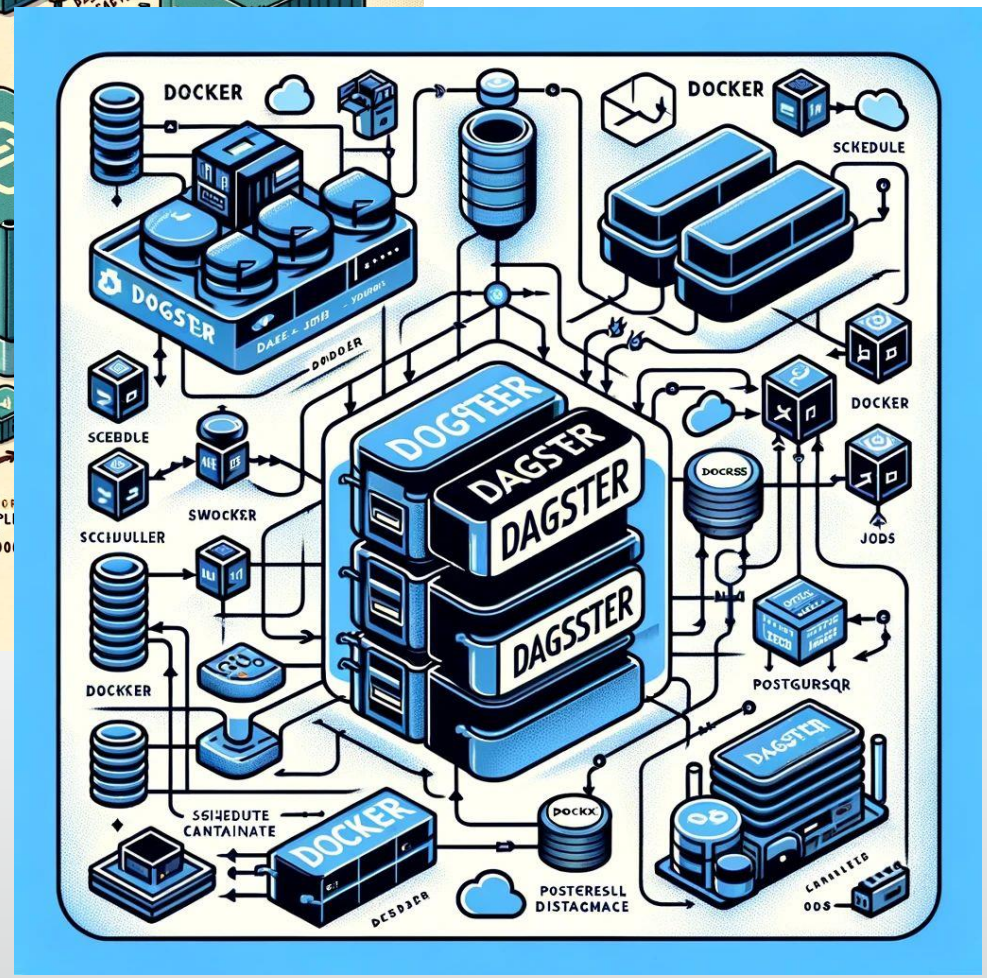
- **Dagster:**
  - Überwachung der Datenqualität.
  - Assets-Management.
  - Deklarative Definition von Datenpipelines.
- **Airflow:**
  - Breite Community-Unterstützung.
  - Flexibilität bei DAG- und Task-Definitionen.
  - Integration mit anderen Tools.

dagster


DAGSTER
DOGGSTER
DAEMON
POSTGRES

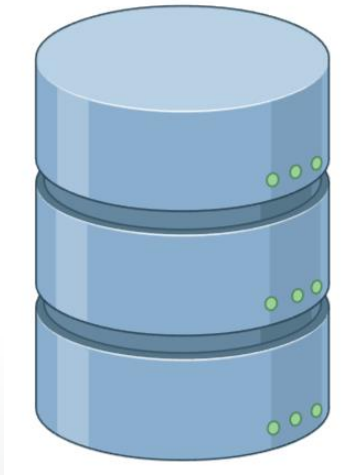

DOCKER
DAGSTER
DAGSTER
DAGSTER


POSTGRES 15.2
DAGSTER
POSTGRESQL
POSTGRES
DOCKER
DOCKER
EXPPL
1000
DAGSTER
DAEMONE
DAGSTER
DOGGSTER


DOCKER
DOCKER
SCHEDULE
DOGSTER
DOGTEER
DAGSTER
DAGSTER
DAGSSTER
DOCKER
SCHEDULE
SWOCKER
SCHEDULLER
DOCKER
JODS
POSTGURSQR
DAGSTER
DOCKER
DOCKER
POSTERESLL
DISTACMACE

