**Apache Airflow**

Apache Airflow — это платформа с открытым исходным кодом, созданная для **программного создания, планирования и мониторинга рабочих процессов** (workflows). Она особенно полезна в задачах **обработки данных**, **ETL-пайплайнах**, **машинном обучении**, и других автоматизированных процессах.

**💡 Основные понятия Airflow:**

1. **DAG (Directed Acyclic Graph)** — направленный ацикличный граф, который описывает порядок выполнения задач. Это основа всего рабочего процесса в Airflow.
   * Каждая вершина графа — это задача.
   * Ребра — зависимости между задачами (что за чем выполняется).
2. **Task** — отдельная задача, например:
   * запуск Python-скрипта;
   * выполнение SQL-запроса;
   * перемещение файлов;
   * вызов API и др.
3. **Operator** — это шаблон или обёртка, определяющая, как выполняется задача. Есть разные типы операторов:
   * PythonOperator — выполняет Python-функцию;
   * BashOperator — запускает bash-команду;
   * PostgresOperator, BigQueryOperator и т. д. — выполняют запросы к базам данных.
4. **Scheduler** — компонент, который планирует запуск DAG'ов в соответствии с расписанием (например, каждый день в 8 утра).
5. **Executor** — отвечает за фактическое выполнение задач (может работать локально, параллельно или распределённо — например, с Celery или Kubernetes).
6. **Web UI** — веб-интерфейс для мониторинга DAG'ов, отслеживания их состояния, перезапуска задач и отладки.

### Как работает Airflow:

1. **Вы пишете DAG** — это Python-файл, в котором описывается, что и когда делать.
2. **Scheduler** регулярно проверяет DAG-и и определяет, какие задачи нужно запустить.
3. **Executor** запускает эти задачи (локально или в распределённой среде).
4. **Metadata Database** — хранит информацию о запусках DAG'ов, состоянии задач, логах и др.
5. **Web UI** показывает всё это в виде удобного графического интерфейса.

Настройка **Apache Airflow** в **Docker**

**🔧 Что мы будем делать?**

1. Установим нужные папки и файлы
2. Создадим docker-compose.yaml
3. Запустим контейнеры
4. Зайдём в Airflow и проверим, что всё работает

## Что нужно ДО начала

* Установлен **Docker**
* Установлен **Docker Compose**
* Любой терминал: PowerShell (Windows), Git Bash, или Terminal (Mac/Linux)

**Шаг 1**. В Docker в нужной папке в терминале введи:

mkdir airflow-docker

cd airflow-docker

mkdir dags, logs, plugins

**Шаг 2.** Создай файл docker-compose.yaml

Открой папку airflow-docker в Проводнике

Создай файл с именем docker-compose.yaml

Открой его и вставь туда следующий код:

version: '3.7'

services:

postgres:

image: postgres:13

environment:

POSTGRES\_USER: airflow

POSTGRES\_PASSWORD: airflow

POSTGRES\_DB: airflow

ports:

- "5432:5432"

volumes:

- postgres-db-volume:/var/lib/postgresql/data

airflow-webserver:

image: apache/airflow:2.9.1

restart: always

depends\_on:

- postgres

environment:

AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

AIRFLOW\_\_WEBSERVER\_\_EXPOSE\_CONFIG: "True"

volumes:

- ./dags:/opt/airflow/dags

- ./logs:/opt/airflow/logs

- ./plugins:/opt/airflow/plugins

ports:

- "8080:8080"

command: >

bash -c "

airflow db upgrade &&

airflow users create --username admin --password admin --firstname Admin --lastname User --role Admin --email admin@example.com &&

airflow webserver

"

airflow-scheduler:

image: apache/airflow:2.9.1

restart: always

depends\_on:

- airflow-webserver

environment:

AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

volumes:

- ./dags:/opt/airflow/dags

- ./logs:/opt/airflow/logs

- ./plugins:/opt/airflow/plugins

command: >

bash -c "

airflow scheduler

"

volumes:

postgres-db-volume:

### Шаг 3: Запусти контейнеры

1. В терминале убедись, что ты находишься в папке airflow-docker.
2. Выполни команду:

docker-compose up -d

Это запустит:

* PostgreSQL (база данных)
* Web-сервер Airflow
* Планировщик задач Airflow (scheduler)

