

KARPOV.COURSES >>>

КОНСПЕКТ



> Конспект > 2 урок > Знакомство с Airflow

> Оглавление

- > [Оглавление](#)
- > [История Airflow](#)
- > [Основные понятия](#)
 - [Процесс выполнения задач](#)
- > [Компоненты Airflow](#)
- > [User Interface](#)
- > [Простой DAG](#)
 - [Создание DAG-а](#)
 - [Составление пайплайнов](#)
 - [Документация](#)
- > [Глоссарий](#)

> История Airflow

- октябрь 2014 - создание Airflow в Airbnb
- март 2016 - Apache Incubator

- январь 2019 - top-level проект
- конец 2020 - Airflow 2.0

> Основные понятия

Напомним, что:

- Граф - множество, состоящее из вершин и соединяющих их ребер.
- Направленный граф - граф, у которого нет двунаправленных ребер.

DAG (Directed Acyclic Graph) - направленный ациклический граф. Это граф, в котором нет заикливания. В нем могут быть разветвления, но в конце они сходятся в одной точке.

Вершины в DAG - это отдельные задачи (task).

Ребра в DAG - зависимости между задачами.

DAG-ов в Airflow может быть неограниченное количество.

Задача (task) - это одна из двух сущностей:

- **сенсор**. Они используются, когда нам нужно дождаться наступления какого-то события, например, появление строчки в БД, наступление определенного времени, появление файла в s3.
- **оператор**. Это действие, которое выполняет конкретная задача, например, забирает данные из базы, отправляет письма, вычитывает данные из API, производит вычисления.

Задачи объединяются в DAG по смыслу.

Пример DAG:











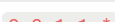
В интерфейсе Airflow по умолчанию задачи прорисовываются слева направо.

Процесс выполнения задач

Сначала запускаются задачи, которые не имеют предшественников. После того, как такие задачи отработали, запускаются зависимые от них задачи. Это продолжается пока мы не дойдем до конца DAG, то есть пока все задачи не будут обработаны.

Если при выполнении задача упала, то она переходит в состояние **retry** и через определенное время (по умолчанию 10 минут) перезапускается. Если после 3 (этот параметр тоже настраивается) перезапусков task так и не смог выполниться успешно, то он переводится в состояние **failed**. Все последующие за ним задачи переводятся в состояние **upstream failed** и выполнены не будут. В таком случае DAG переходит в состояние **failed**.

Расписание

 Airflow	 CRON	 Действие
<u>None</u>		Запускает по триггеру
<u>@once</u>		Один раз
<u>@hourly</u>		Ежечасно
<u>@daily</u>		Ежедневно в полночь
<u>@weekly</u>		Еженедельно в воскресенье в полночь
<u>@monthly</u>		Ежемесячно в полночь первого числа
<u>@quarterly</u>		Каждый квартал в полночь
<u>@yearly</u>		Каждый новый год

DAG выполняется по заданному расписанию.

Определять расписание мы можем несколькими способами:

- явно задать CRON выражение
- использовать специальные алиасы

> Компоненты Airflow

Web Server Airflow - отвечает за пользовательский интерфейс и дает возможность контролировать пайплайны.

Основные задачи:

- Внешний вид DAG-а
- Статус выполнения (их получает из метаданных Airflow)
- Перезапуск (как тасок, так и DAG-ов)
- Отладка

Scheduler - планировщик.

- Анализирует DAG'и (ищет те, которые готовы к запуску)
- Создает DAG Run с конкретным `execution_date`
- Создает Task Instance
- Ставит таски в очередь

Здесь:



DAG Run - это экземпляр DAG-а. Каждый экземпляр характеризуется параметром `execution_date` (начало предыдущего периода).

Task Instance - это экземпляр задачи. Инстансы привязываются к DAG. У них так же определен параметр `execution_date`.

Executor - механизм, с помощью которого запускаются экземпляры задач. Он работает в связке с планировщиком.

Executor-ы делятся на 2 категории:

- локальные - исполняются на той же машине, на которой есть планировщик
- нелокальные - могут запускать таски удаленно

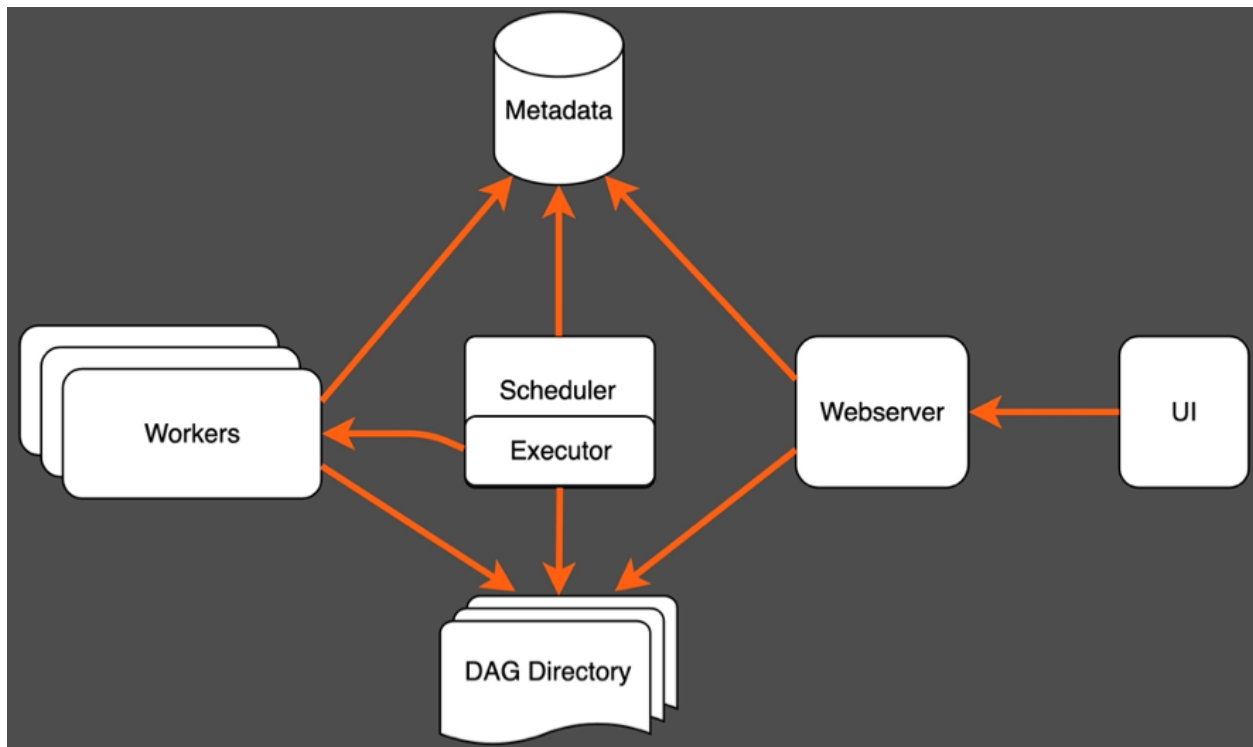
 Executor	<input checked="" type="checkbox"/> Local	 Описание
<u>SequentialExecutor</u>	<input checked="" type="checkbox"/>	Последовательный запуск задач
<u>LocalExecutor</u>	<input checked="" type="checkbox"/>	По дочернему процессу на задачу
<u>DebugExecutor</u>	<input checked="" type="checkbox"/>	Для запуска и отладки из IDE

