Apache Airflow

Agenda (Часть 1)

1. Что такое Apache Airflow?
2. Что такое Workflow Manager (orchestrator) и что такое ETL
3. Серверные компоненты и базовая установка, cli
4. DAG, basic DAG params & DAGFile, Tasks
5. Экземпляр запуска и задачи DAG
6. Operators, Sensors
7. Schedule interval & catch up & execution date
8. Junja2 Шаблоны,
9. Task Statuses

Agenda (Часть 2)

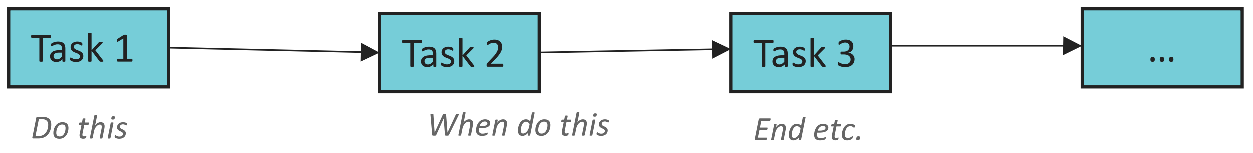
1. Macros, User Defined Marcos, Xcom
2. SLAs, Alerts, Retries
3. BranchOperator, TriggerRules
4. Hooks, Connections
5. Executors
6. Configuration (let’s add Celery Executor & PostgreSQL)
7. Workers & Flower
8. Variables, Run DAG with Params
9. Customization: UI plugins
10. Airflow in clouds: Google Compose (Airflow in GCP), Astronomer.io

Что такое Apache Airflow?

Что такое Apache Airflow?

Это менеджер рабочего процесса!
(или оркестратор)

Что такое рабочий процесс, конвейер или DAG (направленный ациклический граф)



workflow or pipeline or DAG (Direct Acyclic Graph)

Alternatives (Orchestrators)

OOZIE

Luigi

APACHE nifi

PREFECT

For DS & ML

mlflow™

METAFLOW

Similar things for Ops/DevOps/non data workflows

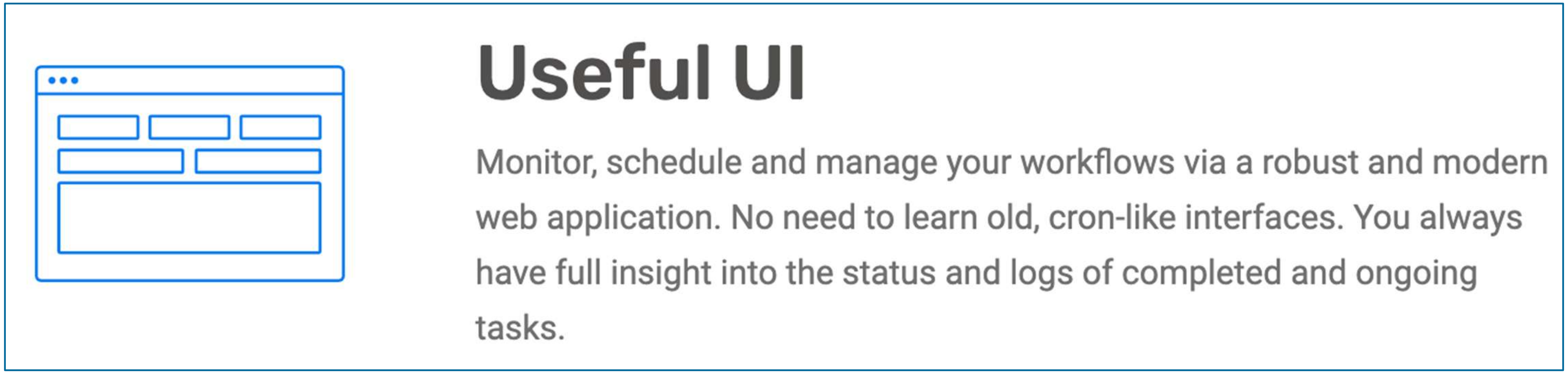
StackStorm

n8n.io

and etc.

circleci

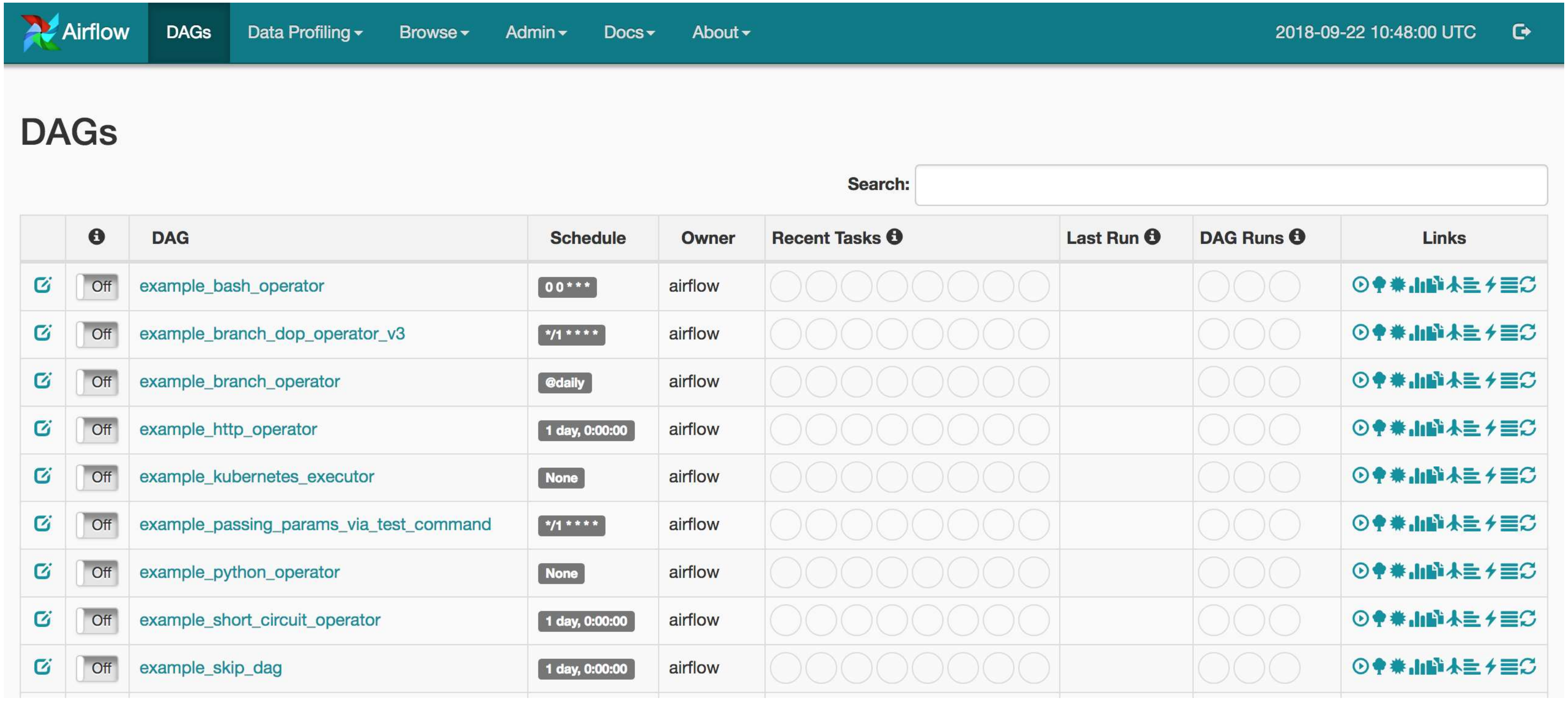
Main Features



Useful UI

Monitor, schedule and manage your workflows via a robust and modern web application. No need to learn old, cron-like interfaces. You always have full insight into the status and logs of completed and ongoing tasks.