**Шаг 5: Добавь пример DAG**

Создай файл dags/hello\_world\_dag.py со следующим содержимым:

from airflow import DAG

from airflow.operators.python import PythonOperator

from datetime import datetime

def hello():

print("Привет от Airflow!")

with DAG(

'hello\_world',

start\_date=datetime(2024, 1, 1),

schedule\_interval='@daily',

catchup=False

) as dag:

task = PythonOperator(

task\_id='say\_hello',

python\_callable=hello

)

**Как остановить**

Если хочешь остановить контейнеры:

docker-compose down

**Быстрая перезапуск-проверка**

Чтобы попробовать перезапустить всё аккуратно:

docker-compose down

docker-compose up -d

Это:

* Остановит все контейнеры
* Перезапустит их заново в фоне

Потом снова:

docker ps

и открыть в браузере:

http://localhost:8080





## 1. **Автоматическое обновление DAG-ов без перезапуска**

Когда ты меняешь DAG-файлы (\*.py) в папке dags/, **Airflow сам их подхватывает** каждые 10–30 секунд.

✅ То есть:

* Ты редактируешь Python-код
* Просто жмёшь "Refresh" в Airflow UI (на сайте)
* Новый код появляется в списке DAG-ов

**❗ Перезапускать контейнеры НЕ НАДО.**

Иногда бывает, что DAG долго не появляется. Тогда просто нажми кнопку "🔄 Refresh" наверху в Airflow UI.



<http://localhost:8080>

## 2. **Смотреть логи контейнера в реальном времени**

Если нужно увидеть, что прямо сейчас происходит внутри контейнера (например, webserver-а):

docker logs -f airflow-webserver

* -f значит **"follow"** — следить за логами вживую.
* Чтобы выйти — нажимаешь Ctrl + C.

Также можно смотреть логи других контейнеров, например:

docker logs -f airflow-scheduler

docker logs -f airflow-triggerer

Очень полезно, когда что-то грузится или не запускается.

# 🚀 Как увидеть список всех сервисов из docker-compose.yaml

Есть команда:

docker-compose config --services

Когда её запустишь, она выведет **только имена сервисов**, которые прописаны в docker-compose.yaml.

**Если хочешь видеть ВСЁ содержимое конфига (а не только сервисы)** — можно просто:

docker-compose config

(без --services) — тогда покажет весь слитый и проверенный yaml.

Это удобно, если ты хочешь увидеть, что реально запустится.

## 4. **Быстро проверить состояние всех сервисов**

Чтобы быстро посмотреть, какие контейнеры работают и сколько они живут:

bash

КопироватьРедактировать

docker ps

Ты увидишь таблицу:

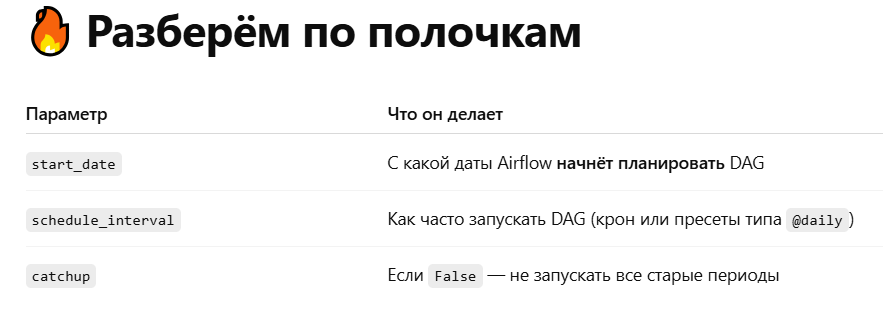
* Название контейнера
* Какой порт проброшен (например, 0.0.0.0:8080->8080/tcp)
* Статус (Up 5 minutes, Exited, и т.д.)

# 📋 Как сделать DAG, который работает по расписанию

Каждый DAG в Airflow имеет специальный параметр:

schedule\_interval

Он задаёт, **когда и как часто** нужно запускать DAG.