<u>Aa</u> Executor	<input checked="" type="checkbox"/> Local	☰ Описание
<u>CeleryExecutor</u>	<input type="checkbox"/>	Требуется брокер сообщений. Несколько серверов с воркерами
<u>DaskExecutor</u>	<input type="checkbox"/>	Использует Dask
<u>KubernetesExecutor</u>	<input type="checkbox"/>	Новый pod для каждого task instance
<u>CeleryKubernetesExecutor</u>	<input type="checkbox"/>	CeleryExecutor/KubernetesExecutor
<u>Custom</u>	<input type="checkbox"/>	Самописанный

Worker - процесс, в котором выполняются задачи. В зависимости от executor-а он может быть размещен локально (на той же машине, что и планировщик), либо на отдельной машине/машинах.

Metadata Database - база метаданных. В ней хранится информация о состоянии всех пайплайнов:

- DAG (абстрактные DAG-и)
- DAG Run (конкретные инстансы DAG-ов)
- Task Instance
- Variable (глобальные переменные Airflow)
- Connection
- Xcom
- ...



На схеме видим, что:

- UI (пользовательский интерфейс) находится на Web Server-е.
- Воркеры могут находиться на той же самой машине, что и планировщик, и executor.
- Воркеры, планировщик и веб сервер смотрят в одну папку **DAG Directory** (тут лежат описание DAG-ов в виде питоновских скриптов).
- Воркеры, планировщик и веб сервер смотрят в одну базу метаданных.

























> User Interface

Посмотрим на пользовательский интерфейс Airflow

Airflow DAGs Security Browse Admin Docs 17:37 MSK (+03:00) ДС

DAGs

All 4 Active 3 Paused 1 Filter DAGs by tag Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
 dina_simple_dag karpov	Karpov	 17	@daily	2021-09-15, 03:00:00	 5	   ...	
 dina_simple_dag_v2 karpov	Karpov	 16	@daily	2021-09-15, 03:00:00	 5	   ...	
 dina_simple_dag_v3 karpov	Karpov	 16	@daily	2021-09-15, 03:00:00	 5	   ...	
 hello_world	grigory51	 3	00 12 * * *	2021-09-06, 15:00:00	 1	   ...	

Showing 1-4 of 4 DAGs

Version: v2.1.3
Git Version: .release:2.1.3+2b80c1edec85fb2b0312b6ae7f415c6f744e322e

Стартовая страница Airflow

На стартовой странице мы видим список DAG-ов. Этот список можно фильтровать:

- по статусу (отобразить только запущенные DAG-и или только остановленные)
- по тэгам
- по названию
- по owner-у (имя человека, который сгенерировал DAG или название группы людей)

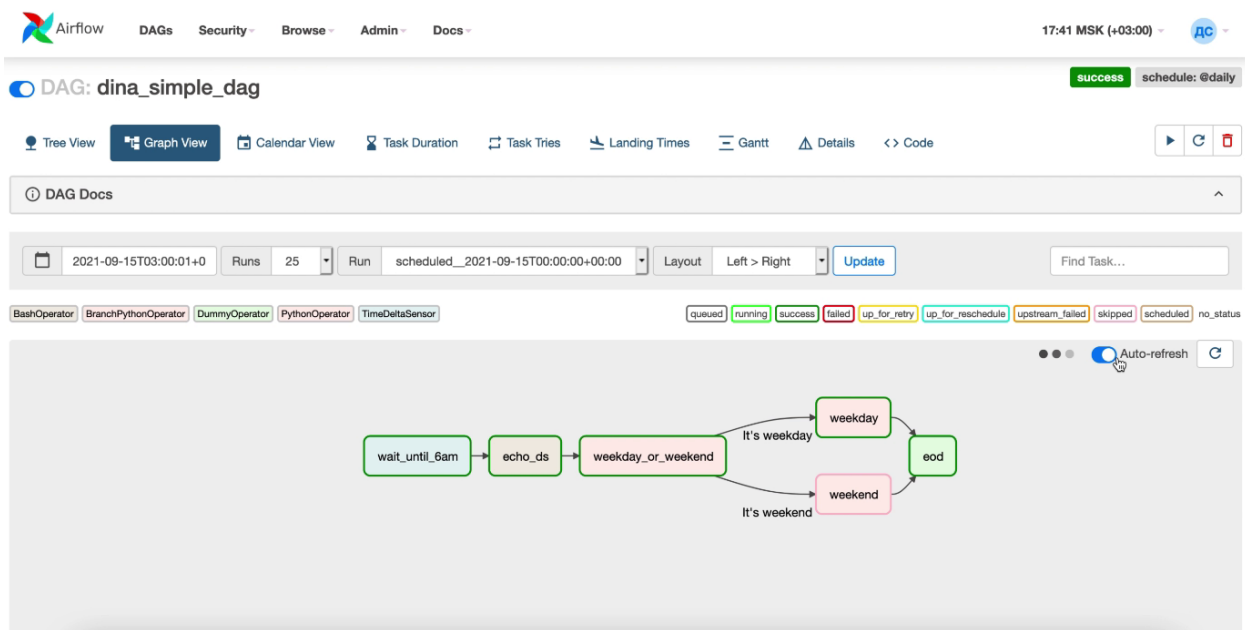
По каждому DAG-у мы видим:

- название
- owner-а
- количество запусков
- расписание
- момент, в который DAG бы запущен в последний раз
- состояние задач в последнем DAG Run

Тут же можно запустить DAG-и (т. е. сгенерировать DAG Run), обновить их из git и удалить их метаданные.

При наведении на значения в колонке Links мы можем перейти на интерфейсы, с более подробной информацией о DAG.

