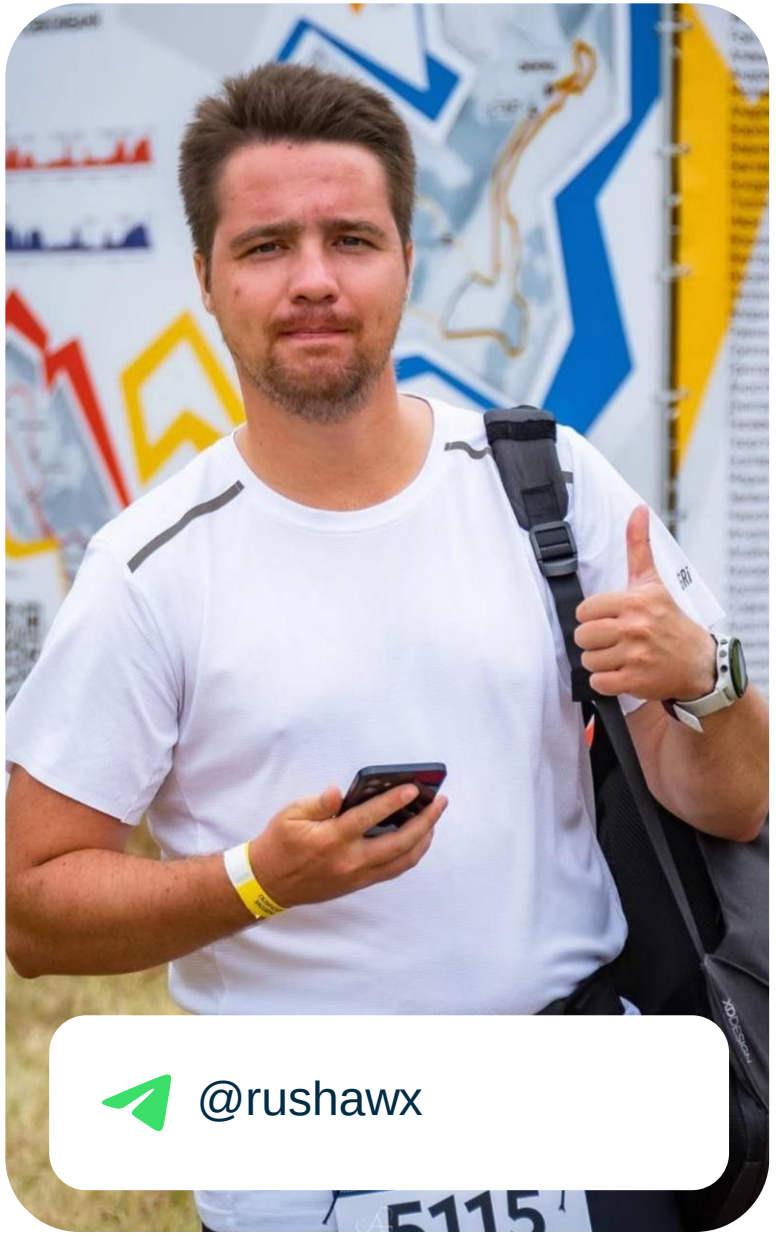
OZON tech

Apache Airflow

Антон Шишков
Разработчик информационных систем



Антон Шишков

Разработчик информационных систем

- Пишу сервисы на **Python**, **Golang**, **SQL**
- Из инструментов также использую **Docker**, **Airflow**, **Postgres**, **Clickhouse**, **Redis**, **Elasticsearch**, **Superset**, **Grafana**, **Kafka**, **Spark**
- В свободное время занимаюсь бегом и образованием

REPOSITORY

AGENDA

- Что такое Airflow
- Что такое DAG
- Архитектура Airflow
- Подготовка окружения
- Написание приложения на FastAPI
- Создание инстансов Postgres, Clickhouse и Minio
- Рассмотрение базовых концепций Airflow
- Написание базового DAG
- Написание продвинутого DAG

Что такое Airflow

Airflow –

фреймворк для построения конвейеров обработки данных

- Ключевая особенность Airflow заключается в том, что он позволяет **легко создавать конвейеры обработки данных**, запускаемых по расписанию, с использованием гибкой платформы Python

