**Сравнение Dagster и Apache Airflow**

1. Концептуальные различия

- Airflow:

* Task-centric (ориентирован на задачи)
* Работает с Directed Acyclic Graphs (DAG)
* Управляет зависимостями через `>>` и `<<`
* Плохо подходит для управления данными как активами

- Dagster:

* Asset-centric (ориентирован на данные)
* Взаимосвязи определяются автоматически через зависимости активов
* Встроенная поддержка метаданных и lineage
* Удобнее интегрируется с современными инструментами (dbt, Spark, Pandas)

2. Интеграция с dbt

- Airflow:

* Использует операторы (например, `DbtRunOperator`)
* Требует ручного описания зависимостей между dbt-моделями
* Сложность отслеживания данных между этапами

- Dagster:

* Автоматически загружает dbt-модели как активы
* Визуализирует lineage граф без дополнительного кода
* Позволяет комбинировать dbt с Python-обработкой (например, визуализация Plotly после dbt)

3. UI и отладка

- Airflow:

* Интерфейс сосредоточен на статусе задач
* Сложная отладка из-за разделения логики и данных

- Dagster:

* UI показывает данные (активы) и их состояние
* Встроенный просмотр метаданных (например, статистика таблиц, графики)
* Интерактивная отладка с помощью `dagster dev`

4. Производительность и масштабирование

- Airflow:

* Проверен в production (используется в Spotify, Airbnb)
* Поддерживает распределенное выполнение (KubernetesExecutor)
* Высокий порог входа для сложных пайплайнов

- Dagster:

* Лучше подходит для data-centric workflows
* Гибкость в тестировании (моки, изоляция активов)
* Менее зрелый для больших кластеров

Короткий отзыв

Почему Dagster?

Dagster идеален для современных ETL/ELT-процессов, где данные — это first-class citizen. Его интеграция с dbt выглядит естественно: модели становятся частью общего графа активов, а не изолированными задачами. В отличие от Airflow, не нужно вручную прописывать зависимости — система автоматически отслеживает связи через метаданные.

Когда выбрать Airflow?

Если у вас legacy-инфраструктура или нужен жесткий контроль над задачами (например, для CI/CD). Airflow выигрывает в зрелости и поддержке сообщества.

**Пошаговая инструкция: интеграция Dagster и dbt**

1. **Подготовка окружения**
2. Создаем новую директорию для проекта:

mkdir dagster\_dbt

cd dagster\_dbt

1. Создаем и активируем виртуальное окружение:

python -m venv venv

venv\Scripts\activate

1. Установим необходимые пакеты:

pip install dagster dagster-webserver dagster-dbt dbt-core dbt-duckdb duckdb pandas plotly

1. **Настройка dbt проекта**
2. Создаем базовый dbt проект:

dbt init basic\_dbt\_project

cd basic\_dbt\_project

1. Обновим `dbt\_project.yml`:

name: 'basic\_dbt\_project'

version: '1.0.0'

profile: 'basic\_dbt\_project'

1. Создаем `profiles.yml` в папке `basic\_dbt\_project`:

basic\_dbt\_project:

target: dev

outputs:

dev:

type: duckdb

path: dev.duckdb

threads: 1

1. Создаем простые модели dbt:

- models/example/my\_first\_dbt\_model.sql:

SELECT 1 AS id

- `models/example/my\_second\_dbt\_model.sql`:

SELECT id FROM {{ ref('my\_first\_dbt\_model') }}

1. **Настройка Dagster проекта**
2. Возвращаемся в корневую директорию и создаем структуру Dagster:

cd ..

mkdir -p dagster\_project/dagster\_project

1. Создаем `dagster\_project/\_\_init\_\_.py` (пустой файл)
2. Создаем `workspace.yaml` в корневой директории:

load\_from:

- python\_file:

relative\_path: "dagster\_project/definitions.py"

1. Создаем `dagster\_project/definitions.py`:

import os

from pathlib import Path

import duckdb

import pandas as pd

import plotly.express as px

from dagster\_dbt import (

DbtCliResource,

dbt\_assets,

get\_asset\_key\_for\_model,

build\_schedule\_from\_dbt\_selection

)

import dagster as dg

Пути к файлам

project\_dir = Path(\_\_file\_\_).parent.parent

dbt\_project\_dir = project\_dir / "basic\_dbt\_project"

duckdb\_path = str(dbt\_project\_dir / "dev.duckdb")

manifest\_path = dbt\_project\_dir / "target" / "manifest.json"

Проверка и компиляция dbt проекта

dbt = DbtCliResource(project\_dir=dbt\_project\_dir)

if not manifest\_path.exists():

dbt.cli(["compile"]).wait()

if not manifest\_path.exists():

raise Exception(f"Manifest file not found at {manifest\_path}")

@dg.asset(compute\_kind="python")

def raw\_customers():

data = pd.read\_csv("https://docs.dagster.io/assets/customers.csv")

conn = duckdb.connect(duckdb\_path)

conn.execute("CREATE SCHEMA IF NOT EXISTS raw")

conn.execute("CREATE OR REPLACE TABLE raw.raw\_customers AS SELECT \* FROM data")

conn.close()

return len(data)

@dbt\_assets(manifest=manifest\_path)

def dbt\_models(context: dg.AssetExecutionContext, dbt: DbtCliResource):

yield from dbt.cli(["build"], context=context).stream()

@dg.asset(

    compute\_kind="python",

    deps=[get\_asset\_key\_for\_model([dbt\_models], "customers")]

)

def customer\_histogram():

    conn = duckdb.connect(duckdb\_path)

    df = conn.sql("SELECT customer\_id, FIRST\_NAME, LAST\_NAME FROM customers").df()

    conn.close()

    fig = px.histogram(df, x="FIRST\_NAME")

    html\_path = str(project\_dir / "customer\_histogram.html")

    fig.write\_html(html\_path)

    return html\_path

defs = dg.Definitions(

assets=[raw\_customers, dbt\_models, customer\_histogram],

resources={"dbt": dbt},

)

1. **Создаем dbt модели**
2. Создаем `models/customers.sql`:

SELECT

id as customer\_id,

first\_name,

last\_name

FROM {{ source('raw', 'raw\_customers') }}

1. Создаем `models/sources.yml`:

version: 2

sources:

- name: raw

tables:

- name: raw\_customers

meta:

dagster:

asset\_key: ["raw\_customers"]

1. Скомпилируем dbt-проект:

cd basic\_dbt\_project

dbt compile

cd ..

1. **Запуск проекта**

1. Запускаем веб-интерфейс Dagster:

dagster dev

2. Откроем в браузере: http://localhost:3000

3. В интерфейсе вы видим:

- Граф зависимостей между активами

- Возможность ручного запуска pipeline

- Расписание автоматического выполнения

1. **Создание .env файла**

Создаем `.env` файл в корневой директории:

DAGSTER\_HOME=./dagster\_home

DBT\_PROFILES\_DIR=./basic\_dbt\_project