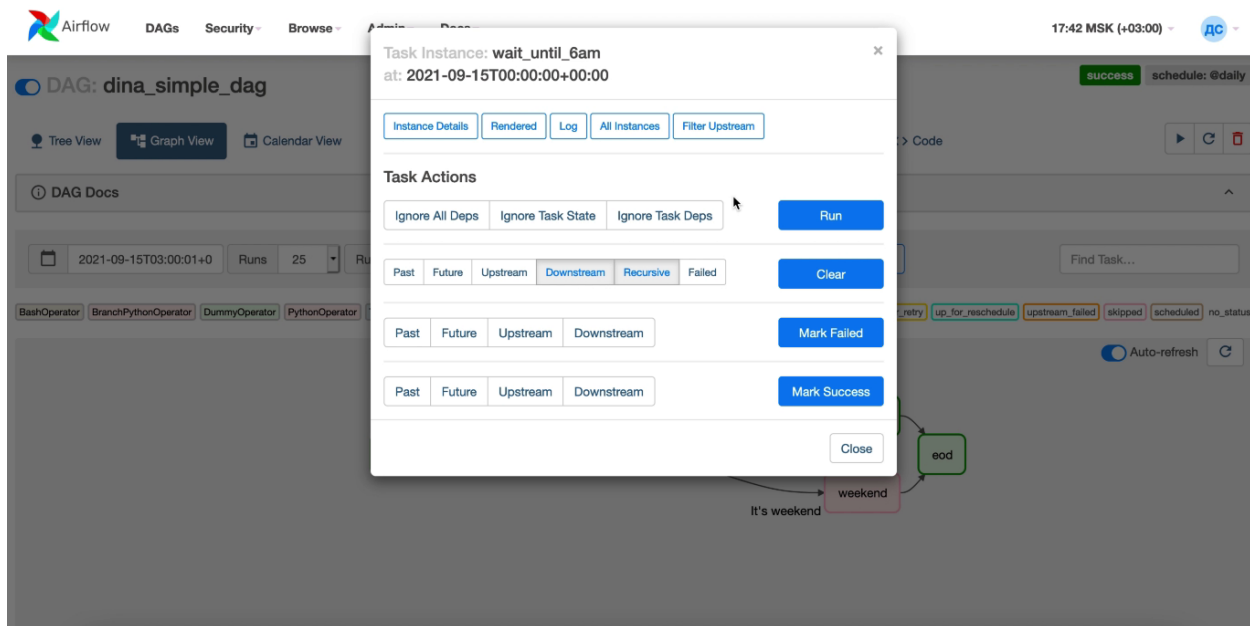
Перейдем на **Graph View**



Здесь мы можем увидеть, как выглядит наш DAG.

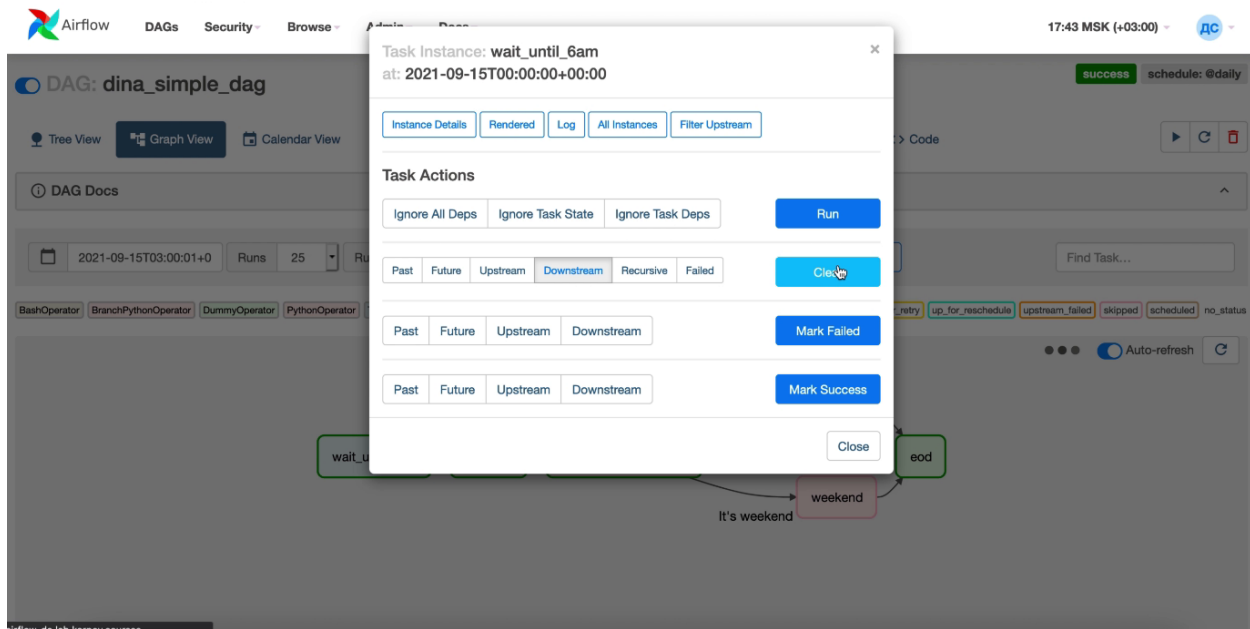
Если включить опцию **Auto-refresh**, то в режиме реального времени мы будем видеть статусы выполнения тасок.

Если кликнуть на определенный таск, то мы увидим окно, в котором мы можем выполнить разные действия:

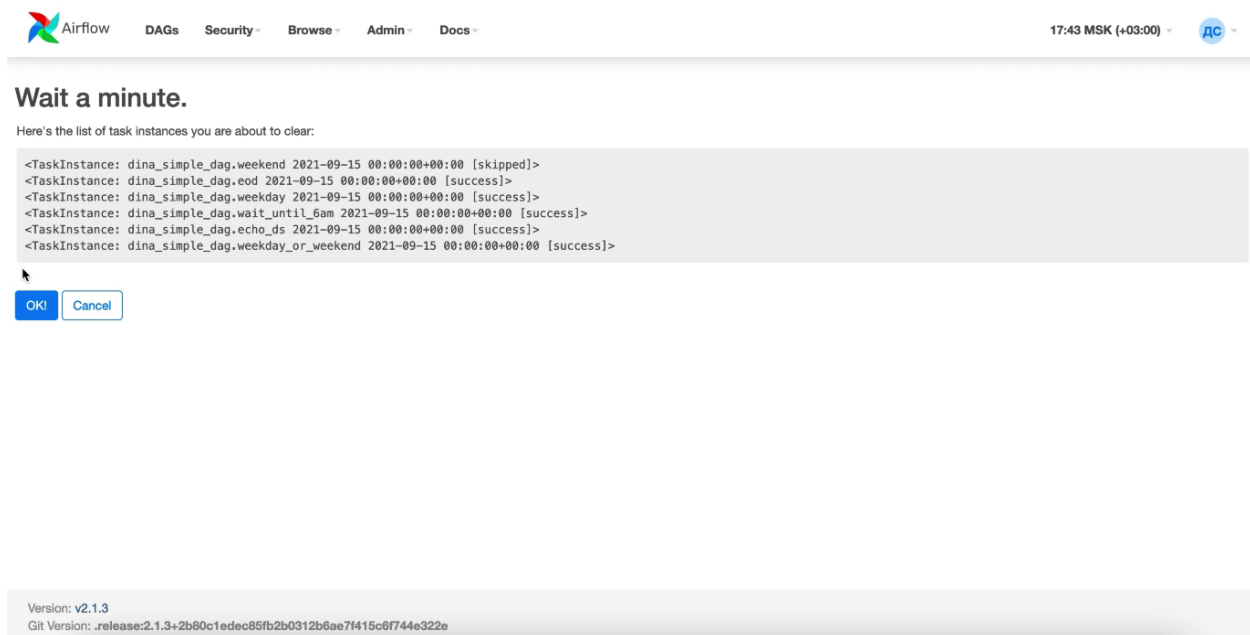


- Filter Upstream - отфильтрует DAG и выведет только ту часть, которая выполняется до выбранного таска
- Clear - очищает таски
- Mark Failed - помечается таски как упавшие
- Mark Success - помечает таски как удачно отработавшие
- Run - позволяет запустить таск вне очереди.

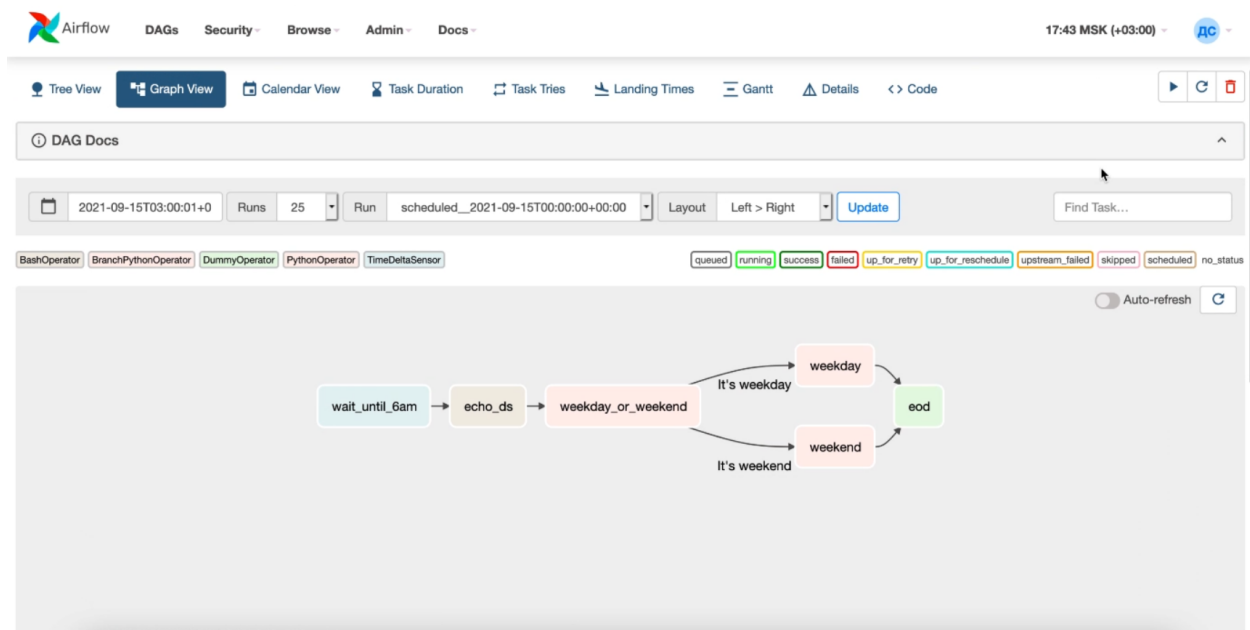
Выберем самый первый таск (wait_until_6am) и очистим его и все последующие за ним таски в DAG Run.



Airflow отображает список задач, которые будут очищены. Нажмем OK.



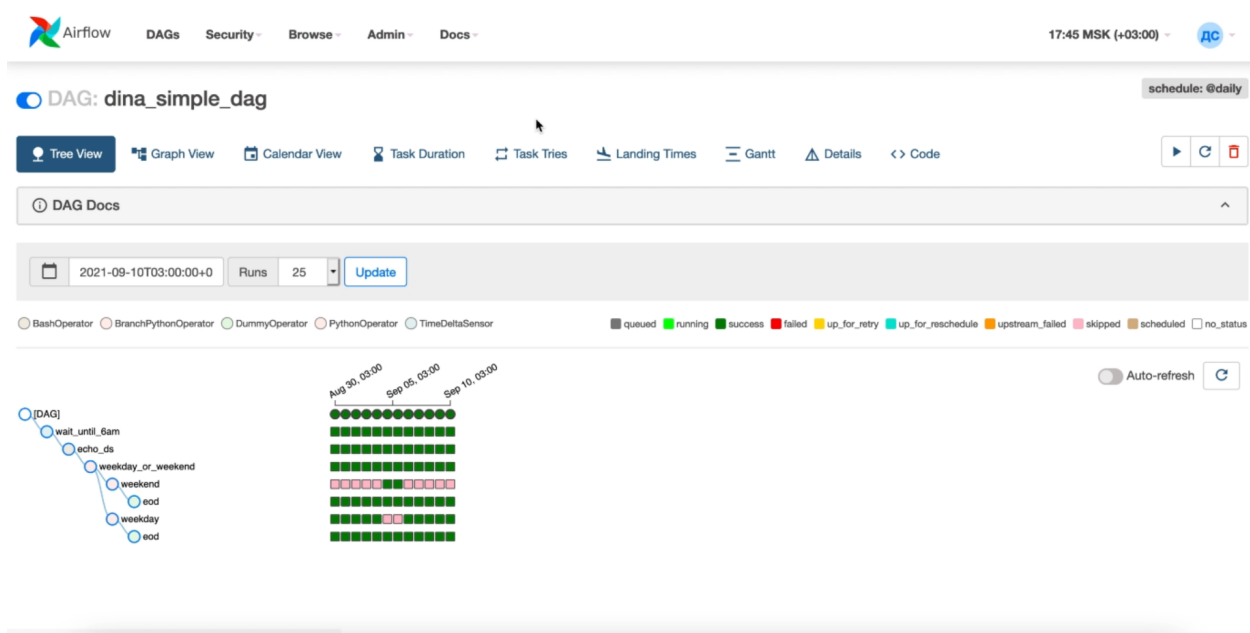
Видим, что задачи очистились. Если включить Auto-refresh, то можно увидеть, как будут последовательно обрабатываться задачи.