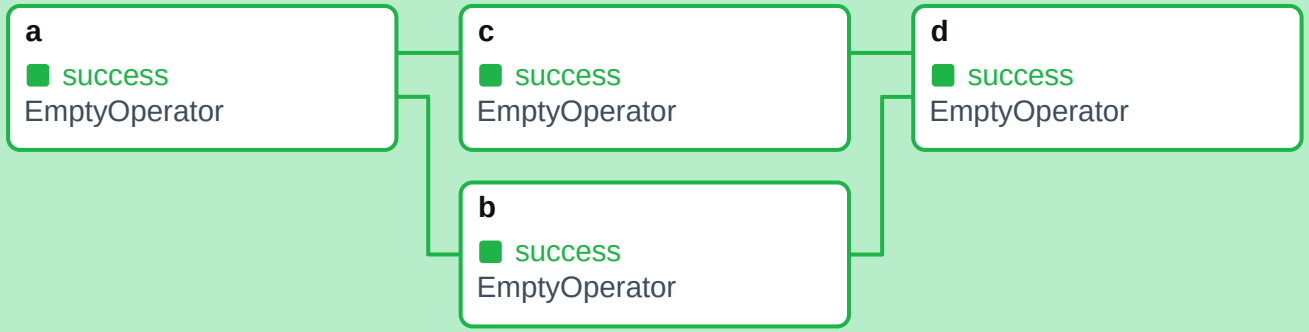
- А также предоставляет **множество строительных блоков**, которые позволяют объединить огромное количество различных технологий, встречающихся в современных технологических ландшафтах

Харенслак Б., де Руйтер Дж. Apache Airflow и конвейеры обработки данных / пер. с англ. Д. А. Беликова. –М.: ДМК Пресс, 2021. – 502 с.

Что такое DAG

Ориентированный ациклический граф (DAG) –

это основная концепция Airflow, объединяющая задачи вместе, организованные с зависимостями и связями, чтобы определить, как они должны выполняться



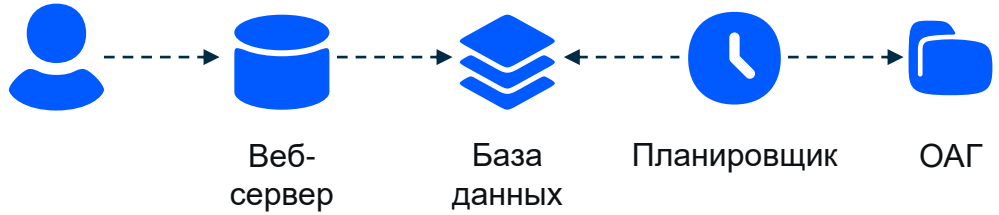
DAG сам по себе не заботится о том, что происходит внутри задач; он лишь занимается тем, **как их выполнять** – в каком порядке их запускать, сколько раз повторять попытки, есть ли у них ограничения по времени и так далее

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

Архитектура Airflow

В минимальном варианте Airflow состоит из **трёх компонентов**:

- **веб-сервер**
- **планировщик**
- **база данных**



- **Веб-сервер** и **планировщик** – это процессы Airflow
- **База данных** – отдельная служба, которую вы должны предоставить Airflow для хранения метаданных с веб-сервера и планировщика
- Папка с определениями ОАГ должна быть доступна планировщику

Харенслак Б., де Руйтер Дж. Apache Airflow и конвейеры обработки данных / пер. с англ. Д. А. Беликова. –М.: ДМК Пресс, 2021. – 502 с.

Подготовка окружения

- Инициализация Git-репозитория
- Создание виртуального Python-окружения
- Установка внешних зависимостей
- Настройка pre-commit
- Создание Makefile
- Подготовка переменных окружения

Написание приложения на FastAPI

fakerApi. Dockerfile

```dockerfile
# Dockerfile

FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "app.app:app", "--host", "0.0.0.0", "
    --port", "8000"]
```

Dockerfile

fakerApi. Models

```python
# fakerApi/app/models/person.py

import datetime
import uuid

from pydantic import BaseModel
from typing import Optional


class PersonResponse(BaseModel):
    id: uuid.UUID
    name: str
    age: int
    address: str
    email: str
    phone_number: str
    registration_date: datetime.datetime
    created_at: datetime.datetime
    updated_at: datetime.datetime
    deleted_at: Optional[datetime.datetime] = None

