



# Apache Airflow



**АНТОН ШИШКОВ**

Разработчик информационных систем





@rushawx

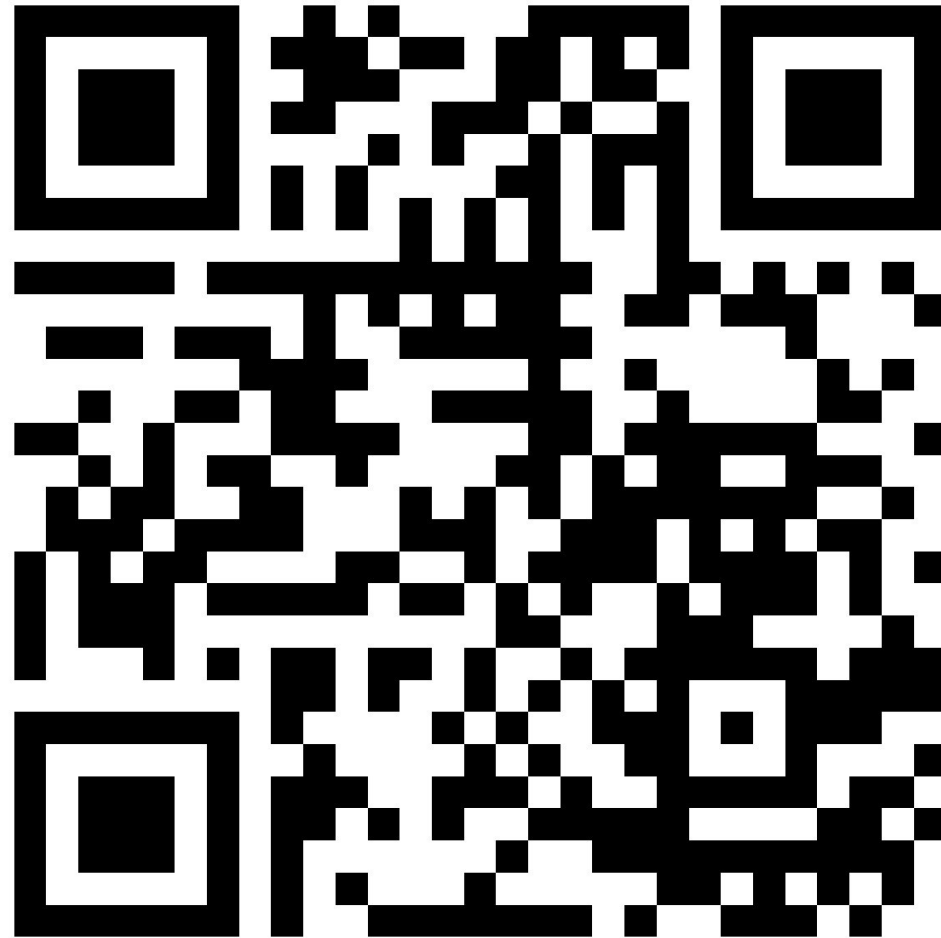
# АНТОН ШИШКОВ



Разработчик информационных систем

- ✓ Пишу сервисы на **Python** и **Golang**
- ✓ Поддерживаю пайплайны данных на **Python** и **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