UI Screens – DAGs List

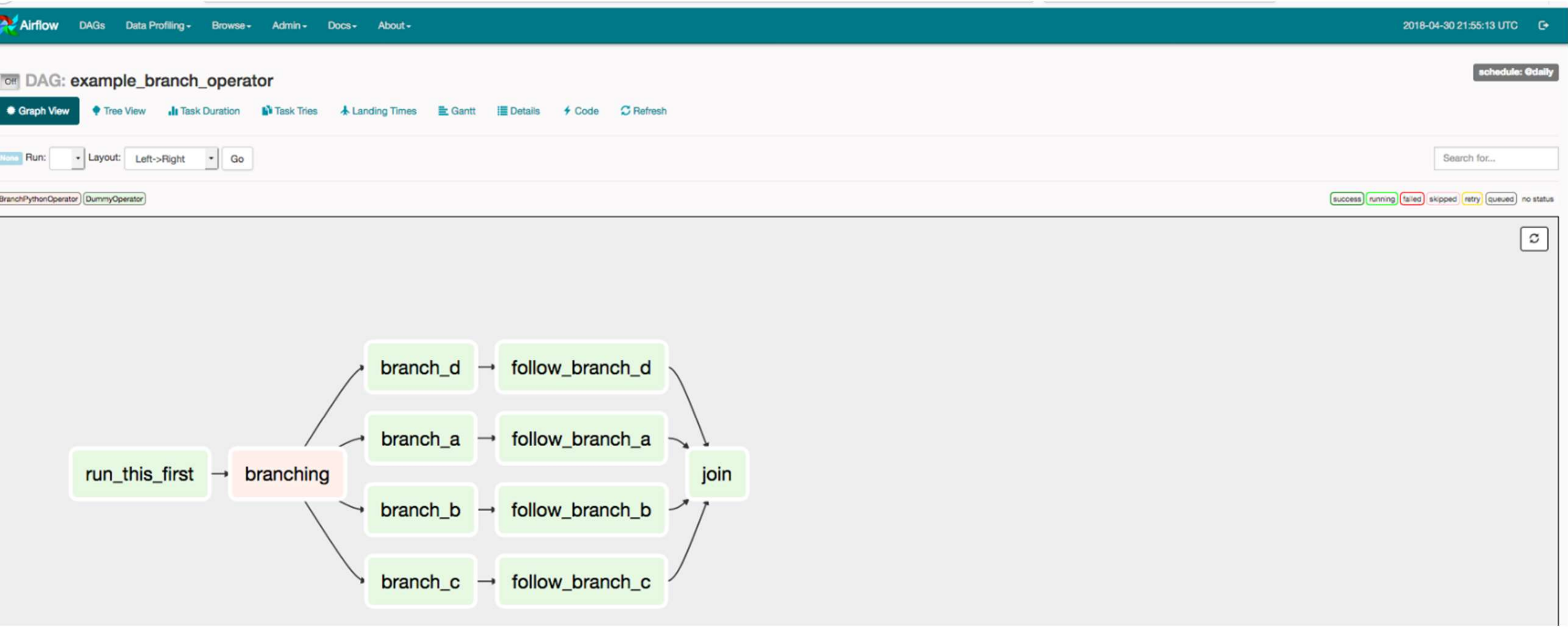


DAGs

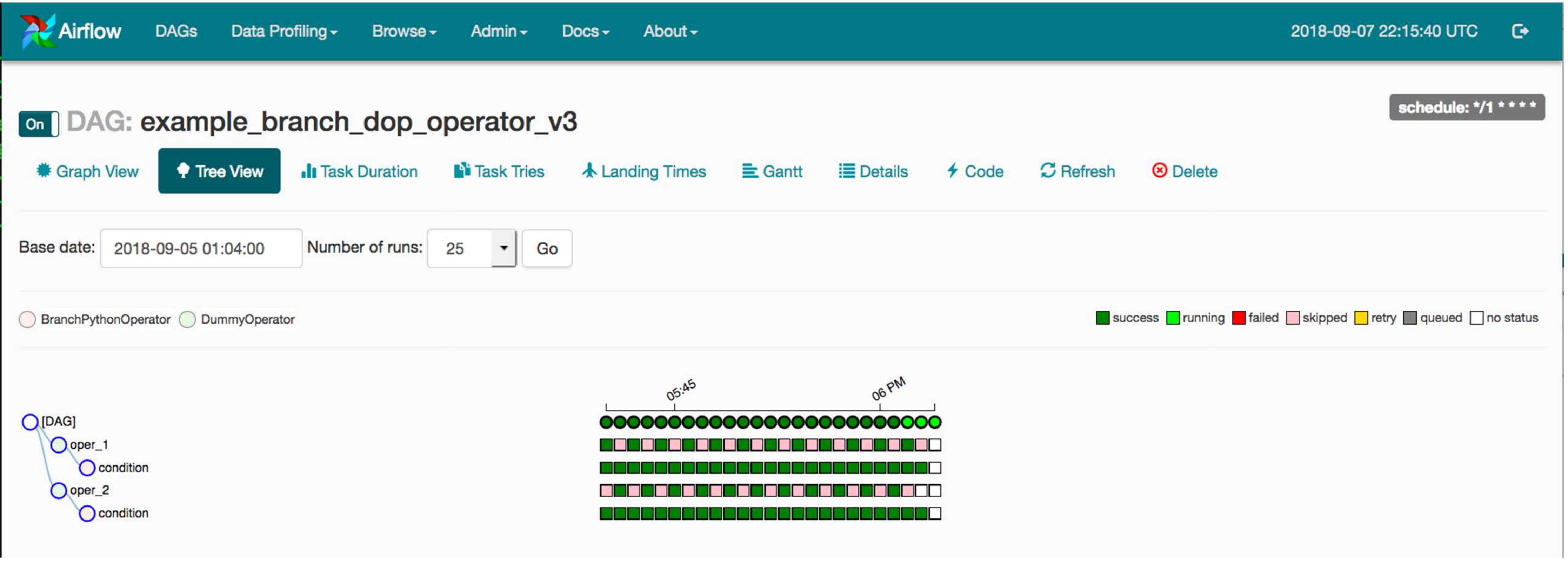
Search:

	ℹ	DAG	Schedule	Owner	Recent Tasks ℹ	Last Run ℹ	DAG Runs ℹ	Links
	Off	example_bash_operator	0 0 * * *	airflow				
	Off	example_branch_dop_operator_v3	*/1 * * * *	airflow				
	Off	example_branch_operator	@daily	airflow				
	Off	example_http_operator	1 day, 0:00:00	airflow				
	Off	example_kubernetes_executor	None	airflow				
	Off	example_passing_params_via_test_command	*/1 * * * *	airflow				
	Off	example_python_operator	None	airflow				
	Off	example_short_circuit_operator	1 day, 0:00:00	airflow				
	Off	example_skip_dag	1 day, 0:00:00	airflow				

UI Screens – DAGs View



UI Screens – DAGs Runs, Tree View



On DAG: example_branch_dop_operator_v3

schedule: */1 * * * *

Base date: 2018-09-05 01:04:00 Number of runs: 25

BranchPythonOperator DummyOperator

🟩 success 🟩 running 🟥 failed ▢ skipped 🟨 retry ▨ queued ☐ no status

[DAG]
oper_1
condition
oper_2
condition

Main Features

Useful UI

Monitor, schedule and manage your workflows via a robust and modern web application. No need to learn old, cron-like interfaces. You always have full insight into the status and logs of completed and ongoing tasks.

Robust Integrations

Airflow provides many plug-and-play operators that are ready to execute your tasks on Google Cloud Platform, Amazon Web Services, Microsoft Azure and many other third-party services. This makes Airflow easy to apply to current infrastructure and extend to next-gen technologies.