DIZ Magdeburg

dagster



Python-slim 3.10

Dagster

Dagster PostgreSQL

Dagster Webserver

Linux – Dagster Mini Tutorial

```
ralf@Linux:~$ pip install dagster
ralf@Linux:~$ pip install dagster-webserver
ralf@Linux:~$ dagster project scaffold --name dwh
ralf@Linux:~$ cd dwh
ralf@Linux:~$ mkdir -p /DATA/repos/dwh/dagster_home
ralf@Linux:~$ export DAGSTER_HOME=$(pwd)/dagster_home
ralf@Linux:~$ DAGSTER_HOME=$(pwd)/dagster_home dagster-webserver -h 0.0.0.0 -p 3000
```

```
dwh/
├── README.md
├── dwh/
│   ├── __init__.py
│   └── assets.py
├── dwh_tests/
├── pyproject.toml
├── setup.cfg
└── setup.py
```

Voraussetzung:

Python 3.8 bis 3.11 mit pip

Linux – Dagster Mini Tutorial

```
from dagster import asset

@asset
def hello_diz():
    return "Hello DIZ"
```

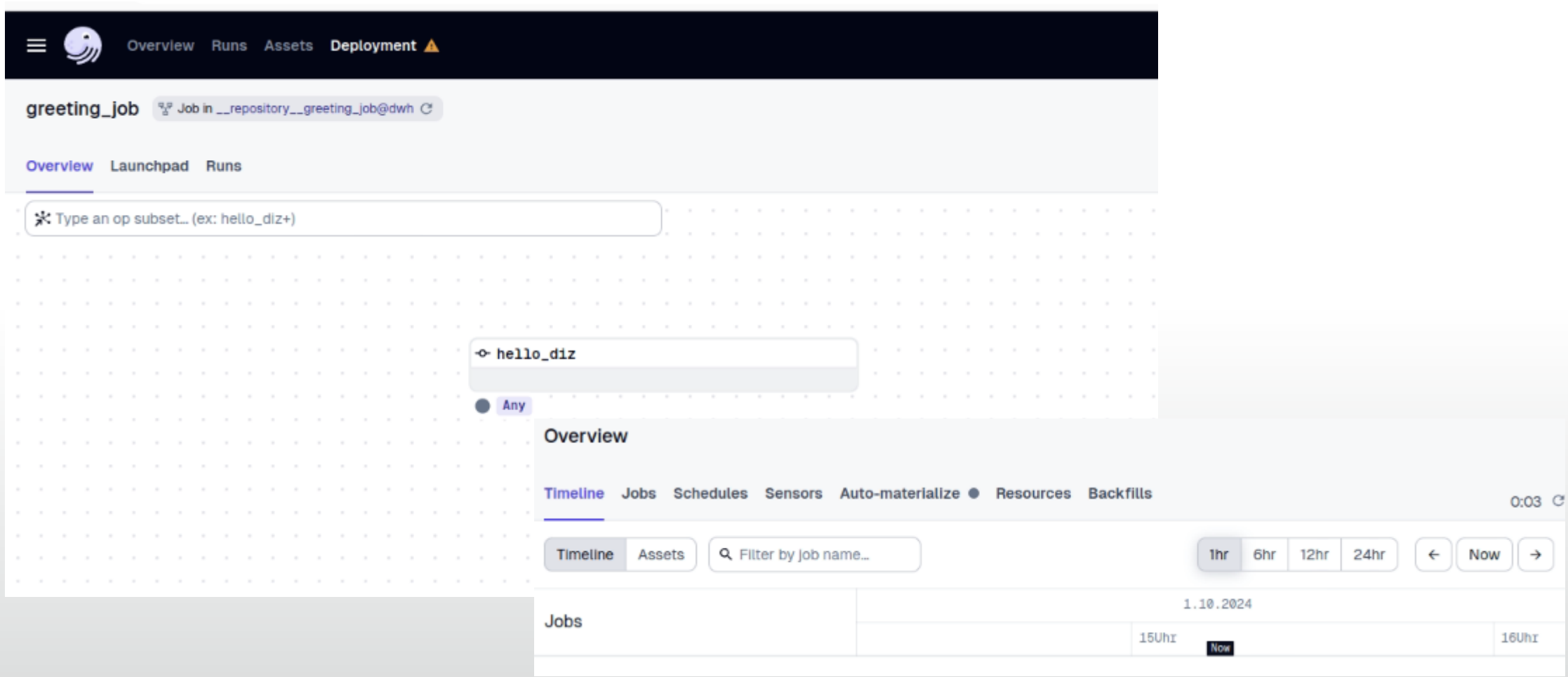
Linux – Dagster Mini Tutorial

```
from dagster import job, schedule
from .assets import my_asset

@job
def my_job(): # pipeline
    asset()  # Asset im Job verwenden

@schedule(cron_schedule="0 * * * *", job=my_job,
execution_timezone="UTC")
def my_hourly_schedule(context):
    context.log.info("Running my hourly job.")
```