```

Models

fakerApi. Handlers

```python
# fakerApi/app/handlers/person.py

import datetime

from fastapi import APIRouter
from faker import Faker
from app.models.person import PersonResponse

router = APIRouter(prefix="/person", tags=["person"])

faker = Faker(locale="ru_RU")


@router.get("/", response_model=PersonResponse)
async def get_person():
    person = PersonResponse(
        id=faker.uuid4(),
        name=faker.name(),
        age=faker.random_int(min=18, max=99),
        address=faker.address(),
        email=faker.email(),
        phone_number=faker.phone_number(),
        registration_date=faker.date_time_between(start_date="-1y", end_date="now"),
        created_at=faker.date_time_between(start_date="-1y", end_date="now"),
        updated_at=datetime.datetime.now(),
        deleted_at=None,
    )
    return person

```

Handlers

fakerApi. Main

```python
# fakerApi/app/app.py

import uvicorn
from fastapi import FastAPI
from app.handlers.person import router as
    person_router

app = FastAPI()

app.include_router(person_router)


@app.get("/")
async def root():
    return {"message": "Hello, Simulative!"}


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)

```

Main

Поднимем Postgres, Clickhouse, Minio и fakerApi

Postgres

```yaml
# docker-compose-services.yaml

services:
  pg:
    image: postgres:latest
    container_name: pg
    env_file:
      - ./.env
    environment:
      - POSTGRES_PASSWORD=${PG_PASSWORD}
      - POSTGRES_USER=${PG_USER}
      - POSTGRES_DB=${PG_DATABASE}
    ports:
      - "5432:5432"
    healthcheck:
      test: /usr/bin/pg_isready
      interval: 10s
      timeout: 10s
      retries: 5
    restart: unless-stopped
    volumes:
      - ./init/pg:/docker-entrypoint-initdb.d

```

Postgres

Clickhouse

```yaml
# docker-compose-services.yaml

services:

  #

  zookeeper:
    image: zookeeper:latest
    container_name: zookeeper
    hostname: zookeeper

  ch:
    image: clickhouse/clickhouse-server:latest
    container_name: ch
    hostname: ch
    ports:
      - "8123:8123"
      - "9000:9000"
    volumes:
      - ./data/clickhouse/node1:/etc/clickhouse-server
      - ./data/clickhouse:/docker-entrypoint-initdb.d
    depends_on:
      - zookeeper
    healthcheck:
      test: wget --no-verbose --tries=1 http://127.0.0.1:8123/ping ||
          exit 1
      interval: 10s
      timeout: 10s
      retries: 5

```

Clickhouse

Minio

```yaml
# docker-compose-services.yaml

services:

  #

  minio:
    image: quay.io/minio/minio
    command: server /minio_data --console-address ":9001"
    ports:
      - "9001:9000"  # Remap MinIO API port
      - "9002:9001"  # Remap MinIO Console port
    environment:
      - MINIO_ROOT_USER=minioadmin
      - MINIO_ROOT_PASSWORD=minioadmin
    volumes:
      - minio_data:/minio_data
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:9002/minio/health/live"]
      interval: 30s
      timeout: 20s
      retries: 3


```

Minio

fakerApi

```yaml
# docker-compose-services.yaml

services:

  #

  faker-api:
    build: fakerApi
    ports:
      - "8000:8000"
    depends_on:
      pg:
        condition: service_healthy
      ch:
        condition: service_healthy

```

fakerApi

Базовые концепции

- **DAG**
- **DummyOperator**
- **BashOperator**
- **PythonOperator**
- **SQLSensor**
- **ExternalTaskSensor**

DAG

```python
import datetime

from airflow.models.dag import DAG
from airflow.operators.dummy import DummyOperator
from airflow.decorators import task
from airflow.sensors.sql import SqlSensor
from airflow.sensors.external_task_sensor import ExternalTaskSensor


with DAG(
    dag_id="basic_dag",
    schedule="0 18 * * *",
    start_date=datetime.datetime(2025, 1, 1),
    catchup=False,
    tags=["simulative"],
) as dag:

    #
```

DAG

Cron

crontab.guru

Crontab.guru - The cron schedule expression generator

An easy to use editor for crontab schedules.

DummyOperator

```python
from airflow.operators.dummy import DummyOperator


dummy_task = DummyOperator(task_id="dummy_task")
```

DummyOperator

BashOperator

```python
from airflow.decorators import task


@task.bash
def bash_task():
    return "echo https://airflow.apache.org/"

bash_task = bash_task()
```

BashOperator

PythonOperator

```python
from airflow.decorators import task


@task
def python_task():
    print("Hello Python!")


python_task = python_task()
```

PythonOperator

SqlOperator

```python
from airflow.sensors.sql import SqlSensortask


sql_sensor = SqlSensor(
    task_id='sql_check',
    conn_id='postgres',
    sql="""SELECT 1""",
)
```