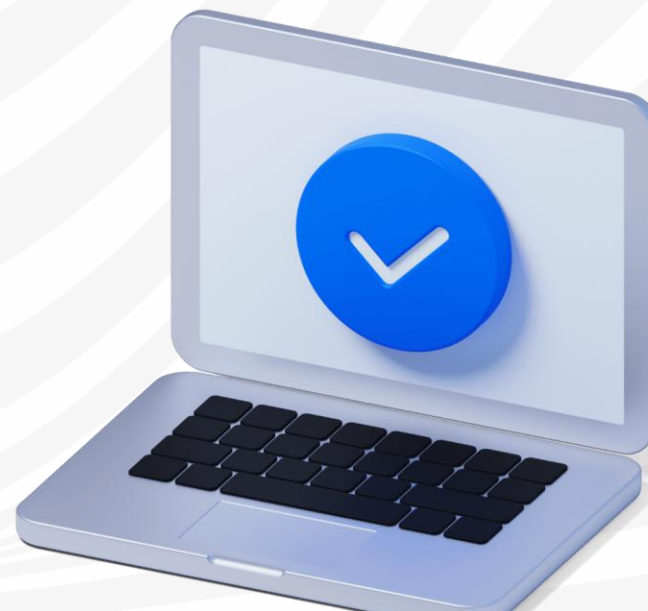
Рассмотрение базовых концепций Airflow



Написание базового DAG



Написание продвинутого DAG



# Что такое Airflow



## Airflow –

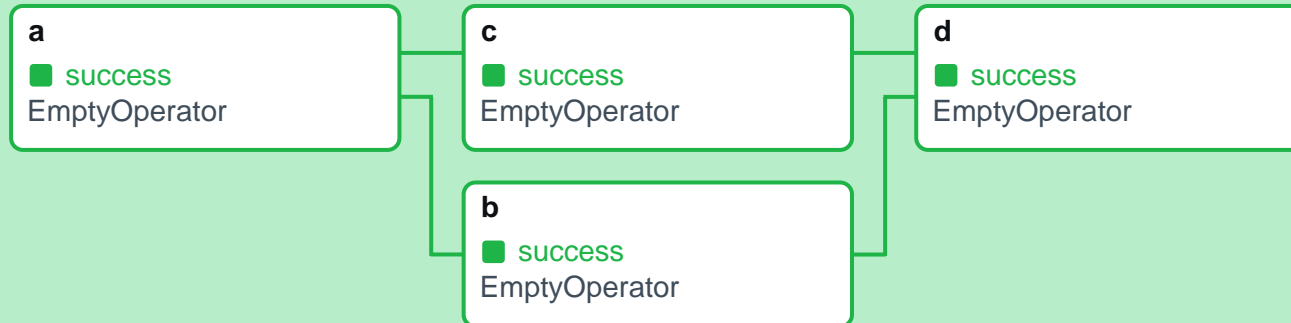
фреймворк для построения конвейеров  
обработки данных

- ➔ 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


# 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



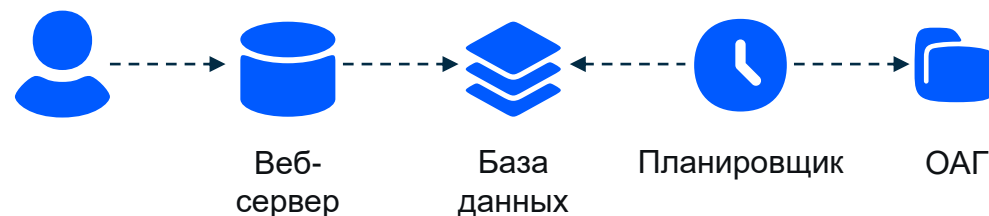
DAG сам по себе не заботится о том, что происходит внутри задач; он лишь занимается тем, **как их выполнять** – в каком порядке их запускать, сколько раз повторять попытки, есть ли у них ограничения по времени и так далее