В этом интерфейсе так же отображается, какой цвет используется, для каких типов операторов/сенсоров (например, `wait_until_6am` - TimeDeltaSensor, а `echo_ds` - BashOperator). И цвета рамок, которые указывают на статусы выполнения задач.

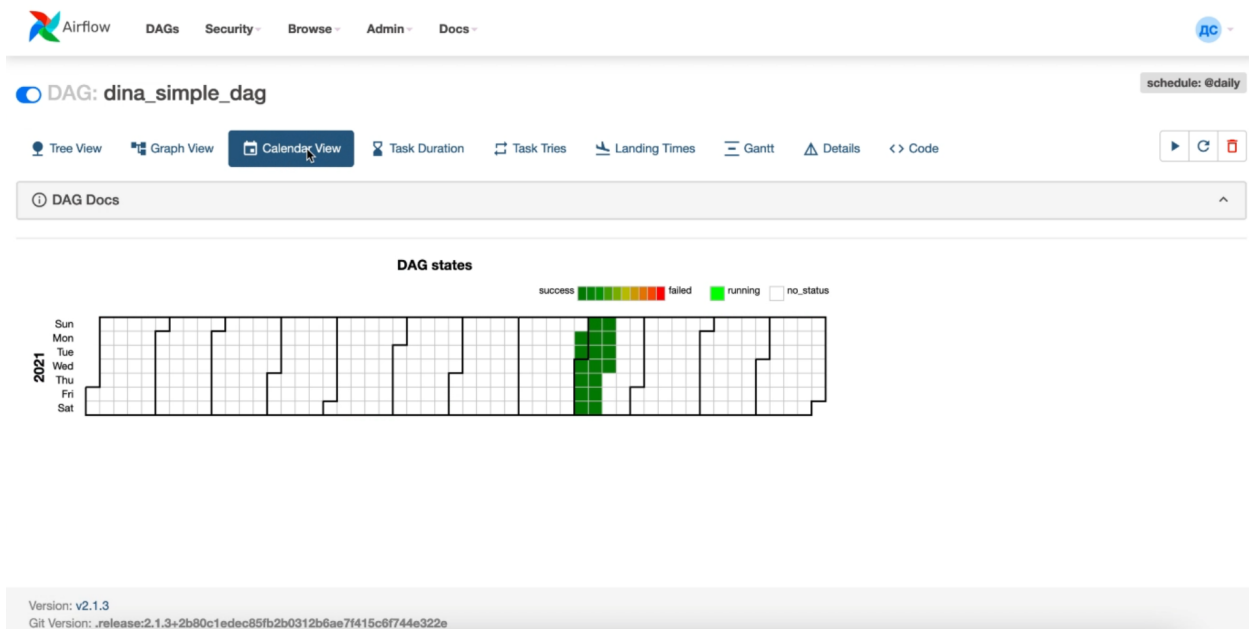
В верхней панели мы можем выбрать конкретный DAG RUN, настроить базовую дату и выбрать вид отображения DAG.

Tree View



Интерфейс, в котором DAG-и отображаются в виде дерева.

Calendar View



В календаре мы видим статусы выполнения DAG-а по дням.

Перейдем на вкладку **Browse** → **DAG Runs**.

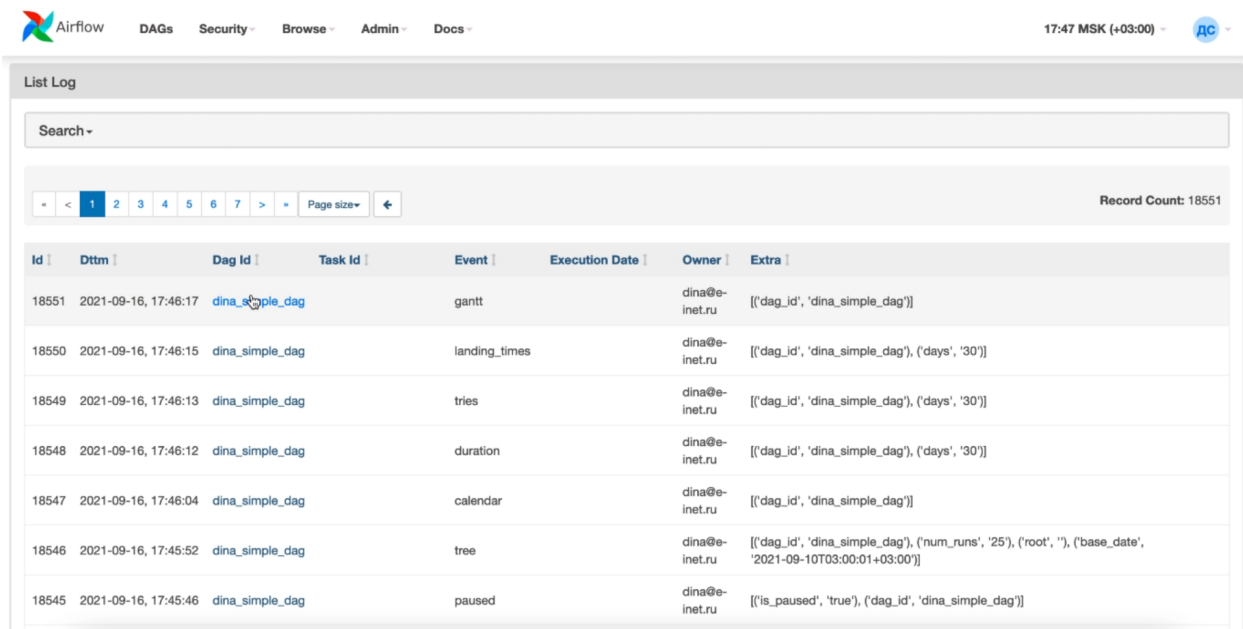
The screenshot shows the 'List Dag Run' view in the Airflow web interface. It displays a table of DAG runs for the 'dina_simple_dag' DAG. The table has columns for State, Dag Id, Execution Date, Run Id, Run Type, Queued At, Start Date, End Date, and External Trig. The runs are listed in descending order of execution date. The first three runs are highlighted in blue, indicating they are in a 'success' state. The interface includes a search bar at the top and a 'Record Count: 52' indicator.