SqlOperator

ExternalTaskOperator

```python
from airflow.sensors.external_task_sensor import ExternalTaskSensor


external_sensor = ExternalTaskSensor(
    task_id='external_check',
    external_dag_id='simulative_example_basic_dag',
    external_task_id='print_hello',
    execution_date_fn=lambda dt: dt,
    dag=dag
)
```

ExternalTaskOperator

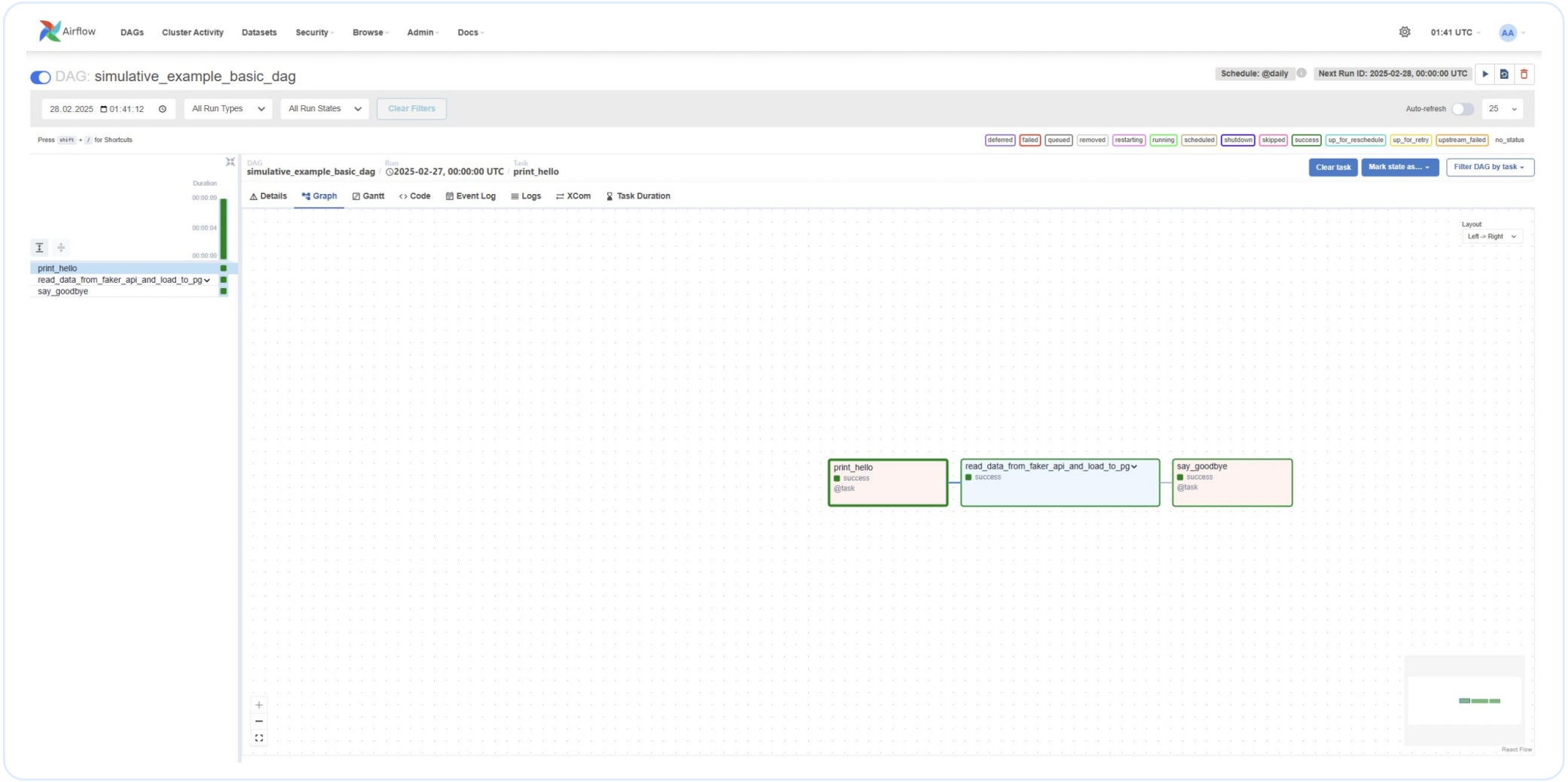
executionOrder

```
1    bash_task >> python_task >> [sql_sensor, external_sensor] >> dummy_task
```