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

# Архитектура Airflow

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

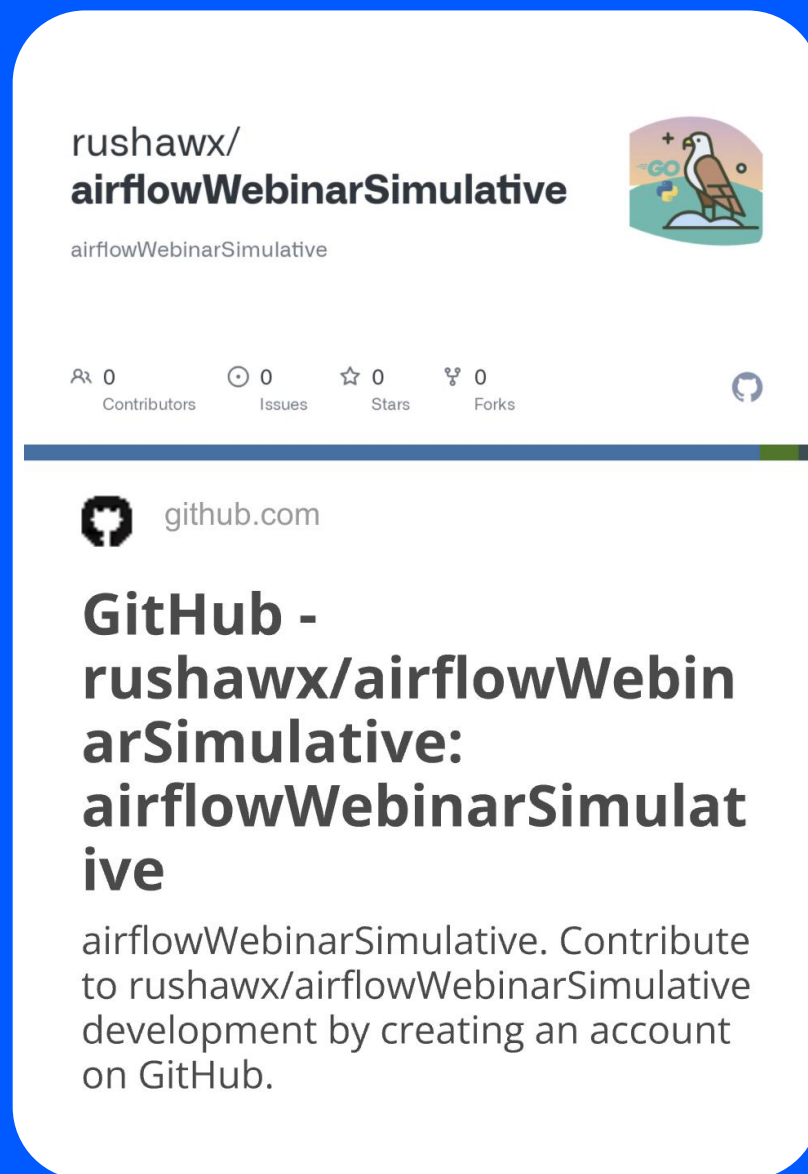
- веб-сервер
- планировщик
- база данных



- 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

# 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

```
1  # Dockerfile
2
3  FROM python:3.12-slim
4
5  WORKDIR /app
6
7  COPY requirements.txt .
8  RUN pip install --no-cache-dir -r requirements.txt
9
10 COPY . .
11
12 CMD ["uvicorn", "app.app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Dockerfile

# fakerApi. Models

```
1  # fakerApi/app/models/person.py
2
3  import datetime
4  import uuid
5
6  from pydantic import BaseModel
7  from typing import Optional
8
9
10 class PersonResponse(BaseModel):
11     id: uuid.UUID
12     name: str
13     age: int
14     address: str
15     email: str
16     phone_number: str
17     registration_date: datetime.datetime
18     created_at: datetime.datetime
19     updated_at: datetime.datetime
20     deleted_at: Optional[datetime.datetime] = None
21
```

Models

# fakerApi. Handlers

```
1  # fakerApi/app/handlers/person.py
2
3  import datetime
4
5  from fastapi import APIRouter
6  from faker import Faker
7  from app.models.person import PersonResponse
8
9  router = APIRouter(prefix="/person", tags=["person"])
10
11  faker = Faker(locale="ru_RU")
12
13
14  @router.get("/", response_model=PersonResponse)
15  async def get_person():
16      person = PersonResponse(
17          id=faker.uuid4(),
18          name=faker.name(),
19          age=faker.random_int(min=18, max=99),
20          address=faker.address(),
21          email=faker.email(),
22          phone_number=faker.phone_number(),
23          registration_date=faker.date_time_between(start_date="-1y", end_date="now"),
24          created_at=faker.date_time_between(start_date="-1y", end_date="now"),
25          updated_at=datetime.datetime.now(),
26          deleted_at=None,
27      )
28      return person
29
```

Handlers

# fakerApi. Main

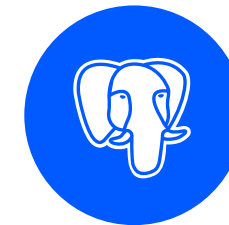
```
1  # fakerApi/app/app.py
2
3  import uvicorn
4  from fastapi import FastAPI
5  from app.handlers.person import router as
    person_router
6
7  app = FastAPI()
8
9  app.include_router(person_router)
10
11
12  @app.get("/")
13  async def root():
14      return {"message": "Hello, Simulative!"}
15
16
17  if __name__ == "__main__":
18      uvicorn.run(app, host="0.0.0.0", port=8000)
19
```

Main



Поднимем Postgres,  
Clickhouse, Minio и fakerApi

# Postgres