Main Features

Integrations from the box (*Operators, Sensors, Connectors & Hooks*)

amazon web services

Google Cloud Platform

PostgreSQL

slack

Azure

druid

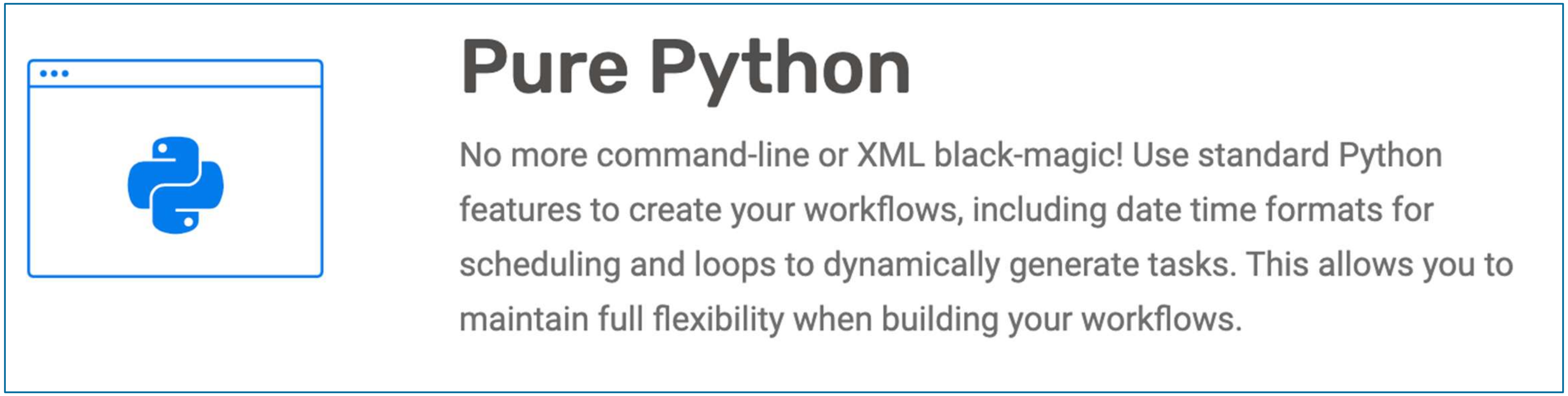
hadoop

MySQL

ORACLE

https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html
https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/contrib/operators/

Main Features



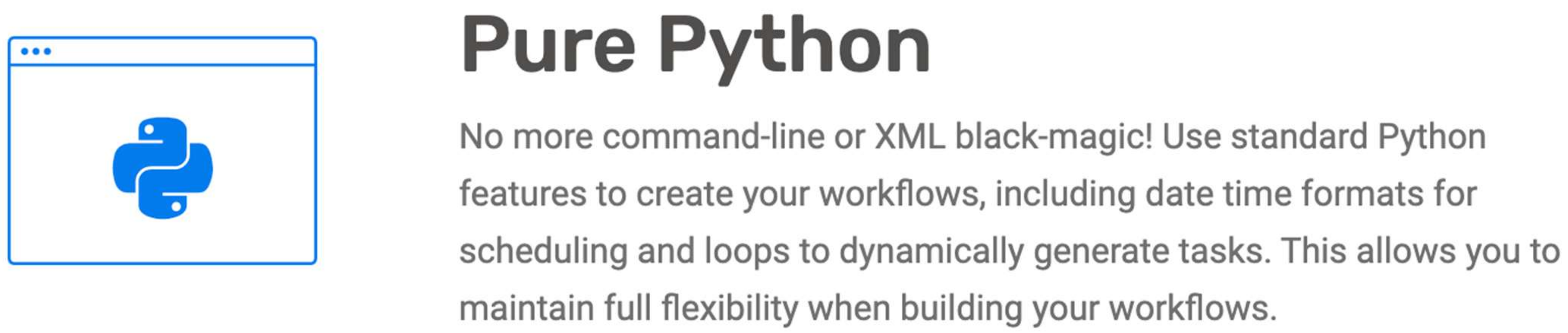
Pure Python

No more command-line or XML black-magic! Use standard Python features to create your workflows, including date time formats for scheduling and loops to dynamically generate tasks. This allows you to maintain full flexibility when building your workflows.

DAG Code Example

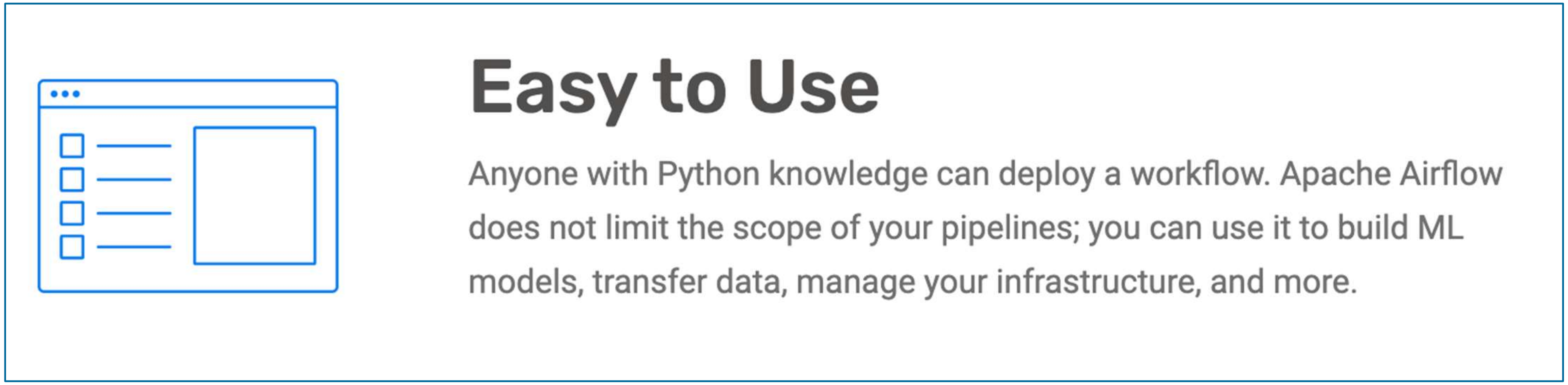
```python
import uuid
from datetime import datetime
from airflow import DAG
from airflow.utils.trigger_rule import TriggerRule
from airflow.operators.postgres_operator import PostgresOperator


dag_params = {
    'dag_id': 'PostgresOperator_dag',
    'start_date': datetime(2019, 10, 7),
    'schedule_interval': None
}


with DAG(**dag_params) as dag:

    create_table = PostgresOperator(
        task_id='create_table',
        sql='''CREATE TABLE new_table(
            custom_id integer NOT NULL, timestamp TIMESTAMP NOT NULL, user_id VARCHAR (50) NOT NULL
            );''',
    )

    insert_row = PostgresOperator(
        task_id='insert_row',
        sql='INSERT INTO new_table VALUES(%s, %s, %s)',
        trigger_rule=TriggerRule.ALL_DONE,
        parameters=(uuid.uuid4().int % 123456789, datetime.now(), uuid.uuid4().hex[:10])
    )

    create_table >> insert_row
```

Main Features



Pure Python

No more command-line or XML black-magic! Use standard Python features to create your workflows, including date time formats for scheduling and loops to dynamically generate tasks. This allows you to maintain full flexibility when building your workflows.



Easy to Use

Anyone with Python knowledge can deploy a workflow. Apache Airflow does not limit the scope of your pipelines; you can use it to build ML models, transfer data, manage your infrastructure, and more.

Main Features

Pure Python

No more command-line or XML black-magic! Use standard Python features to create your workflows, including date time formats for scheduling and loops to dynamically generate tasks. This allows you to maintain full flexibility when building your workflows.

Easy to Use

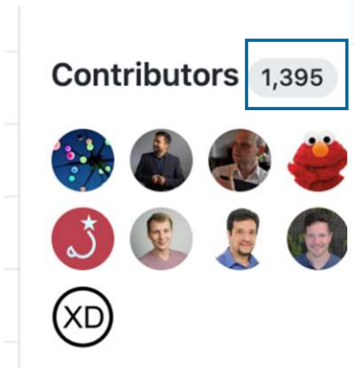
Anyone with Python knowledge can deploy a workflow. Apache Airflow does not limit the scope of your pipelines; you can use it to build ML models, transfer data, manage your infrastructure, and more.