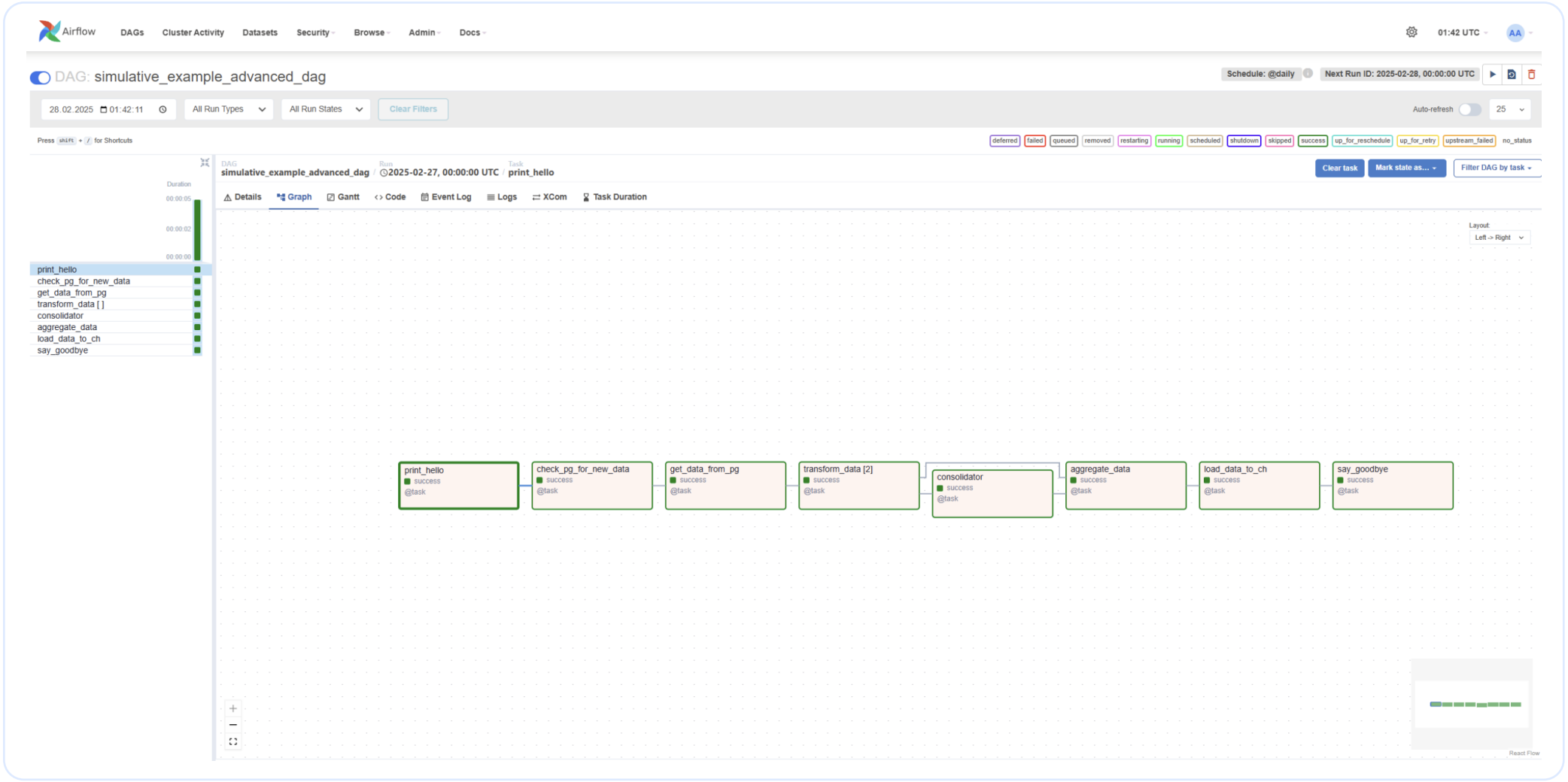
executionOrder

LiveCoding. Airflow

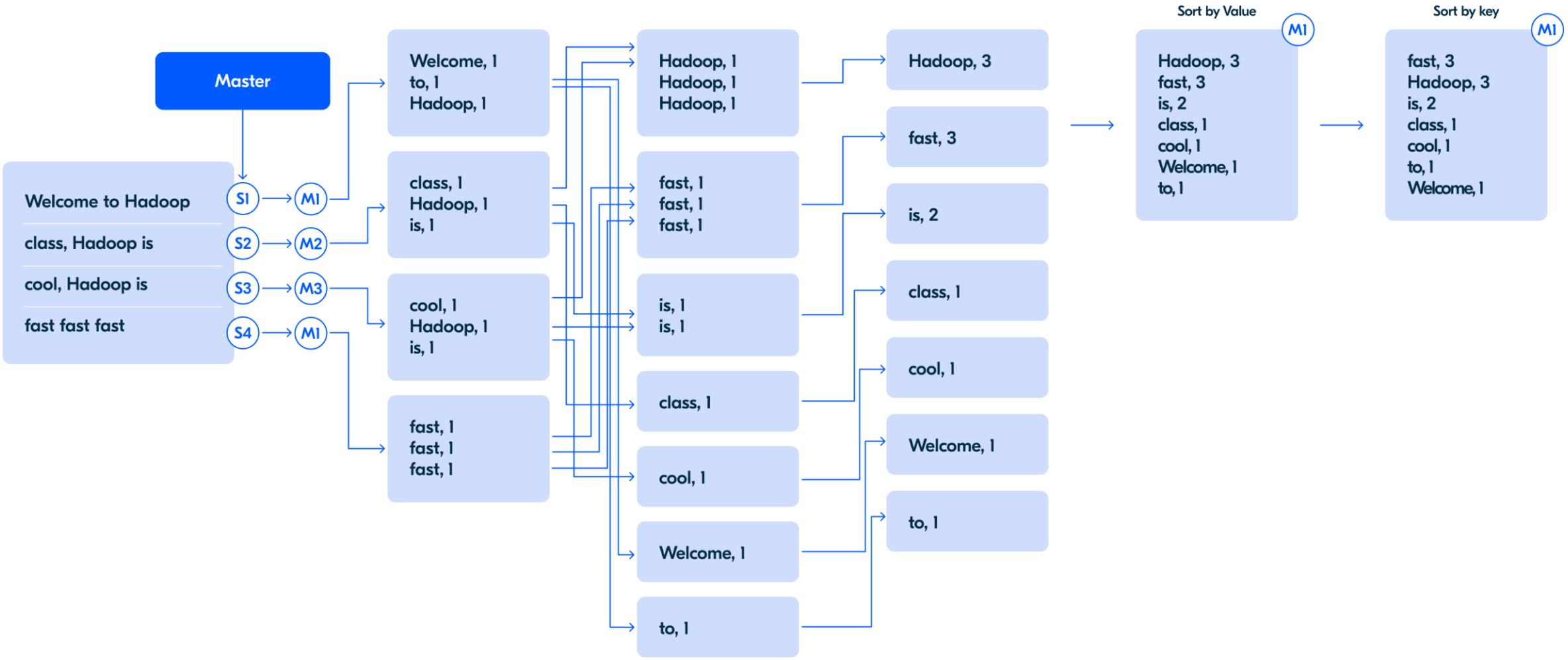
LiveCoding. Airflow. Basic DAG



LiveCoding. Airflow. Advanced DAG



MapReduce


Master
Welcome to Hadoop
class, Hadoop is
cool, Hadoop is
fast fast fast
S1
S2
S3
S4
M1
M2
M3
M1
Welcome, 1
to, 1
Hadoop, 1
class, 1
Hadoop, 1
is, 1
cool, 1
Hadoop, 1
is, 1
fast, 1
fast, 1
fast, 1
Hadoop, 1
Hadoop, 1
Hadoop, 1
fast, 1
fast, 1
fast, 1
is, 1
is, 1
class, 1
cool, 1
Welcome, 1
to, 1
Hadoop, 3
fast, 3
is, 2
class, 1
cool, 1
Welcome, 1
to, 1
Sort by Value
M1
Hadoop, 3
fast, 3
is, 2
class, 1
cool, 1
Welcome, 1
to, 1
Sort by key
M1
fast, 3
Hadoop, 3
is, 2
class, 1
cool, 1
to, 1
Welcome, 1

Заключение

За сегодняшний вебинар мы с вами:

- познакомились с Airflow
- узнали что такое DAG
- создали веб-сервис на FastAPI
- подняли Airflow, Postgres, Clickhouse, Minio
- настроили ETL процессы HTTP-SQL-NoSQL

Data Analyst Intern

https://job.ozon.ru/vacancy/stazhyor-analitik-dannikh-marketpleis-otdel-produktovoi-analitiki-117822969

OZON tech

Спасибо за внимание!

Антон Шишков
Разработчик информационных систем

ashishkov@ozon.ru / @rushawx