```
1  # docker-compose-services.yaml
2
3  services:
4    pg:
5      image: postgres:latest
6      container_name: pg
7      env_file:
8        - ./env
9      environment:
10        - POSTGRES_PASSWORD=${PG_PASSWORD}
11        - POSTGRES_USER=${PG_USER}
12        - POSTGRES_DB=${PG_DATABASE}
13      ports:
14        - "5432:5432"
15      healthcheck:
16        test: /usr/bin/pg_isready
17        interval: 10s
18        timeout: 10s
19        retries: 5
20      restart: unless-stopped
21      volumes:
22        - ./init/pg:/docker-entrypoint-initdb.d
23
```

Postgres

# Clickhouse



```
1  # docker-compose-services.yaml
2
3  services:
4
5      #
6
7      zookeeper:
8          image: zookeeper:latest
9          container_name: zookeeper
10         hostname: zookeeper
11
12     ch:
13         image: clickhouse/clickhouse-server:latest
14         container_name: ch
15         hostname: ch
16         ports:
17             - "8123:8123"
18             - "9000:9000"
19         volumes:
20             - ./data/clickhouse/node1:/etc/clickhouse-server
21             - ./data/clickhouse:/docker-entrypoint-initdb.d
22         depends_on:
23             - zookeeper
24         healthcheck:
25             test: wget --no-verbose --tries=1 http://127.0.0.1:8123/ping ||
26                 exit 1
27             interval: 10s
28             timeout: 10s
29             retries: 5
```

Clickhouse



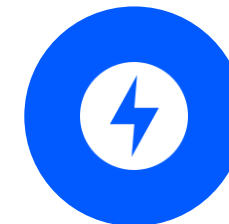
# Minio



```
1  # docker-compose-services.yaml
2
3  services:
4
5      #
6
7      minio:
8          image: quay.io/minio/minio
9          command: server /minio_data --console-address ":9001"
10         ports:
11             - "9001:9000" # Remap MinIO API port
12             - "9002:9001" # Remap MinIO Console port
13         environment:
14             - MINIO_ROOT_USER=minioadmin
15             - MINIO_ROOT_PASSWORD=minioadmin
16         volumes:
17             - minio_data:/minio_data
18         healthcheck:
19             test: ["CMD", "curl", "-f", "http://localhost:9002/minio/health/live"]
20             interval: 30s
21             timeout: 20s
22             retries: 3
23
24
```

Minio

# fakerApi



```
1  # docker-compose-services.yaml
2
3  services:
4
5      #
6
7      faker-api:
8          build: fakerApi
9          ports:
10             - "8000:8000"
11          depends_on:
12              pg:
13                  condition: service_healthy
14              ch:
15                  condition: service_healthy
16
```

fakerApi

# Базовые концепции

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



# DAG

```
1  import datetime
2
3  from airflow.models.dag import DAG
4  from airflow.operators.dummy import DummyOperator
5  from airflow.decorators import task
6  from airflow.sensors.sql import SqlSensor
7  from airflow.sensors.external_task_sensor import ExternalTaskSensor
8
9
10 with DAG(
11     dag_id="basic_dag",
12     schedule="0 18 * * *",
13     start_date=datetime.datetime(2025, 1, 1),
14     catchup=False,
15     tags=["simulative"],
16 ) as dag:
17
18     #
```

DAG

# Cron

crontab guru

The quick and simple editor for cron schedule expressions by [Cronitor](#)

“At 18:00.”

next at 2025-02-28 18:00:00

random

018\*\*

minute	hour	day (month)	month	day (week)
*		any value		
'		value list		
-		separator		
-		range of values		
/		step values		
@yearly		(non-standard)		
@annually		(non-standard)		
@monthly		(non-standard)		
@weekly		(non-standard)		
@daily		(non-standard)		
@hourly		(non-standard)		
@reboot		(non-standard)		



**Crontab.guru - The cron schedule expression generator**

An easy to use editor for crontab schedules.

# DummyOperator

```
1  from airflow.operators.dummy import DummyOperator
2
3
4  dummy_task = DummyOperator(task_id="dummy_task")
```

DummyOperator

# BashOperator