State	Dag Id	Execution Date	Run Id	Run Type	Queued At	Start Date	End Date	External Trig
success	dina_simple_dag_v3	2021-09-15, 03:00:00	scheduled__2021-09-15T00:00:00+00:00	scheduled	2021-09-16 00:00:01.076735+00:00	2021-09-16, 03:00:01	2021-09-16, 09:00:49	False
success	dina_simple_dag_v2	2021-09-15, 03:00:00	scheduled__2021-09-15T00:00:00+00:00	scheduled	2021-09-16 00:00:01.049330+00:00	2021-09-16, 03:00:01	2021-09-16, 09:00:49	False
success	dina_simple_dag	2021-09-15, 03:00:00	scheduled__2021-09-15T00:00:00+00:00	scheduled	2021-09-16 14:43:34.574843+00:00	2021-09-16, 17:43:35	2021-09-16, 17:43:50	False
success	dina_simple_dag_v3	2021-09-14, 03:00:00	scheduled__2021-09-14T00:00:00+00:00	scheduled	2021-09-15 00:00:00.700036+00:00	2021-09-15, 03:00:00	2021-09-15, 09:00:50	False
success	dina_simple_dag_v2	2021-09-14, 03:00:00	scheduled__2021-09-14T00:00:00+00:00	scheduled	2021-09-15 00:00:00.681249+00:00	2021-09-15, 03:00:00	2021-09-15, 09:00:50	False
success	dina_simple_dag	2021-09-14, 03:00:00	scheduled__2021-09-14T00:00:00+00:00	scheduled	2021-09-15 00:00:00.654811+00:00	2021-09-15, 03:00:00	2021-09-15, 09:00:50	False
success	dina_simple_dag_v3	2021-09-13, 03:00:00	scheduled__2021-09-13T00:00:00+00:00	scheduled	2021-09-14 00:00:00.065116+00:00	2021-09-14, 03:00:00	2021-09-14, 09:00:47	False
success	dina_simple_dag	2021-09-13, 03:00:00	scheduled__2021-09-13T00:00:00+00:00	scheduled	2021-09-14 00:00:00.065116+00:00	2021-09-14, 03:00:00	2021-09-14, 09:00:47	False

Здесь отображены все DAG Run-ы, которые есть в базе метаданных. Здесь можно увидеть упавшие инстансы, перезапустить их.

Похожий интерфейс есть и для задач ([Browse](#) → [Task Instances](#)).

Перейдем на вкладку [Browse](#) → [Audit Logs](#).



Id	Dttm	Dag Id	Task Id	Event	Execution Date	Owner	Extra
18551	2021-09-16, 17:46:17	dina_simple_dag		gantt		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}]
18550	2021-09-16, 17:46:15	dina_simple_dag		landing_times		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}, {"days", "30"}]
18549	2021-09-16, 17:46:13	dina_simple_dag		tries		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}, {"days", "30"}]
18548	2021-09-16, 17:46:12	dina_simple_dag		duration		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}, {"days", "30"}]
18547	2021-09-16, 17:46:04	dina_simple_dag		calendar		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}]
18546	2021-09-16, 17:45:52	dina_simple_dag		tree		dina@e-inet.ru	[{"dag_id", "dina_simple_dag"}, {"num_runs", "25"}, {"root", ""}, {"base_date", "2021-09-10T03:00:01+03:00"}]
18545	2021-09-16, 17:45:46	dina_simple_dag		paused		dina@e-inet.ru	[{"is_paused", "true"}, {"dag_id", "dina_simple_dag"}]

В этом логе отображены все действия в Airflow, их время, параметры и owner.

Перейдем на вкладку [Admin](#) → [Variables](#).

Added Row

Browse... No file selected. Import Variables

List Variable

Search

+ Actions

Record Count: 1

	Key	Val	Description	Is Encrypted
<input type="checkbox"/>	gv_karpov	{one': 1, 'two': 2}		False

Version: v2.1.3
Git Version: .release:2.1.3+2b80c1edec85fb2b0312b5ae7f415c6f744e322e

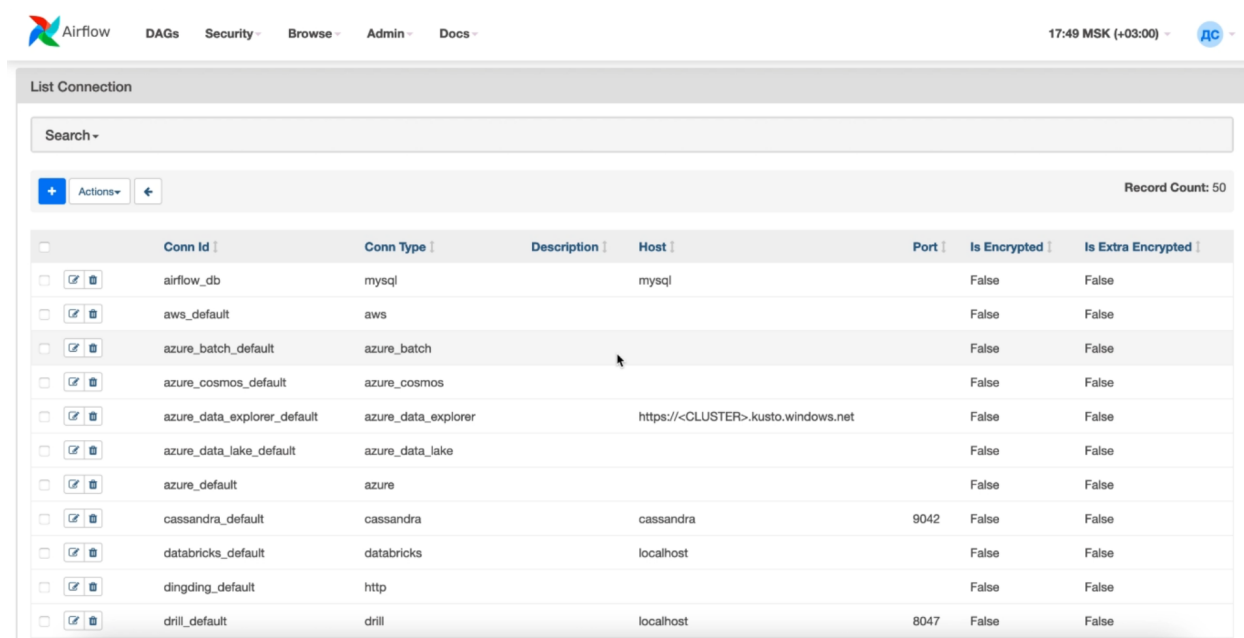
Здесь отображен список глобальных переменных. Такие переменные изменяются через интерфейс Airflow, их можно использовать в коде описания DAG-ов.

По клику на + можно добавить новую переменную. В значение мы можем записать строку, число, обычный список или словарь.

При использовании переменных нужно учитывать:

- в базе метаданных не хранится история изменений переменных
- при использовании переменной в коде, если на основании нее что-то изменяется внутри DAG-а, то каждый раз при обращении к папке с DAG-ами шедулер будет обращаться к метаданным, чтобы получить значение переменной.