Linux – Dagster Mini Tutorial



Linux – Dagster Mini Tutorial

Scheduler

```
from dagster import (
    AssetSelection,
    Definitions,
    ScheduleDefinition,
    define_asset_job,
    load_assets_from_modules,
)

from . import assets

all_assets = load_assets_from_modules([assets])

# Define a job that will materialize the assets
ETL_DWH_Strecke = define_asset_job("ETL_DWH_Strecke", selection=AssetSelection.all())

# Define a ScheduleDefinition for the job with a cron schedule (every 12 hours)
ETL_DWH_Schedule = ScheduleDefinition(
    job=ETL_DWH_Strecke,
    cron_schedule="0 */12 * * *",  # every 12 hours
)

defs = Definitions(
    assets=all_assets,
    schedules=[ETL_DWH_Schedule],
)
```

Linux – Dagster Mini Tutorial

Asset lesen

```
from dagster import asset, MetadataValue
import pandas as pd
from datetime import datetime
from hdbcli import dbapi
import psycopg2


# Tabelle/Cube 1 lesen
@asset(
    name="fetch_from_hana_1",
    description="Lädt CV_UKMLD10_DIAGNOSE von SA
)
def fetch_from_hana_1(context) -> pd.DataFrame:
    conn = None
    try:
        conn = dbapi.connect(
            address="dwh-schnittstellehannah",
            port="30000",
            user="USER",
            password="GeheimesPassword",
            encrypt=False,
            sslValidateCertificate=False
        )
        query1 = 'SELECT * FROM "_SYS_BIC"."DIZ_DATA
        df1 = pd.read_sql(query1, conn)
        # Anzahl der Zeilen als Metadaten hinzufügen
        num_rows = len(df1)
        timestamp = datetime.now().isoformat()

        context.add_output_metadata({
            "Anzahl der Zeilen": MetadataValue.int(num_rows),
            "Zeitstempel": MetadataValue.text(timestamp)
        })

        return df1
    except Exception as e:
        context.log.error(f"Ein Fehler ist aufgetreten: {e}")
        raise e
    finally:
        if conn:
            conn.close()
```

Linux – Dagster Mini Tutorial

Asset schreiben

```
# Tabelle 1 schreiben
@asset(
    name="write_to_postgres_1",
    description="Schreibt CV_UKMLD10_DIAGNOSE aus einem DataFrame in PostgreSQL",
)
def write_to_postgres_1(context, fetch_from_
    conn = None
    try:
        conn = psycopg2.connect(
            host="140.111.111.111"
            port
            user
            pass
            data
        )
        cursor

        table_
```

```
        # Tabelle leeren (Delta Load)
        cursor.execute(f"TRUNCATE T

        columns_with_types = ", ".joi
            f'"{col}" TEXT' for col in fetc
        ])

        create_table_query = f"""
        CREATE TABLE IF NOT EXISTS
            {columns_with_types}
        );
        """
        cursor.execute(create_table_query)
```