```
1  from airflow.decorators import task
2
3
4  @task.bash
5  def bash_task():
6      return "echo https://airflow.apache.org/"
7
8
9  bash_task = bash_task()
```

BashOperator

# PythonOperator

```
1  from airflow.decorators import task
2
3
4  @task
5  def python_task():
6      print("Hello Python!")
7
8
9  python_task = python_task()
```

PythonOperator



# SqlOperator

```
1  from airflow.sensors.sql import SqlSensortask
2
3
4  sql_sensor = SqlSensor(
5      task_id='sql_check',
6      conn_id='postgres',
7      sql="\"\"\"SELECT 1\"\"\"",
8  )
```

SqlOperator

# ExternalTaskOperator

```
1  from airflow.sensors.external_task_sensor import ExternalTaskSensor
2
3
4  external_sensor = ExternalTaskSensor(
5      task_id='external_check',
6      external_dag_id='simulative_example_basic_dag',
7      external_task_id='print_hello',
8      execution_date_fn=lambda dt: dt,
9      dag=dag
10 )
```

ExternalTaskOperator

# executionOrder

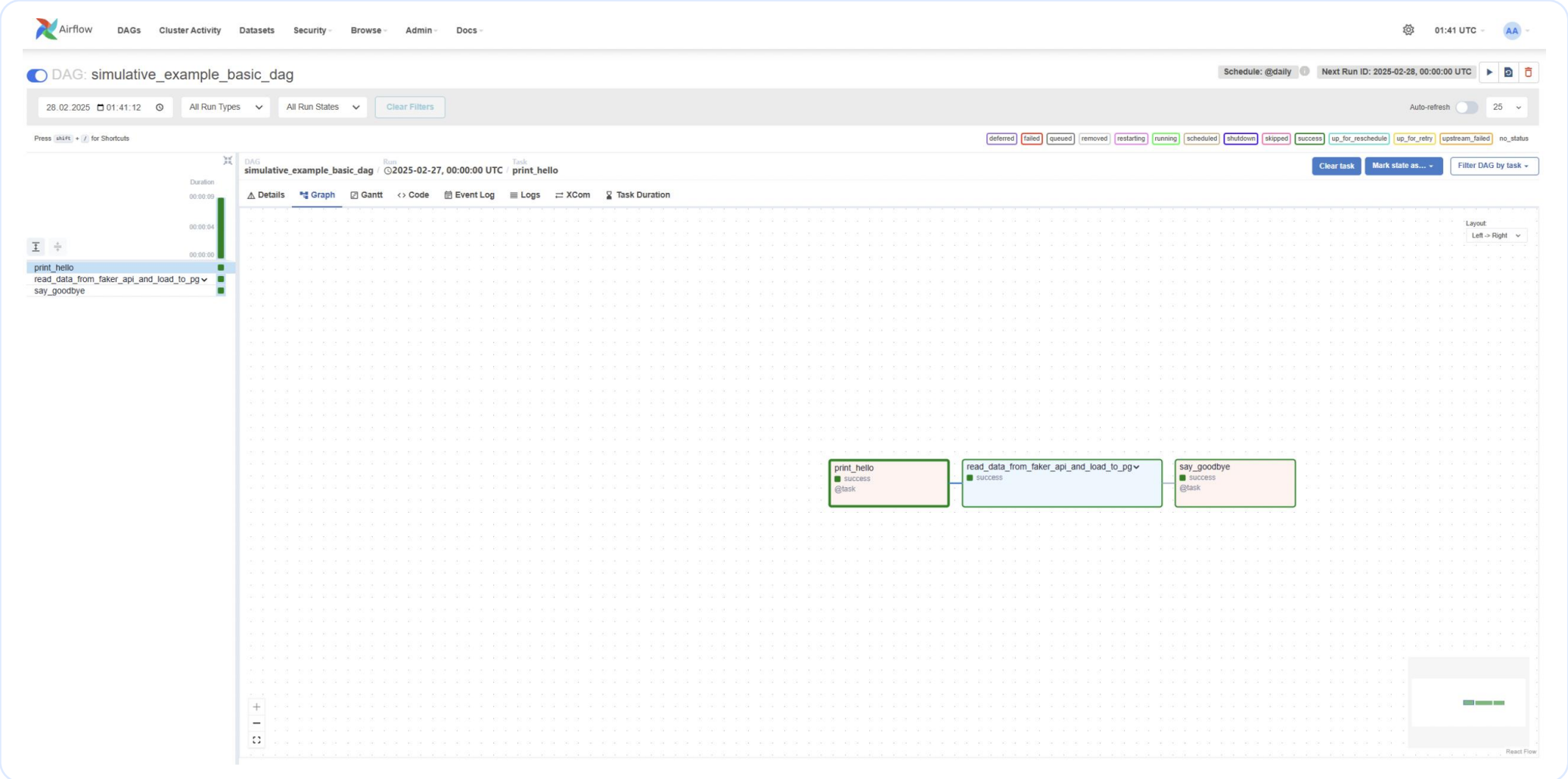