Open Source

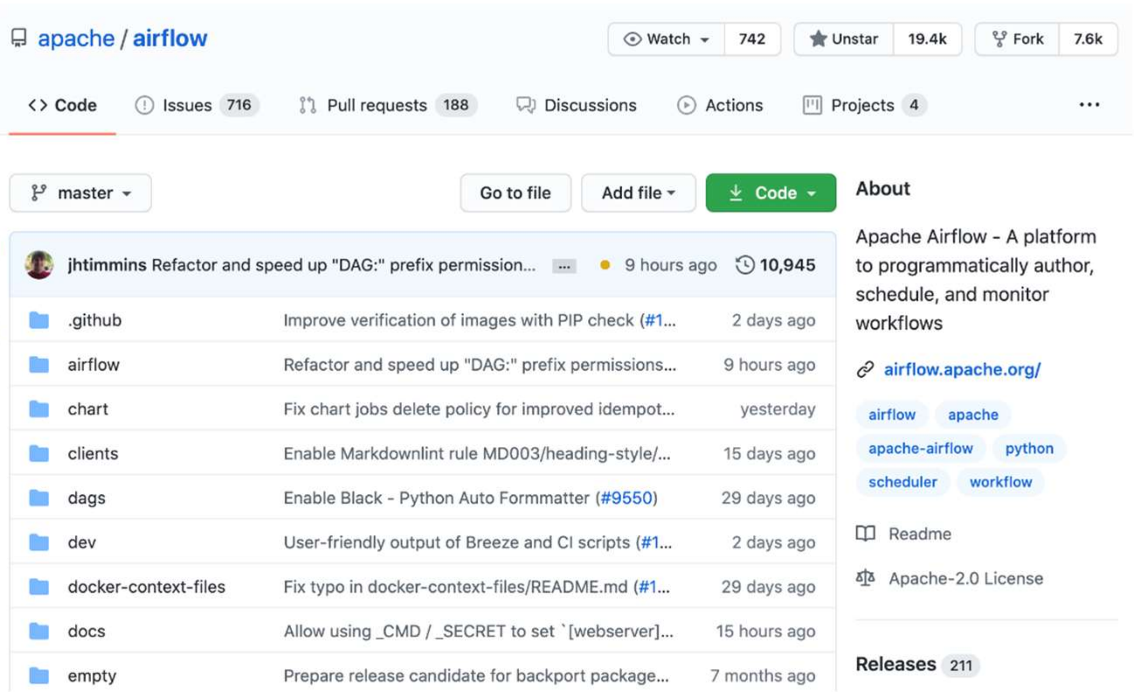
Wherever you want to share your improvement you can do this by opening a PR. It's simple as that, no barriers, no prolonged procedures. Airflow has many active users who willingly share their experiences. Have any questions? Check out our buzzing slack.

Apache Airflow Community

https://github.com/apache/airflow



Contributors 1,395



apache / airflow

Watch 742 | Unstar 19.4k | Fork 7.6k

Code | Issues 716 | Pull requests 188 | Discussions | Actions | Projects 4

master | Go to file | Add file | Code

jhtimmins Refactor and speed up "DAG:" prefix permission... 9 hours ago 10,945

.github	Improve verification of images with PIP check (#1...	2 days ago
airflow	Refactor and speed up "DAG:" prefix permissions...	9 hours ago
chart	Fix chart jobs delete policy for improved idempot...	yesterday
clients	Enable Markdownlint rule MD003/heading-style/...	15 days ago
dags	Enable Black - Python Auto Formmatter (#9550)	29 days ago
dev	User-friendly output of Breeze and CI scripts (#1...	2 days ago
docker-context-files	Fix typo in docker-context-files/README.md (#1...	29 days ago
docs	Allow using _CMD / _SECRET to set `[webserver]...	15 hours ago
empty	Prepare release candidate for backport package...	7 months ago

About

Apache Airflow - A platform to programmatically author, schedule, and monitor workflows

airflow.apache.org/

airflow apache apache-airflow python scheduler workflow

Readme

Apache-2.0 License

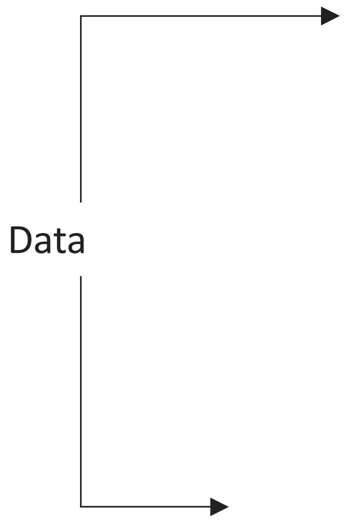
Releases 211

Official community Slack:
https://apache-airflow-slack.herokuapp.com/

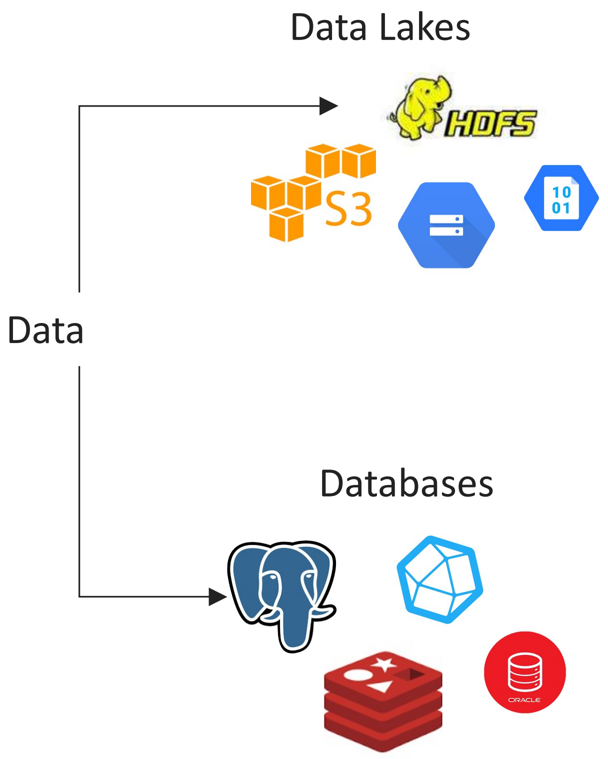
List of committers (maintainers):
https://people.apache.org/committers-by-project.html#airflow (about 40 people)

Что такое менеджер рабочего процесса? (или оркестратор)

