

Оркестр процессов обработки данных с помощью **Apache Airflow**

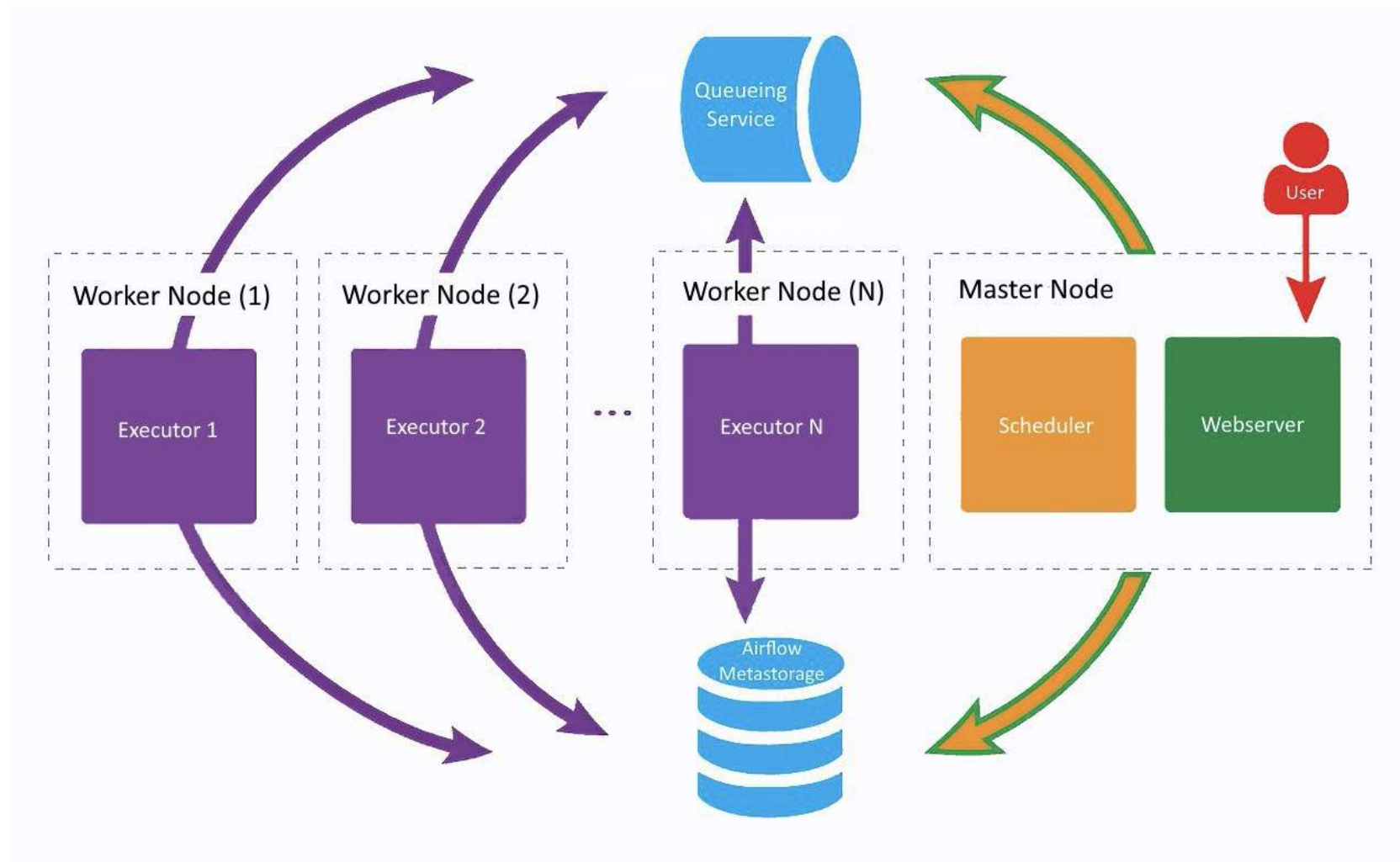
Что такое Airflow

Airflow – это платформа для создания, мониторинга и оркестрации пайплайнов.

Этот open source проект, написанный на Python, был создан в 2014 году в компании **Airbnb**.