Перейдем на вкладку **Admin** → **Connections**.



	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
<input type="checkbox"/>	airflow_db	mysql		mysql		False	False
<input type="checkbox"/>	aws_default	aws				False	False
<input type="checkbox"/>	azure_batch_default	azure_batch				False	False
<input type="checkbox"/>	azure_cosmos_default	azure_cosmos				False	False
<input type="checkbox"/>	azure_data_explorer_default	azure_data_explorer		https://<CLUSTER>.kusto.windows.net		False	False
<input type="checkbox"/>	azure_data_lake_default	azure_data_lake				False	False
<input type="checkbox"/>	azure_default	azure				False	False
<input type="checkbox"/>	cassandra_default	cassandra		cassandra	9042	False	False
<input type="checkbox"/>	databricks_default	databricks		localhost		False	False
<input type="checkbox"/>	dingding_default	http				False	False
<input type="checkbox"/>	drill_default	drill		localhost	8047	False	False

Здесь видим список всех соединений.

По клику на **+** можно добавить новое соединение. Airflow позволяет работать со множеством типов соединений.

Для работы с соединениями в Airflow используются хуки.

Хуки (hooks) - это интерфейсы для работы с различными внешними системами. Они позволяют писать код операторов универсально, то есть для обращения к какой-то базе нужно будет заменить только хук.

Перейдем на вкладку **Admin** → **Pools**.

Pool	Slots	Running Slots	Queued Slots
default_pool	128	0	0
dummy_pool	10	0	0
karpov_pool	10	0	0
student_pool	40	0	0

Version: v2.1.3
Git Version: .release:2.1.3+2b80c1edec85fb2b0312b5ae7f415c6f744e322e

Здесь видим набор пулов.

У executor-ов есть ограниченное количество воркеров. Если запущен некий не самый важные DAG, который генерирует слишком много задач одновременно, то такому DAG-у нужно ограничить ресурсы с помощью пулов, чтобы другие DAG-и так же могли запускаться.

Пул - это абстрактное ограничение на количество одновременно выполняемых задач.

Каждый пул имеет ограничение по количеству **слотов** - количество задач, которые одновременно могут быть запущены в пуле.

В этой вкладке мы видим список пулов, количество слотов, количество запущенных задач, и количество задач, которые стоят в очереди.

> Простой DAG

Airflow DAGs Security Browse Admin Docs 18:24 MSK (+03:00) ДС

DAG: dina_simple_dag success schedule: @daily

Tree View **Graph View** Calendar View Task Duration Task Tries Landing Times Gantt Details < > Code

DAG Docs

Это простейший даг. Он состоит из сенсора (ждёт 6ам) баш-оператора (выводит execution_date) оператора ветвления (в зависимости от execution_date выбирает следующий task) двух питон-операторов (выводят тип дня) dummy-оператора (ничего не делает и всё равно успешен)

2021-09-11T03:00:01+0 Runs 25 Run scheduled__2021-09-11T00:00:00+00:00 Layout Left > Right Update Find Task...

BashOperator BranchPythonOperator DummyOperator PythonOperator TimeDeltaSensor

queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled no_status

Auto-refresh

```

graph LR
    wait_until_6am[wait_until_6am] --> echo_ds[echo_ds]
    echo_ds --> weekday_or_weekend[weekday_or_weekend]
    weekday_or_weekend -- "It's weekday" --> weekday[weekday]
    weekday_or_weekend -- "It's weekend" --> weekend[weekend]
    weekday --> eod[eod]
    weekend --> eod[eod]
  
```

Рассмотрим пример DAG-а. Здесь мы видим:

1. `wait_until_6am` - сенсор, который ожидает наступления 6 утра по Гринвичу.
2. `echo_ds` - в логи записывает текущий `execution_date`
3. `weekday_or_weekend` - проверяет `execution_date` на соответствие критерию
 - a. `weekday` - записывает в лог `weekday`, если `execution_date` - будний день
 - b. `weekend` - записывает в лог `weekend`, если `execution_date` - выходной
4. `eod` - ничего не делает, нужен только для соединения задач (например, что бы другой DAG, смотря на этот оператор понимал, что DAG завершил работу)

Перейдем на табик **Code**.

[DAGs](#)
[Security](#)
[Browse](#)
[Admin](#)
[Docs](#)
18:44 MSK (+03:00)
ДС

DAG: dina_simple_dag
schedule: @daily

Tree View
Graph View
Calendar View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code

```

1  """
2  Это простейший dag. Он состоит из \
3  сенсора (ждёт бат) \
4  баш-оператора (выводит execution_date) \
5  оператора ветвления (в зависимости от execution_date выбирает следующий task) \
6  двух питон-операторов (выводят тип дня) \
7  dummy-оператора (ничего не делает и всё равно успешен)
8  """
9
10 from airflow import DAG
11 from airflow.utils.dates import days_ago
12 from datetime import datetime, timedelta
13 import logging
14
15 from airflow.operators.dummy import DummyOperator
16 from airflow.sensors.time_delta import TimeDeltaSensor
17 from airflow.operators.bash import BashOperator
18 from airflow.operators.python_operator import BranchPythonOperator
19 from airflow.operators.python_operator import PythonOperator
20 from airflow.utils.edgemodifier import Label
21
22 DEFAULT_ARGS = {
23     'start_date': days_ago(12),
24     'owner': 'Karpov',
25     'poke_interval': 600
26 }
27

```

Toggle Wrap

Здесь мы видим, как DAG описывается скриптом на python.

Сверху мы видим документацию к DAG-у, потому идут импорты, а затем дефолтные переменные (они относятся к DAG-у и часть из них используется задачами).

```

DEFAULT_ARGS = {
    'start_date': days_ago(12), # дата, с которой генерируются DAG Run-ы
    'owner': 'Karpov',
    'poke_interval': 600 # задает интервал перезапуска сенсоров (каждые 600 с.)
}

```

Создание DAG-а

Вариант 1 (с помощью контекстного менеджера):

```

with DAG(
    dag_id='dina_simple_dag',      # уникальный идентификатор DAG-а внутри Airflow
    schedule_interval='@daily',    # расписание
    default_args=DEFAULT_ARGS,     # дефолтные переменные
    max_active_runs=1,             # позволяет держать активным только один DAG Run
    tags=['karpov']                # тэги
) as dag:

    # описываем задачи и присваиваем их DAG-у

```

```
wait_until_6am = TimeDeltaSensor(
    task_id='wait_until_6am', # уникальный идентификатор задачи внутри DAG
    delta=timedelta(seconds=6*60*60) # время, которое мы ждем от запуска DAG
)
```

Вариант 2:

```
dag = DAG(
    'dina_simple_dag',
    schedule_interval='@daily',
    default_args=DEFAULT_ARGS,
    max_active_runs=1,
    tags=['karpov']
)

wait_until_6am = TimeDeltaSensor(
    task_id='wait_until_6am',
    delta=timedelta(seconds=6*60*60),
    dag=dag # в этом варианте нужно явно прописывать к какому DAG относится задача
)
```

Вариант 3 (с помощью декоратора)

```
@dag(
    start_date=days_ago(12),
    dag_id='dina_simple_dag',
    schedule_interval='@daily',
    default_args=DEFAULT_ARGS,
    max_active_runs=1,
    tags=['karpov']
)
def generate_dag():
    wait_until_6am = TimeDeltaSensor(
        task_id='wait_until_6am',
        delta=timedelta(seconds=6*60*60)
    )

dag = generate_dag()
```

здесь **TimeDeltaSensor** - ждет заданный промежуток времени после execution_date + schedule_interval задачи.