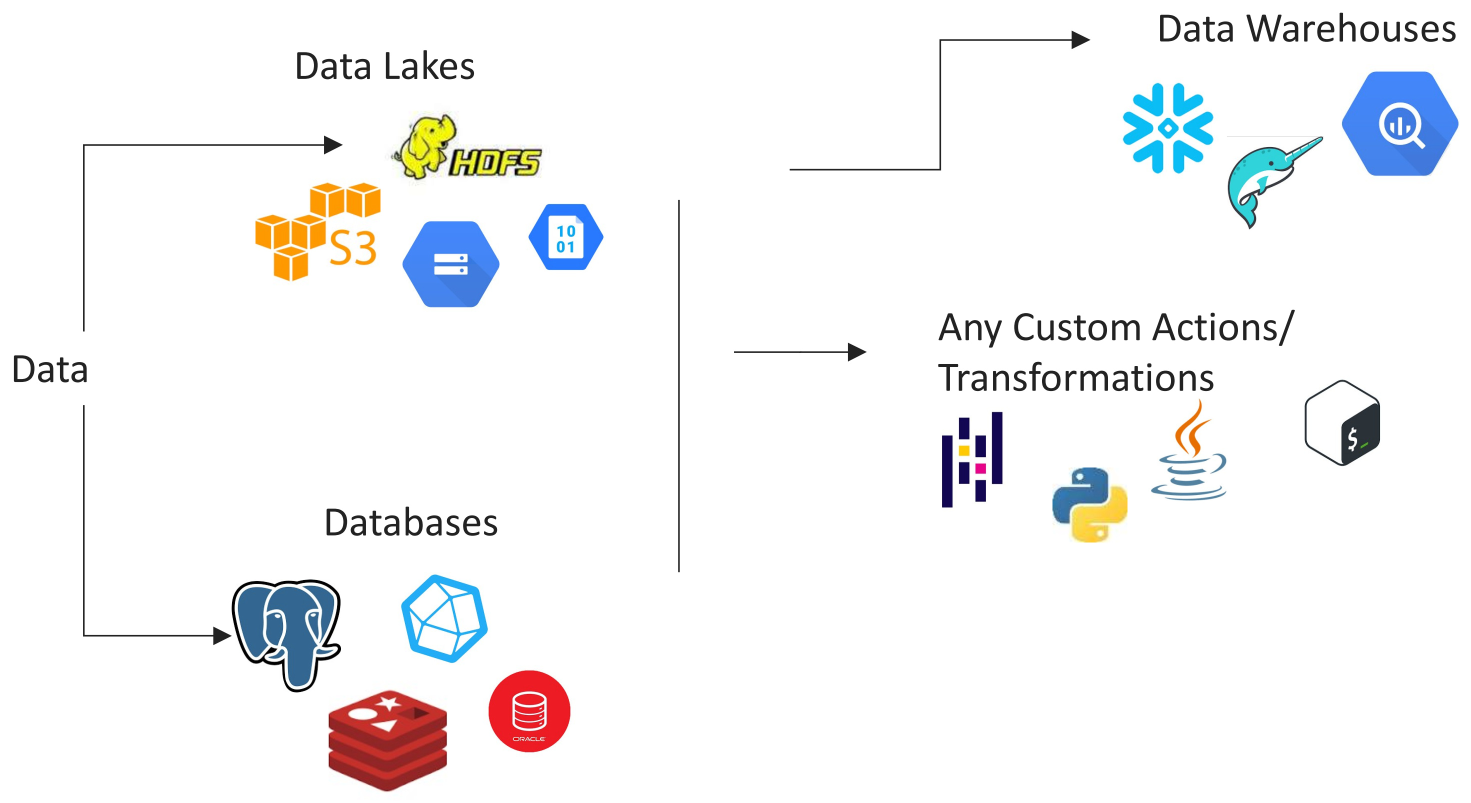
Data Flow



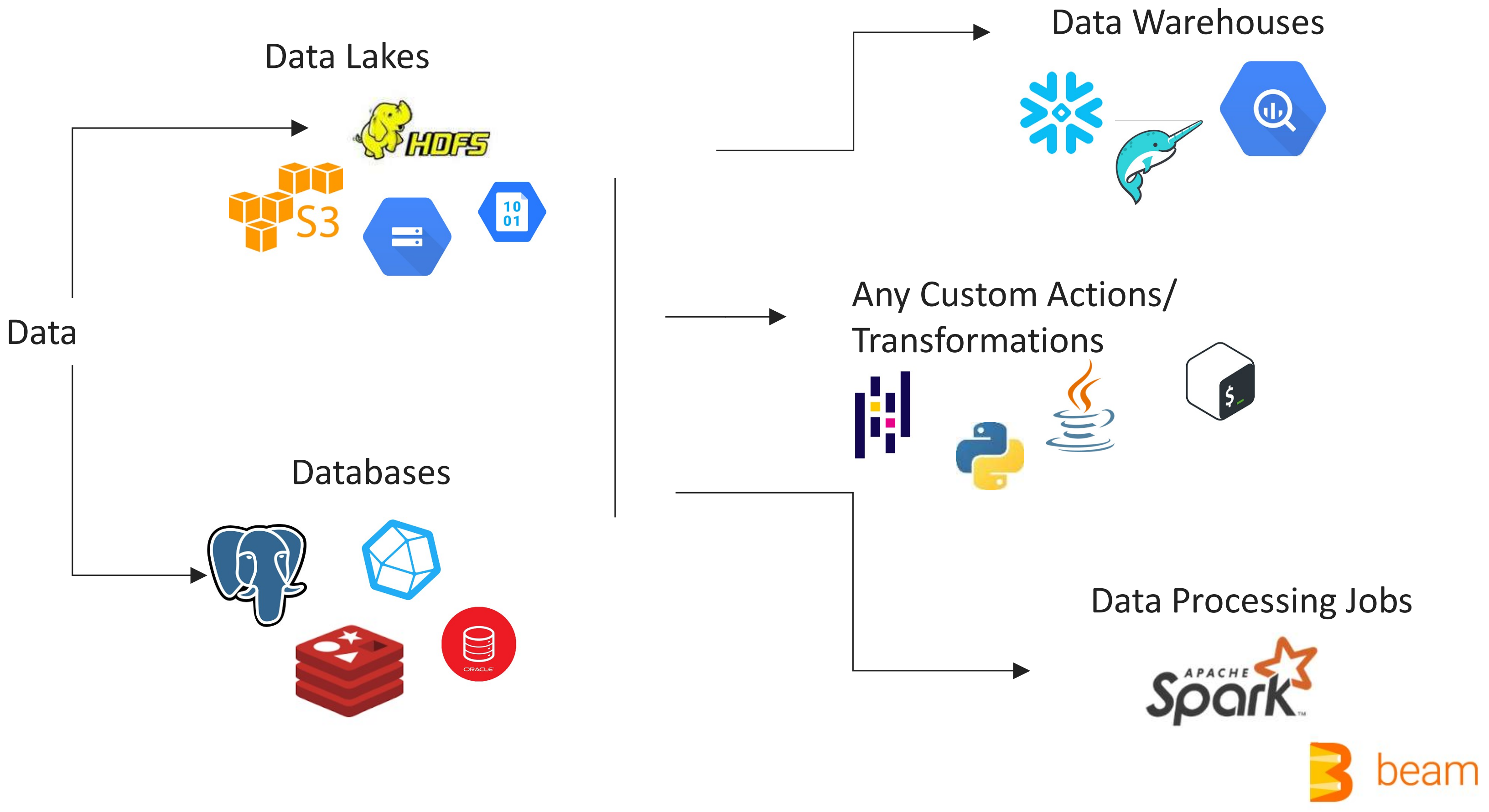
Data Flow



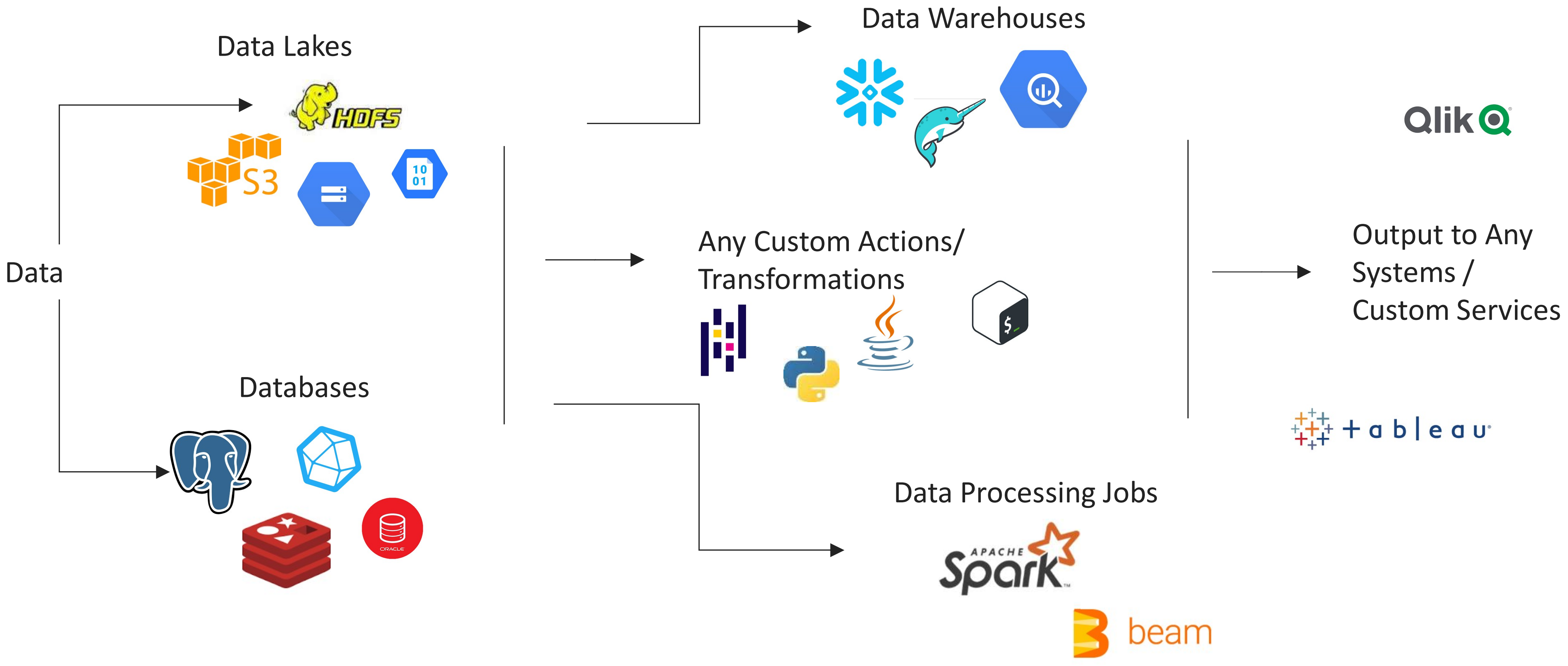
Data Flow


Data
Data Lakes
HDFS
S3
10
01
Databases
ORACLE
Data Warehouses
Any Custom Actions/
Transformations
$