```
1    bash_task >> python_task >> [sql_sensor, external_sensor] >> dummy_task
```

executionOrder

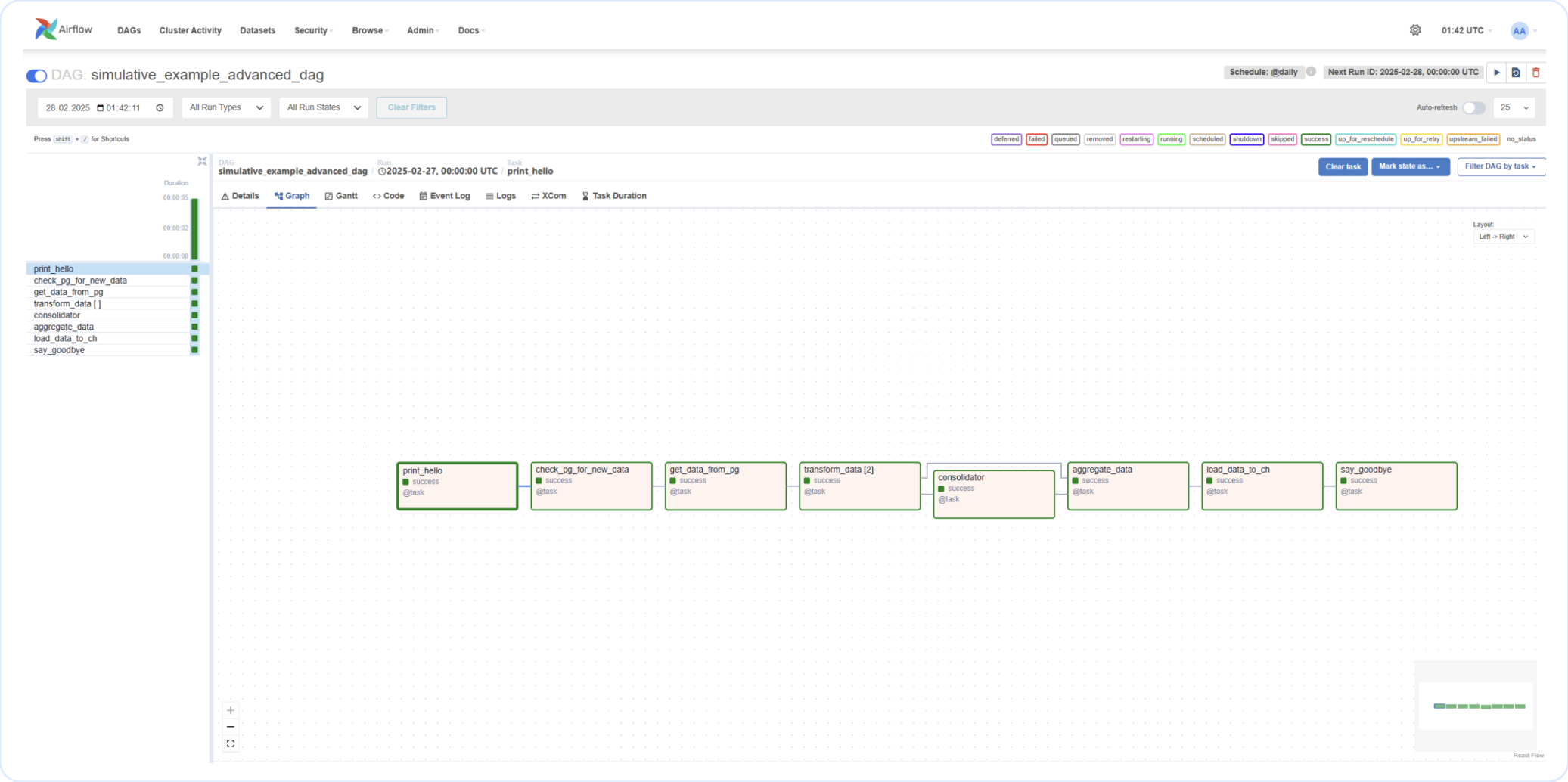


LiveCoding. Airflow

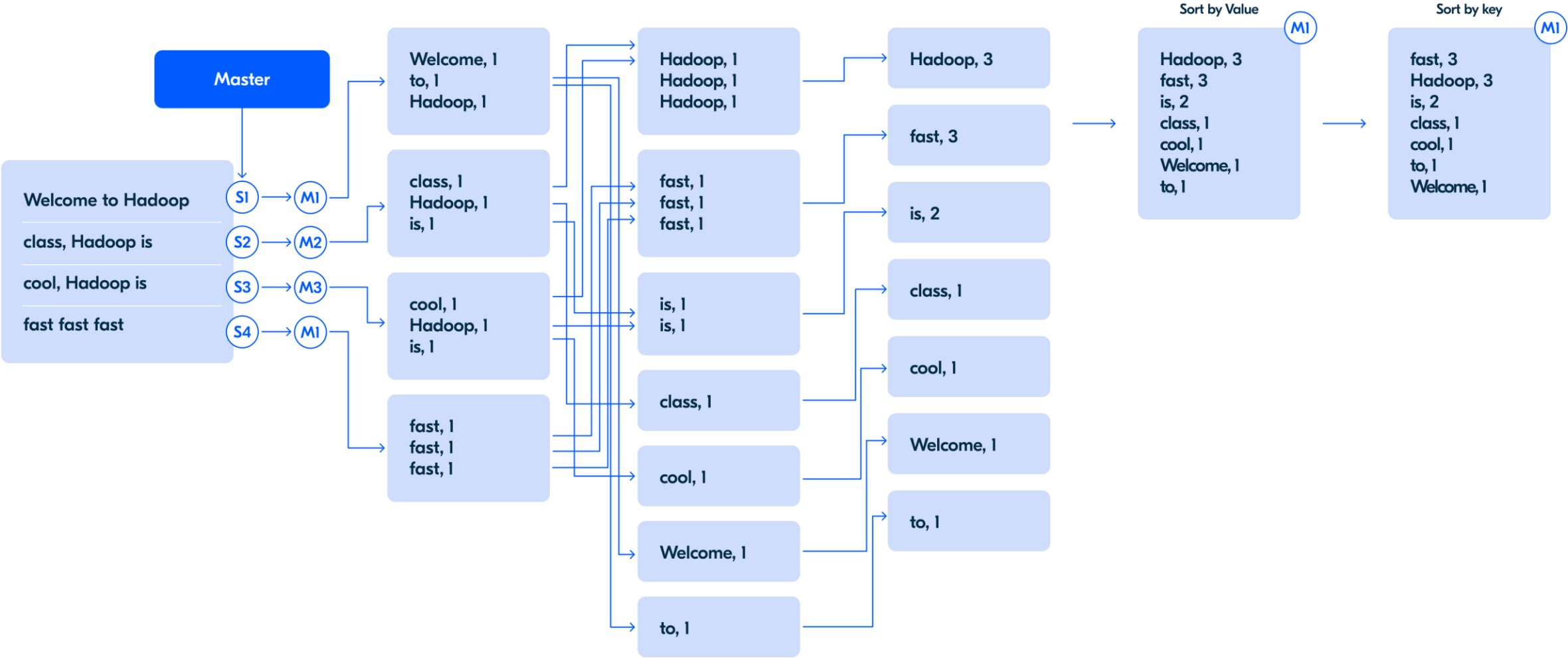
# LiveCoding. Airflow. Basic DAG



# LiveCoding. Airflow. Advanced DAG



# MapReduce



# 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





# Спасибо за внимание!



**АНТОН ШИШКОВ**

Разработчик информационных систем

ashishkov@ozon.ru / @rushawx