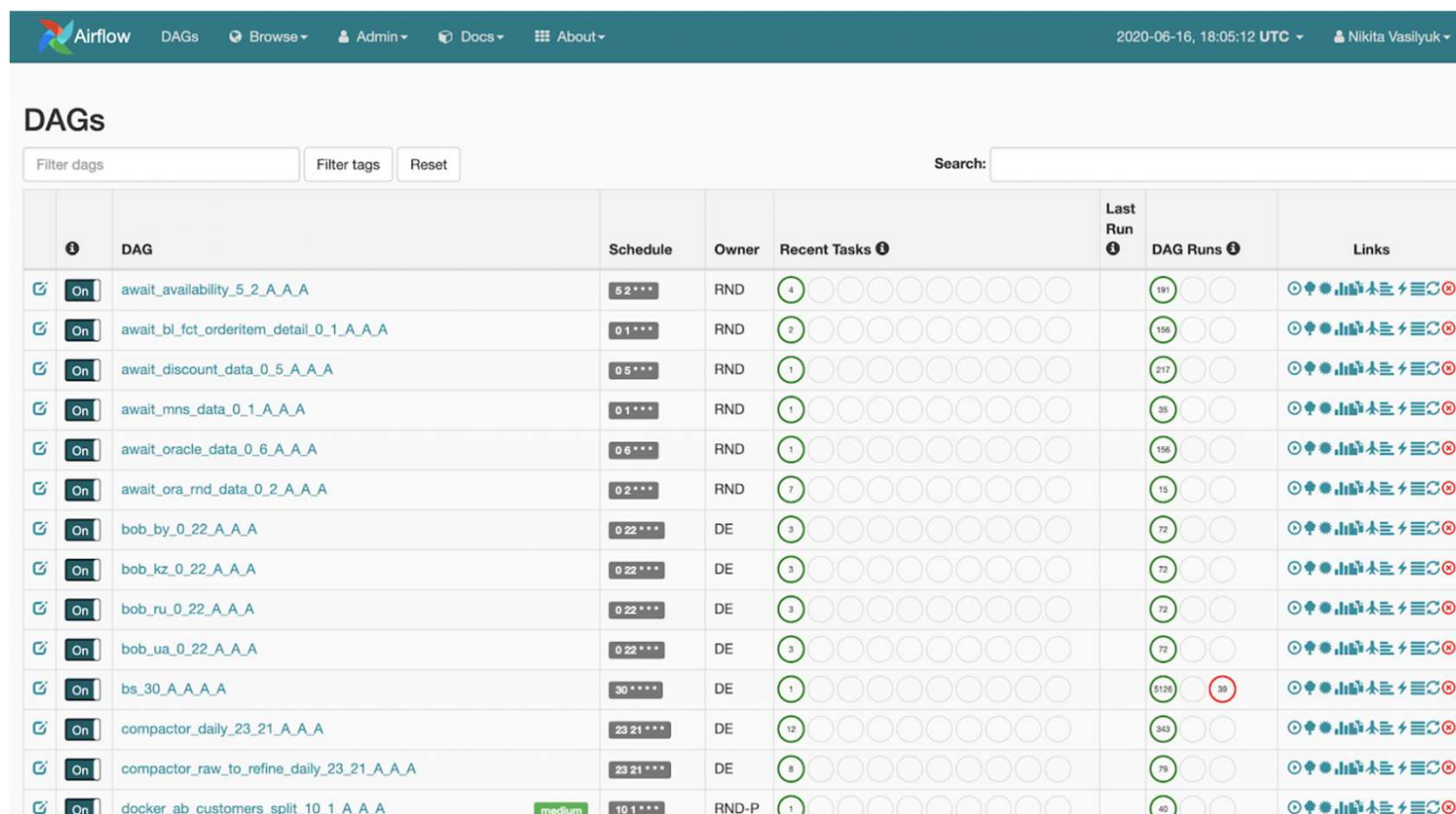
В 2016 году **Airflow** ушел к **Apache Software Foundation**, прошел через инкубатор и в начале 2019 года перешел в статус **top-level** проекта **Apache**.

Архитектура Airflow



Компоненты Airflow

- **Webserver** – это веб-интерфейс, показывающий, что сейчас происходит с пайплайном. Эту страницу видит пользователь:



The screenshot shows the Apache Airflow Webserver interface. At the top, there's a navigation bar with links for DAGs, Browse, Admin, Docs, and About. The current date and time are 2020-06-16, 18:05:12 UTC, and the user is Nikita Vasilyuk. Below the navigation bar, the title "DAGs" is displayed. There are filters for "Filter dags", "Filter tags", and "Reset". A search bar is also present. The main table lists various DAGs with columns for DAG name, Schedule, Owner, Recent Tasks, Last Run, DAG Runs, and Links. The table shows 15 DAGs, with the last one, "docker_ab_customers_split_10_1_A_A_A", highlighted in green.

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
On	await_availability_5_2_A_A_A	5 2 ***	RND	4	191		
On	await_bl_fct_orderitem_detail_0_1_A_A_A	0 1 ***	RND	2	156		
On	await_discount_data_0_5_A_A_A	0 5 ***	RND	1	217		
On	await_mns_data_0_1_A_A_A	0 1 ***	RND	1	35		
On	await_oracle_data_0_6_A_A_A	0 6 ***	RND	1	156		
On	await_ora_rnd_data_0_2_A_A_A	0 2 ***	RND	7	15		
On	bob_by_0_22_A_A_A	0 22 ***	DE	3	72		
On	bob_kz_0_22_A_A_A	0 22 ***	DE	3	72		
On	bob_ru_0_22_A_A_A	0 22 ***	DE	3	72		
On	bob_ua_0_22_A_A_A	0 22 ***	DE	3	72		
On	bs_30_A_A_A_A	30 * ***	DE	1	5126	39	
On	compactor_daily_23_21_A_A_A	23 21 ***	DE	12	343		
On	compactor_raw_to_refine_daily_23_21_A_A_A	23 21 ***	DE	8	79		
On	docker_ab_customers_split_10_1_A_A_A	10 1 ***	RND-P	1	40		

Worker

- **Worker** – это место, где запускается наш код и выполняются задачи.