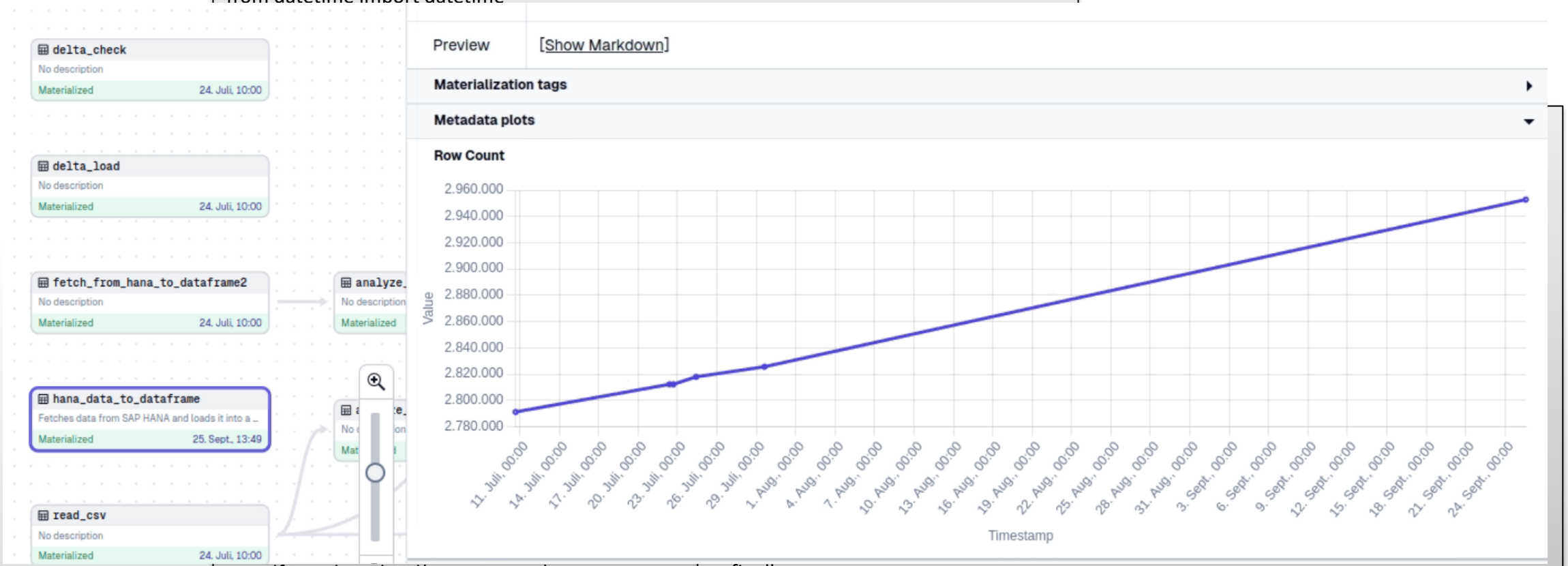
```
# Daten einfügen
        for i, row in fetch_from_hana_1.iterrows():
            insert_query = f"""
            INSERT INTO {table_name} ({', '.join([f'"{col}"' for col in fetch_from_hana_1.columns])})
            VALUES ({', '.join(['%s'] * len(fetch_from_hana_1.columns))});
            """
            cursor.execute(insert_query, tuple(row))

        conn.commit()
    except Exception as e:
        context.log.error(f"Ein Fehler ist aufgetreten: {e}")
        raise e
    finally:
        if conn:
            conn.close()
```

Linux – Dagster Mini Tutorial

Metadaten

```
from dagster import asset, MetadataValue
import pandas as pd
from datetime import datetime
```



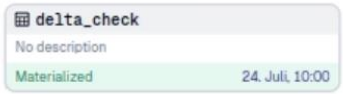

```
    df1 = pd.read_sql(query1, conn)
finally:
    if conn:
        conn.close()
```