Data Flow


Data
Data Lakes
HDFS
S3
10
01
Databases
ORACLE
Data Warehouses
Any Custom Actions/
Transformations
$
Data Processing Jobs
APACHE
Spark
beam

Data Flow


Data
Data Lakes
HDFS
S3
Databases
ORACLE
Data Warehouses
Any Custom Actions/
Transformations
Data Processing Jobs
APACHE Spark
beam
Qlik
Output to Any
Systems /
Custom Services
tableau

Оркестратор или менеджер рабочих процессов
Позволяет создавать конвейеры данных
& describe all steps of your Data Flow:

откуда куда, что, когда и как– многозадачность в любой последовательности
(не только классическая ETL)

Extract, transform, load

Extract

Data Lakes

Databases

Transform

Any Custom Actions/ Transformations

Data Processing Jobs

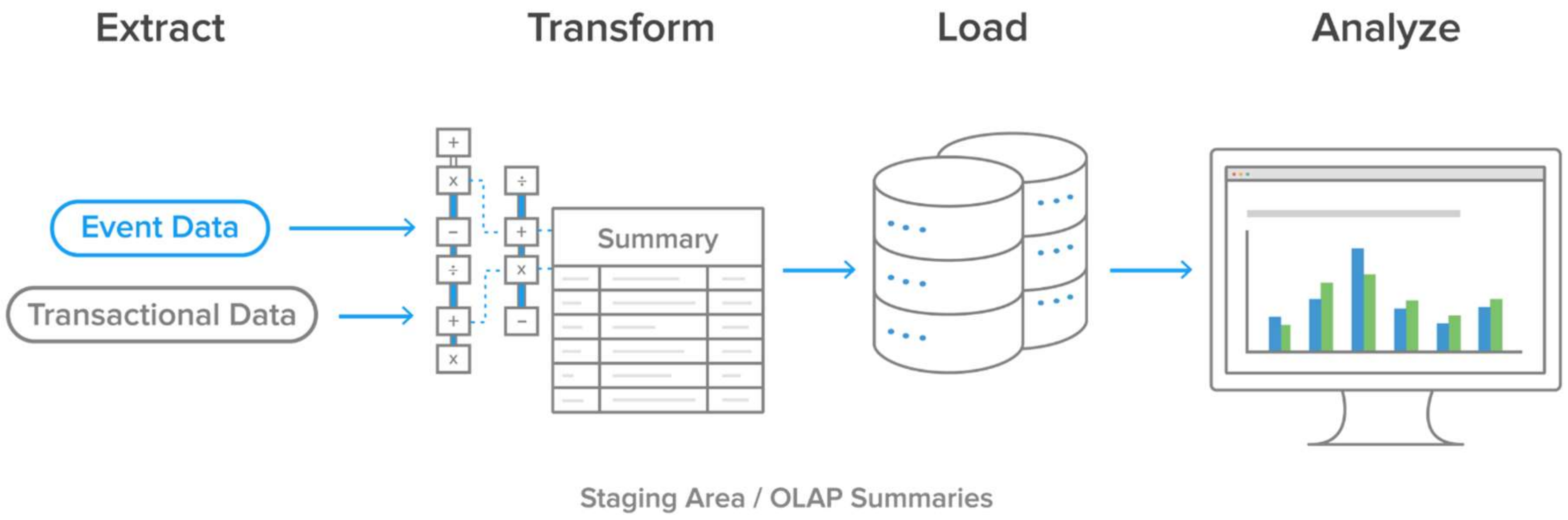
Load

Data Warehouses

Output to Any Systems / Custom Services

Возможности рабочего процесса, которые нам нужны

1. Monitoring Dashboard (что происходит с нашим конвейером?)
2. Alerts (если что-то не так - Я должен знать об этом быстро)
3. SLAs (если у нас нет данных за день - у нас проблемы?))
4. Way to make customization



Pipeline Example in Words

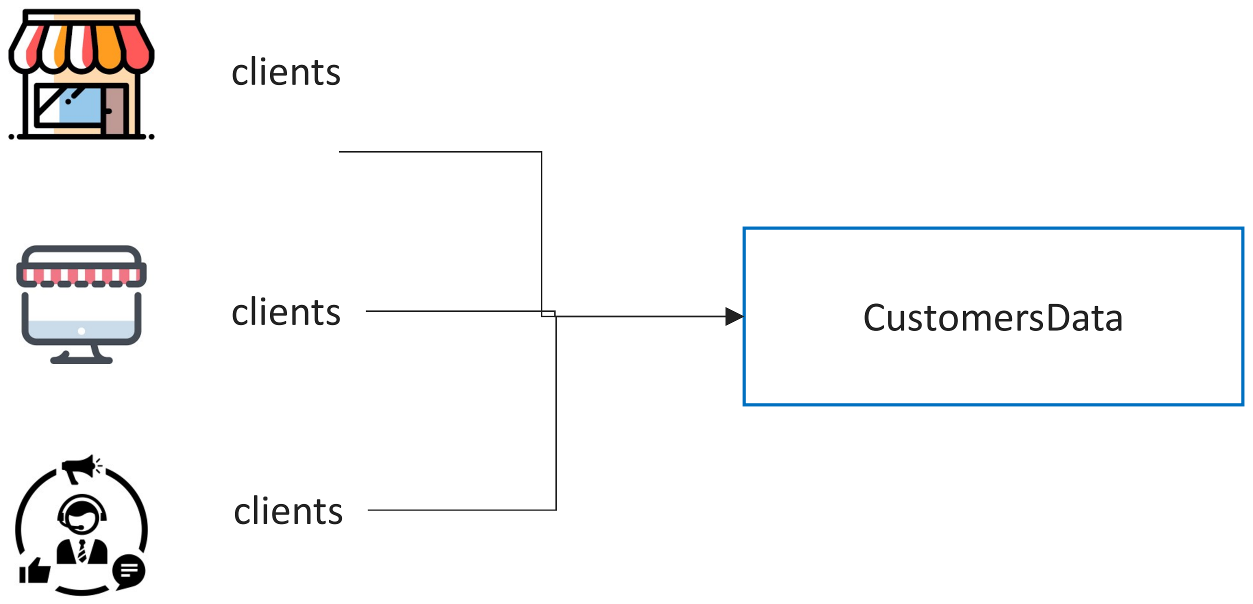
Мы работаем в команде Data Engineering в магазине канцелярских товаров (ручки, бумага и т. д.). У нас около 1500 офлайн-магазинов, интернет-магазин и прямые продажи.

Мы работаем над конвейером данных, который предполагает получение информации о наших клиентах из разных источников.

Pipeline Example in Words

Мы работаем в команде Data Engineering в магазине канцелярских товаров (ручки, бумага и т. д.). У нас около 1500 офлайн-магазинов, интернет-магазин и прямые продажи.