Worker

- **Worker** – **Airflow** поддерживает несколько экзекьюторов:

- **SequentialExecutor**. Он последовательно запускает прилетающие задачи, а на время их выполнения приостанавливает шедюлер.

- **LocalExecutor** на каждую прилетающую задачу стартует новый процесс, с ним появляется возможность запускать несколько задач в параллель, поэтому **LocalExecutor** лучше предыдущего экзекьютора.

- Есть один нюанс: если в качестве базы метаданных используется что-то типа однопоточного **SQLite**, ваш **LocalExecutor** превращается в **SequentialExecutor**.

Worker

- **Worker** – **Airflow** поддерживает несколько экзекьюторов:

- **CeleryExecutor** позволяет иметь несколько воркеров, работающих на разных машинах. **Celery** – это распределенная очередь задач, которая под капотом использует **RabbitMQ** или **Redis**. При запуске воркера ему можно указать названия очередей, из которых он будет принимать задачи от шедулера.
- **DaskExecutor** запускает задачу с помощью Dask – библиотеки для параллельных вычислений.
- **KubernetesExecutor** на каждую задачу запускает новый pod в Kubernetes.
- **DebugExecutor** создан для запуска и отладки пайплайнов из IDE.

Сущности **Apache Airflow**

- **Пайплайн, или DAG**

Самая важная сущность **Airflow** – это **DAG**, он же пайплайн, он же направленный ациклический граф.

Пайплайн, или DAG

Допустим, к нам пришел аналитик и попросил раз в день загружать данные в определенную таблицу. Он подготовил всю информацию: что откуда нужно брать, когда нужно запускаться. Вот пример того, как мы могли бы описывать наш **пайплайн**.

Пайплайн, или DAG

```
dag = DAG(  
    dag_id="load_some_data",  
    schedule_interval="0 1 * * *",  
    default_args={  
        "start_date": datetime(2020, 4, 20),  
        "owner": "DE",  
        "depends_on_past": False,  
        "sla": timedelta(minutes=45),  
        "email": "<your_email_here>",  
        "email_on_failure": True,  
        "retries": 2,  
        "retry_delay": timedelta(minutes=5)  
    }  
)
```

Оператор

Оператор – это **Python класс**, который описывает, какие действия надо совершить в рамках нашей ежедневной задачи, чтобы порадовать аналитика.

Оператор

Для запуска **Оператора** надо указать название задачи, пайплайн, идентификатор соединения к Hive и выполняемый запрос.

Оператор

```
run_sql = HiveOperator(  
    dag=dag,  
    task_id="run_sql",  
    hive_cli_conn_id="hive",  
    hql="""  
        INSERT OVERWRITE TABLE some_table  
        SELECT * FROM other_table t1  
        JOIN another_table t2 on ...  
        WHERE other_table.dt = '{{ ds }}' """  
)  
  
notify = SlackAPIPostOperator(  
    dag=dag,  
    task_id="notify_slack",  
    slack_conn_id="slack",  
    token=token,  
    channel="airflow_alerts",  
    text="Guys, I'm done for {{ ds }}"  
)  
run_sql >> notify
```

Самые распространенные операторы **Airflow**

BashOperator и **PythonOperator**. С ними все понятно: они отправляют на выполнение bash команду и python функцию соответственно.

Самые распространенные операторы **Airflow**

Поддерживаются стандартные Postgres, **MySQL, Oracle, Hive, Presto**. Если оператора для вашей базы данных почему-то нет, можно использовать более общий **JdbcOperator** или написать свой.

Самые распространенные операторы **Airflow**

Sensor – это оператор, который при запуске проверяет выполнение определенного условия. И если оно не выполняется, оператор на какое-то время засыпает.

Например, оператор может проверять количество строк в таблице, наличие файлов в файловой системе.

Есть сенсор, которому можно сказать: подожди до 3 часов ночи, после этого передай эстафету следующей задаче. Например, мы используем их для проверки готовности данных во внешних системах, чтобы не запускать тяжелый отчет на неполных данных.

Самые распространенные операторы **Airflow**

- **BranchPythonOperator** – это оператор ветвления, который на основании определенного условия, заданного python кодом, решает, какую задачу надо запустить следующей.

Самые распространенные операторы **Airflow**

- **DockerOperator** запускает Docker-контейнер на воркере. Тут нужно понимать, что внутри Docker-контейнера может запускаться все, что угодно. Поэтому очень важно при этом мониторить ресурсы воркера, чтобы они внезапно не закончились.
- **DummyOperator** выполняет роль пустышки и создан для того, чтобы склеивать различные участки пайплайна между собой.



129226, г. Москва, 2-й Сельскохозяйственный проезд, 4
info@mgpu.ru
+7 (499) 181-24-62
www.mgpu.ru

Университет твоих возможностей

Copyright ©ГАОУ ВО МГПУ 2022