Scheduler

Linux – Dagster Mini Tutorial

Markdown

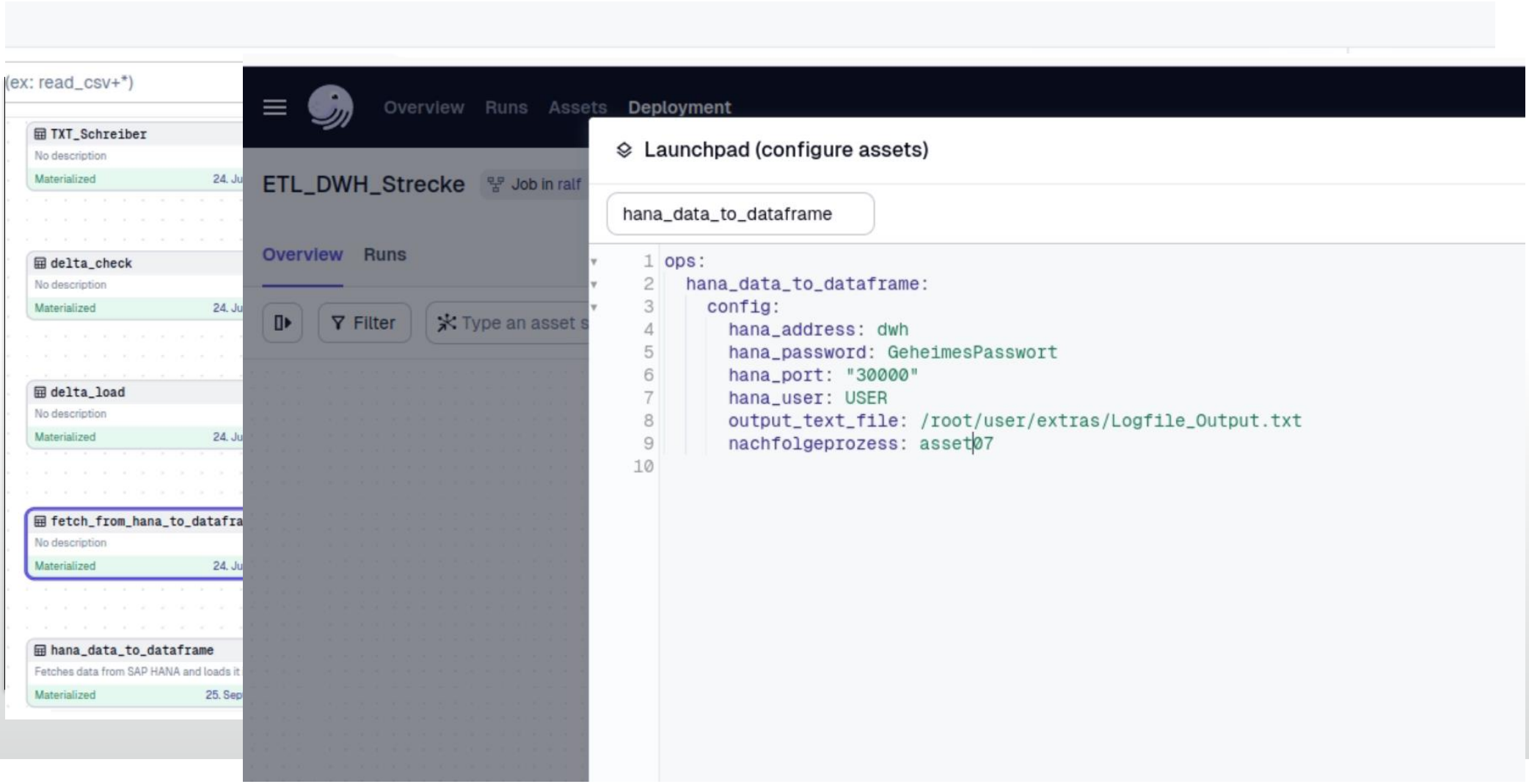
return Materiali
current_time.st



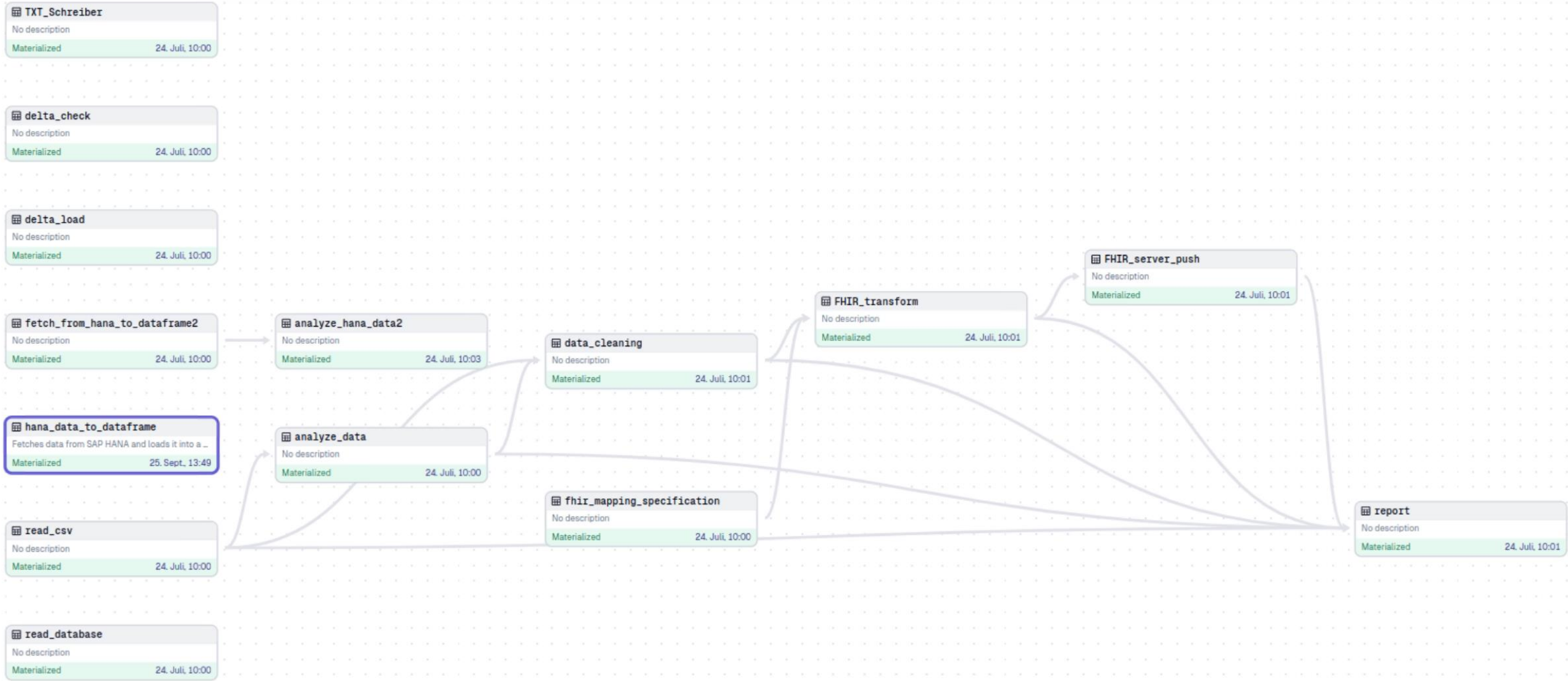
PatientID	DiagnoseCode	DiagnoseBeschreibung	Datum	Arzt
123456	A01.1	Typhus abdominalis	2024-09-30	Dr. Müller
234567	J20.9	Akute Bronchitis	2024-09-29	Dr. Schmidt
345678	E11.9	Diabetes mellitus Typ 2	2024-09-28	Dr. Weber
456789	I10	Essentielle Hypertonie	2024-09-27	Dr. Fischer
567890	M54.5	Kreuzschmerzen	2024-09-26	Dr. Wagner

Linux – Dagster Mini Tutorial

Launchpad

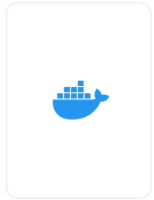


DWH Pipeline



Tools/Plugins/Integrations

1. **Auto-materializing**
2. **Scheduler**
3. **Markdown (Metadaten)**
4. **Launchpad**
5. **Backfill**
6. **Sensors**
7. **User Code „Server“**



Dagster Integration:

Using Dagster with Docker

Run runs

Dagster Integration:

Dagster + Jupyter

s that overlap. Use this chea

agstermill eliminates the tedious "productionization" of Jupyter notebooks.

Directed Acyclic Graphs (DAG)	Jobs	
Task	Ops	
Datasets	Assets	Dagster assets are more powerful and matur include support for things like partitioning.

Dagster Integration:

Using Dagster with Kubernetes

Launch Kubernetes pods and execute external code directly from Dagster.

https://dagster.io/integrations

DIZ Team Magdeburg:

Dr.-Ing.Tim Herrmann
Antonia Schulz
Stefan Krötki
Christian Bruns
Frederike Euchner
Jan Maluche
Dr.rer.nat. Robert Waschipky
Prof.Dr.rer.nat.Dr.med.Johannes Bernarding

Fragen zum Thema:

Ralf.Luetzkendorf@med.ovgu.de

Vielen Dank für Ihre Aufmerksamkeit!

Fragen oder Anmerkungen ?

Referenzen

https://airflow.apache.org/

https://dagster.io/