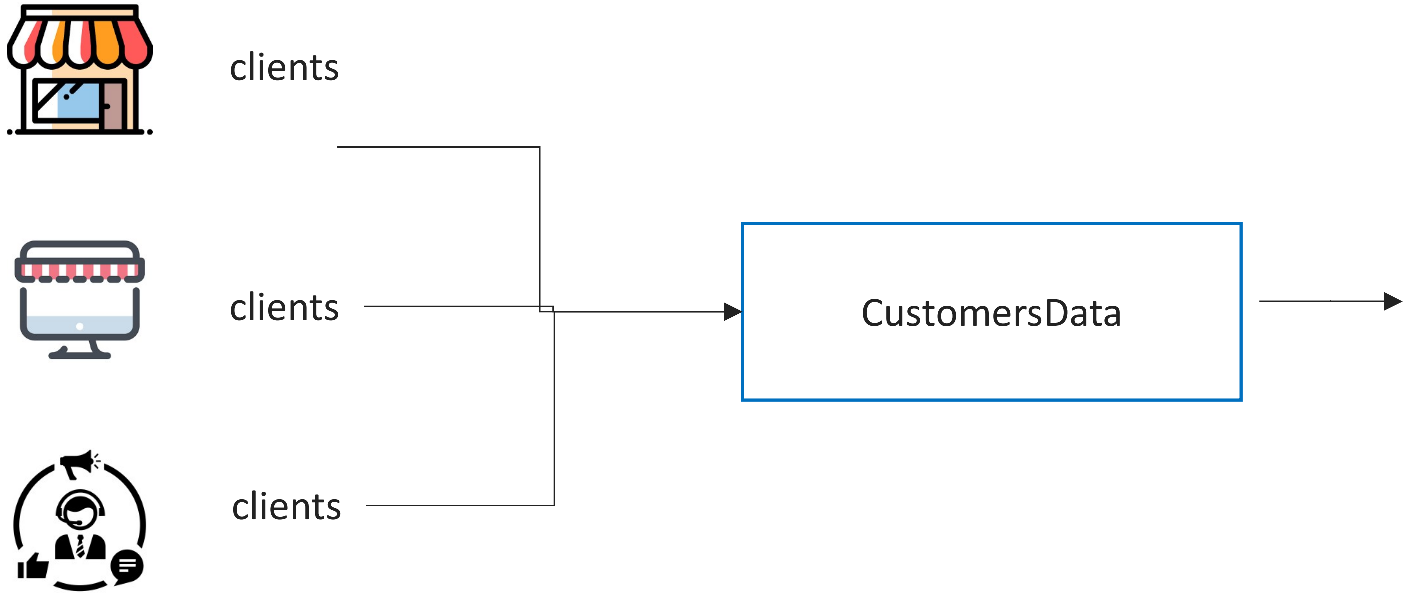
Мы работаем над конвейером данных, который предполагает получение информации о наших клиентах из разных источников.



Pipeline Example in Words

Мы работаем в команде Data Engineering в магазине канцелярских товаров (ручки, бумага и т. д.). У нас около 1500 офлайн-магазинов, интернет-магазин и прямые продажи.

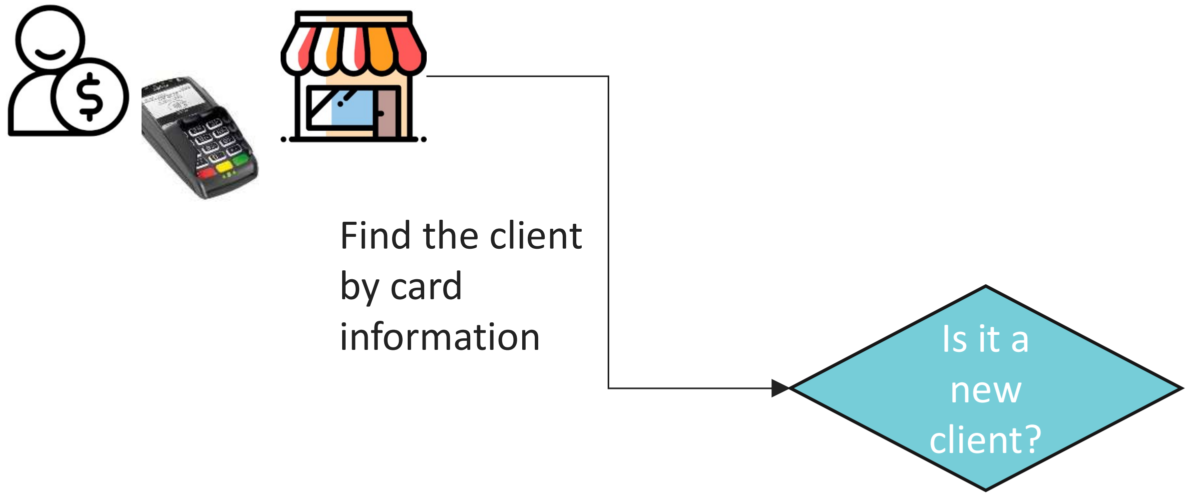
Мы работаем над конвейером данных, который предполагает получение информации о наших клиентах из разных источников.



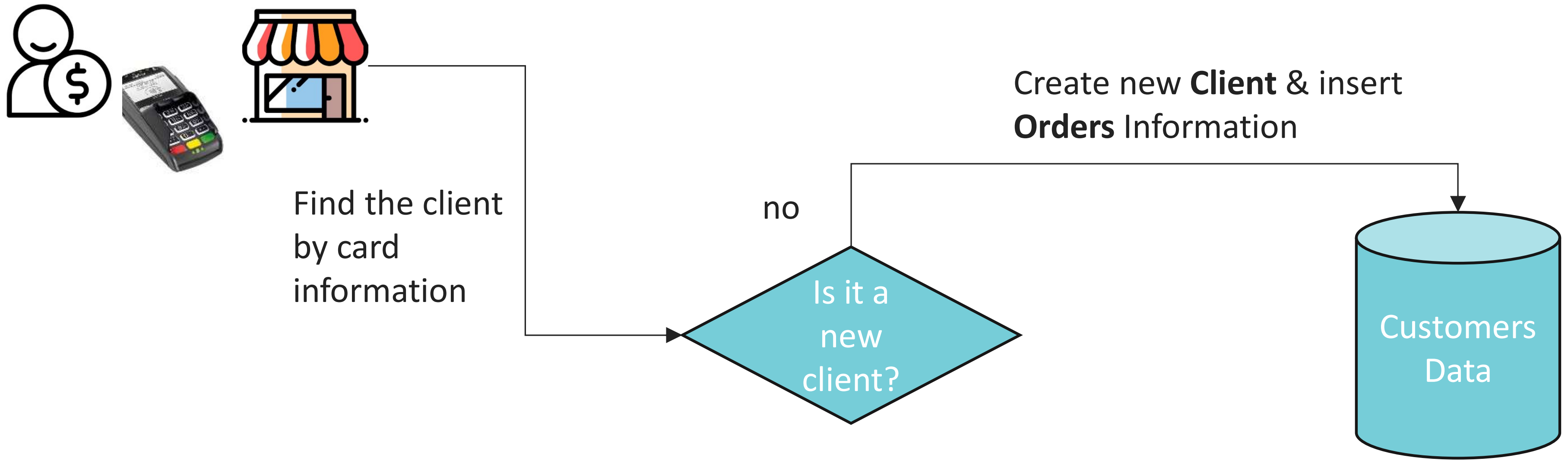
1. Это новый клиент?
2. Что мы уже знаем об этом клиенте?
3. Попробуйте сопоставить клиента по некоторым «критериям» на основе уже существующей информации.
4. Обновление данных (данные изменены на временной шкале — заказы, маркетинговая деятельность…)

Pipeline – find the client by credit card number

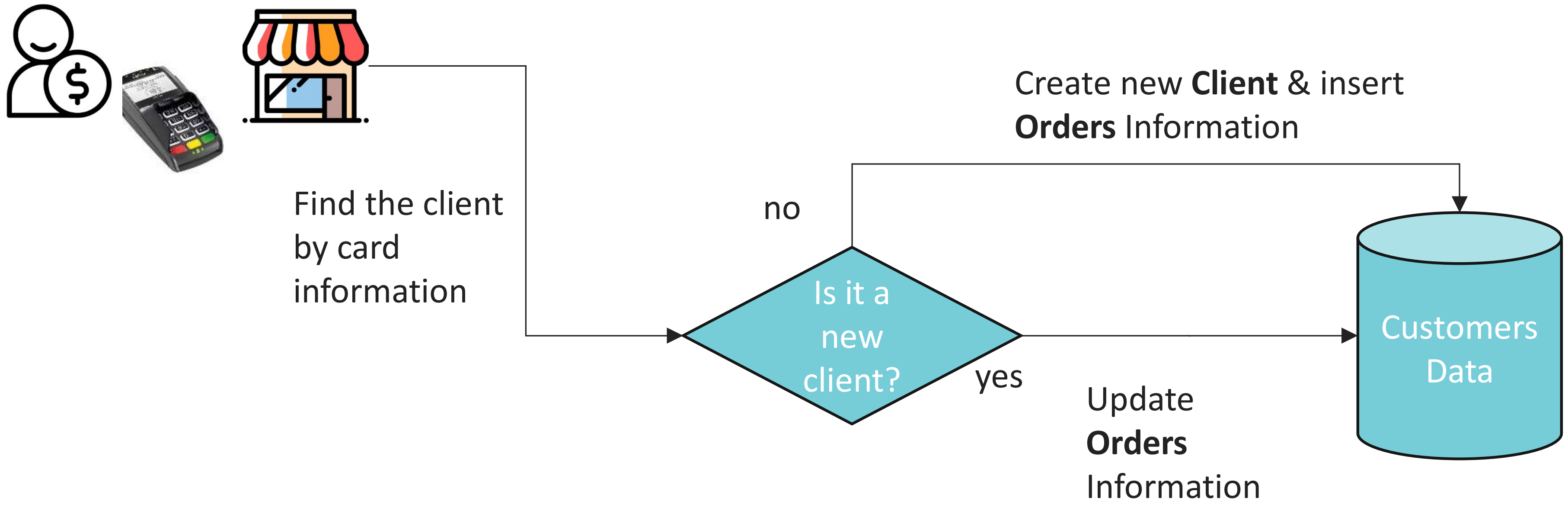
Pipeline – find the client by credit card number