BashOperator

Выполняет заданный bash script

```
echo_ds = BashOperator(
    task_id='echo_ds',
    bash_command='echo {{ ds }}', # выполняемый bash script (ds = execution_date)
    dag=dag
)
```

BranchPythonOperator

Это оператор ветвления. В результате выполнения возвращает название следующей задачи. В нашем случае возвращает `weekend` или `weekday` в зависимости от `execution_dt`.

```
def select_day_func(**kwargs):
    execution_dt = kwargs['templates_dict']['execution_dt']
    exec_day = datetime.strptime(execution_dt, '%Y-%m-%d').weekday()
    return 'weekend' if exec_day in [5, 6] else 'weekday'

weekday_or_weekend = BranchPythonOperator(
    task_id='weekday_or_weekend',
    python_callable=select_day_func,
    templates_dict={'execution_dt': '{{ ds }}'},
    dag=dag
)
```

PythonOperator

```
def weekday_func():
    logging.info("It's weekday")

weekday = PythonOperator(
    task_id='weekday',
    python_callable=weekday_func, # ссылка на функцию, выполняемую в рамках задачи
    dag=dag
)
```

DummyOperator

```
eod = DummyOperator(
    task_id='eod',
    trigger_rule='one_success',
```

```
dag=dag
)
```

Оператор запускается в зависимости от заданного trigger rule.

Trigger Rule:

- `all_success` (по умолчанию) - task запустится в тот момент, когда все taskи, от которых он зависит, выполняются успешно
- `all_failed` - все taskи провалились
- `all_done` - все taskи пришли в конечное состояние (не `run` / `queued` / `retry`)
- `one_failed` - хотя бы один task упал
- `one_success` - хотя бы один task успешно отработал
- `none_failed` - ни одна задача не упала
- `none_failed_or_skipped` - ни одна задача не упала и не была пропущена
- `none_skipped` - ни одна задача не была пропущена
- `dummy` - task ни от чего не зависит, может стартовать в любое время

Составление пайплайнов

Громоздкий способ: для всех переменных, которые описывают taskи, определяем последователей (downstream) и предшественников (upstream). Их можно задавать по одному или же сразу подать список из нескольких задач (как для `eod`).

```
wait_until_6am.set_downstream(echo_ds)
echo_ds.set_downstream(weekday_or_weekend)
weekday_or_weekend.set_downstream(weekend, Label("It's weekend"))
weekday_or_weekend.set_downstream(weekday, Label("It's weekday"))
eod.set_upstream([weekend, weekday])
```

Наглядный и короткий способ. Здесь мы так же указываем лейблы (они маркируют ребра).

```
wait_until_6am >> echo_ds >> weekday_or_weekend >> Label("It's weekday") >> weekday >> eod
weekday_or_weekend >> Label("It's weekend") >> weekend >> eod
```

Самый короткий способ (без лейблов).

```
wait_until_6am >> echo_ds >> weekday_or_weekend >> [weekend, weekday] >> eod
```

Документация

Документацию можно писать не только для самого DAG-а, но и для отдельных задач:

```
dag.doc_md = __doc__
wait_until_6am.doc_md = """Сенсор. Ждёт наступление 6ам по Гринвичу"""
echo_ds.doc_md = """Пишет в лог execution_date"""
weekday_or_weekend.doc_md = """Выбирает ветку для исполнения в зависимости от дня недели"""
weekday.doc_md = """Пишет в лог 'It`s weekday'"""
weekend.doc_md = """Пишет в лог 'It`s weekend'"""
eod.doc_md = """Ничего не делает. trigger_rule='one_success'"""
```

Заданное описание можно будет увидеть при переходе на **Task instance details**:

The screenshot shows the Airflow web interface. At the top, there's a navigation bar with 'Airflow', 'DAGs', 'Security', 'Browse', 'Admin', and 'Docs'. The current page is 'Task Instance Details' for the task 'weekday_or_weekend' at 2021-09-11, 03:00:00. Below the navigation bar, there are tabs: 'Task Instance Details' (selected), 'Rendered Template', 'Log', and 'XCom'. The main content area is titled 'Task Instance Details' and shows 'Dependencies Blocking Task From Getting Scheduled'. There are two dependency rows:

Dependency	Reason
Dagrun Running	Task instance's dagrun was not in the 'running' state but in the state 'success'.
Task Instance State	Task is in the 'success' state which is not a valid state for execution. The task must be cleared in order to be run.

Below the dependencies, there's a section for 'Attribute: doc_md' with the value 'Выбирает ветку для исполнения в зависимости от дня недели'. Then, there's a table for 'Task Instance Attributes':

Attribute	Value
dag_id	dina_simple_dag
dag_model	None
dag_run	None
duration	0.395496
end_date	2021-09-12 06:00:15.549824+00:00
execution_date	2021-09-11T00:00:00+00:00

> Глоссарий

DAG (Directed Acyclic Graph) - направленный ациклический граф. Это граф, в котором нет закливания. В нем могут быть разветвления, но в конце они

сходятся в одной точке.

Оператор - описание действия, которое выполняет конкретная задача.

Сенсор - таск, ожидающий наступления какого-либо события.

DAG Run - экземпляр DAG-а.

Task Instance - экземпляр задачи.

Sheduler - планировщик, который анализирует DAG-и, создает их инстансы, инстансы задач и ставит задачи в очередь.

Executor - механизм, с помощью которого запускаются экземпляры задач.

Worker - процесс, в котором исполняются задачи.

Metadata Database - база метаданных, в которой хранится информация о состоянии всех пайплайнов.

DAG Directory - папка, где лежат описания DAG-ов в виде питоновских скриптов.

Хуки (hooks) - это интерфейсы для работы с различными внешними системами.

Пул - это абстрактное ограничение на количество одновременно выполняемых задач.