# 📅 Как писать расписания?

Можно задавать через:

* **готовые шаблоны**:
  + @once — один раз
  + @daily — раз в сутки
  + @hourly — каждый час
  + @weekly — каждую неделю
* **или в формате cron**:
  + "0 0 \* \* \*" — каждый день в полночь
  + "0 \*/6 \* \* \*" — каждые 6 часов
  + "15 10 \* \* 1-5" — в 10:15 утра по будням

# Очень ВАЖНО!

* start\_date должна быть в прошлом или сейчас.  
  Иначе DAG просто не начнёт запускаться.
* Если не хочешь, чтобы DAG запускался **за все пропущенные даты**, обязательно укажи catchup=False.
* Scheduler должен работать! (airflow-scheduler контейнер должен быть запущен)

**Примеры DAG’ов**

**Несколько задач подряд**

****

**Ветвление**

****

**Обработка ошибки**

****

**Передача данных из одной задачи в другую с помощью XCom**

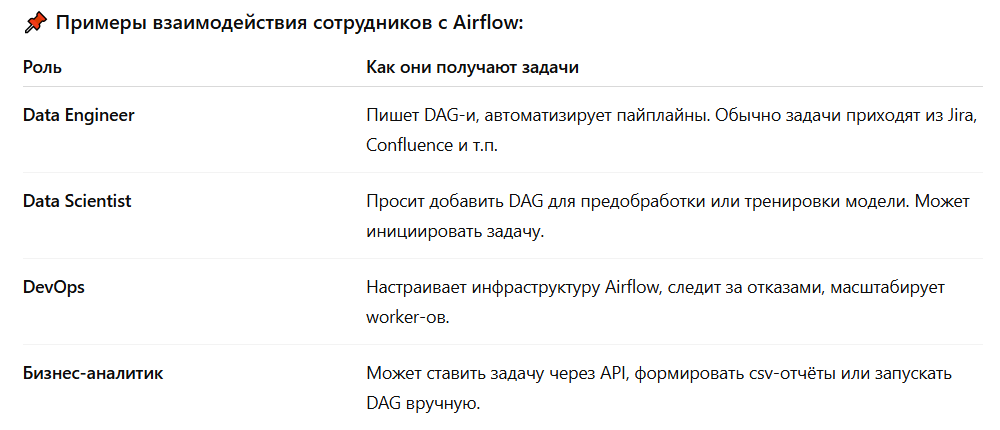
****

### 1. ****Как задачи назначаются и исполняются в Airflow (между машинами/процессами)?****

Если ты спрашиваешь, как Airflow распределяет задачи между исполнителями (workers), то вот подробное объяснение:

#### 💡 Архитектура Airflow:

1. **Scheduler**
   * Отвечает за планирование DAG-ов (на основе расписания).
   * Находит задачи, которые нужно запустить.
   * Помещает их в очередь (обычно через брокер — например, Redis или RabbitMQ).
2. **Executor**
   * Выбирает, как запускать задачи:
     + SequentialExecutor — по одной задаче за раз (для тестов).
     + LocalExecutor — многопоточно на одной машине.
     + CeleryExecutor — распределённо на разных машинах (через брокер и workers).
     + KubernetesExecutor — каждая задача — отдельный pod в Kubernetes.
3. **Workers (исполнители)**
   * Это процессы или контейнеры, которые берут задачи из очереди и исполняют.
   * Именно они "получают" задачи — т.е. запускают Python-код или другой скрипт, определённый в PythonOperator, BashOperator, HttpOperator и т.д.

****

**Сценарий: как "сотрудник получает задачу через Airflow"**

Предположим, у тебя есть аналитик Ольга, которая хочет каждый день в 8:00 получать отчёт по продажам.

1. **Data Engineer** создаёт DAG с расписанием daily at 08:00.
2. DAG генерирует отчёт (например, Excel-файл).
3. Финальный шаг DAG-а — отправка email через EmailOperator или загрузка файла в Google Drive.
4. **Ольга получает задачу в виде готового отчёта или уведомления.**

**Ключевые инструменты в Airflow для "получения задач"**

* TriggerDagRunOperator — позволяет одному DAG запускать другой (реагировать на событие).
* BranchPythonOperator — реализует логику "кому что делать".
* EmailOperator, SlackWebhookOperator — отправляют сотрудникам уведомления.
* XCom — передаёт данные между задачами, в том числе нужную информацию сотруднику.

**Вывод**

* **Сотрудники напрямую задачи из Airflow не получают** — Airflow работает внутри инфраструктуры.
* **"Получение задач" происходит через автоматизацию**: генерация файлов, отчётов, уведомлений и прочих результатов DAG-ов.
* Технически, **workers получают задачи от Scheduler-а** через Executor и очередь.