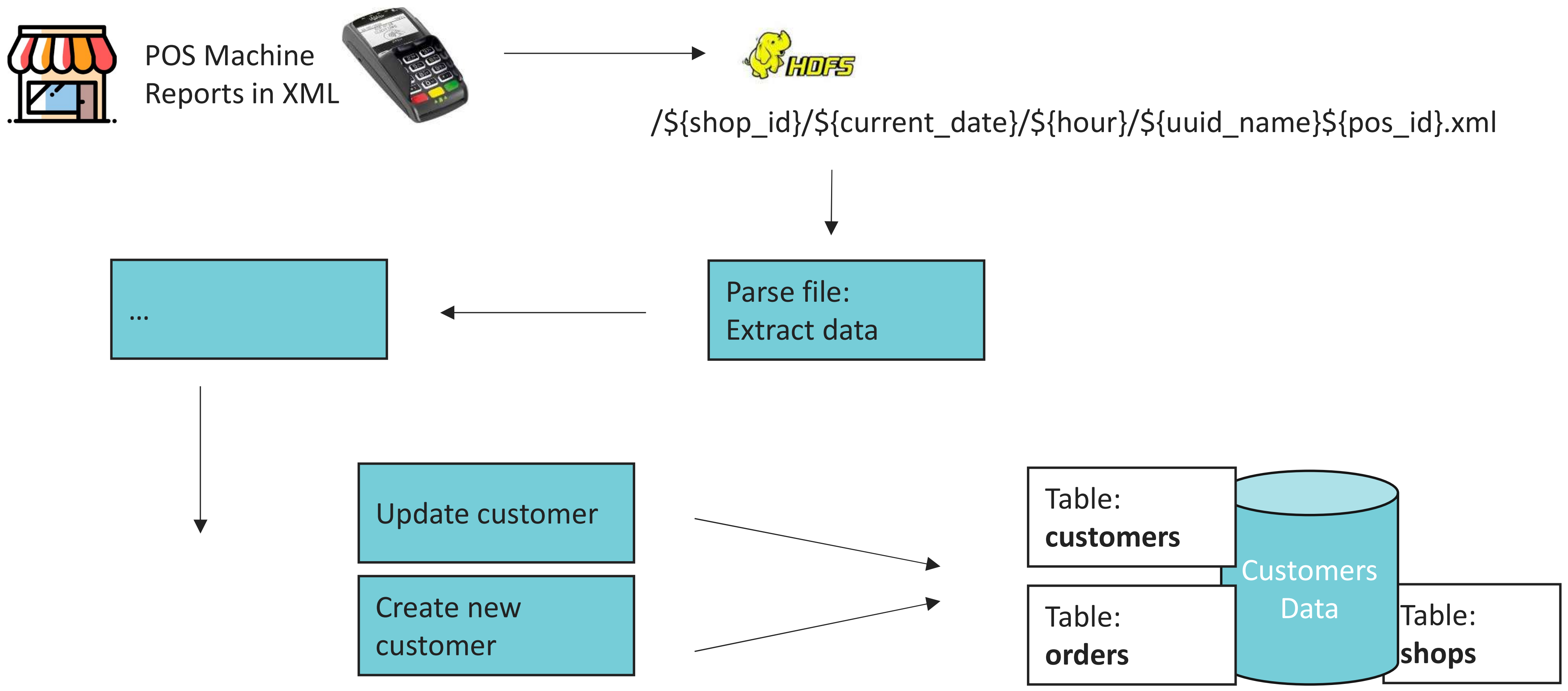
Pipeline – find the client by credit card number


Find the client by card information
no
Is it a new client?
Create new **Client** & insert **Orders** Information
Customers Data

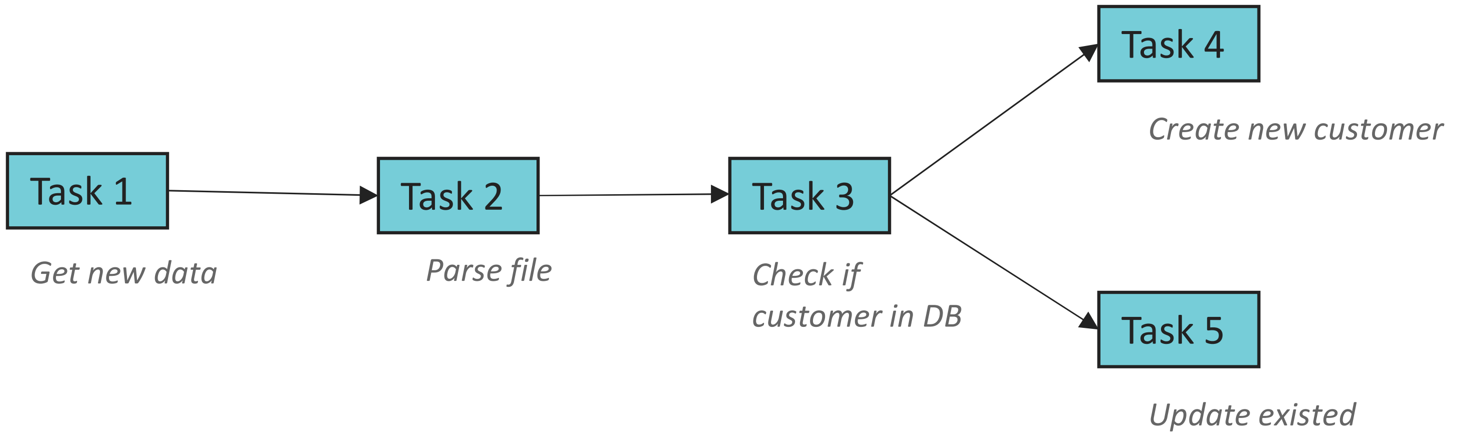
Pipeline – find the client by credit card number


Find the client by card information
no
Create new **Client** & insert **Orders** Information
Is it a new client?
yes
Update **Orders** Information
Customers Data

Pipeline – find the client by credit card number



Abstract visualization



Мы только что описали рабочий процесс или конвейер…

Or DAG – main concept of Apache Airflow

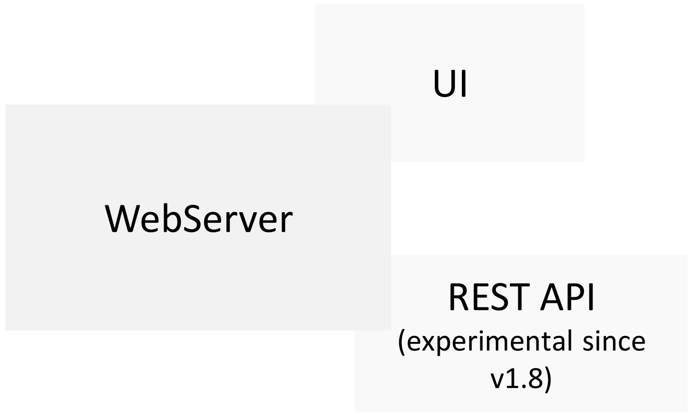
Some key characteristic of Pipelines

1. **Расписание:** они выполняются вовремя с разным расписанием, продолжительностью и т. д.
2. **Триггеры:** конвейеры могут иметь триггеры, вызывающие необходимость запуска конвейера.
3. **Сбой:** конвейеры могут выйти из строя. Нам нужно
    1. Как можно скорее узнать об этом
    2. запустить с момента сбоя —ваша задача должна быть атомарной и простой.
4. **Повторная обработка:** иногда вам нужно повторно обрабатывать данные за целые длительные периоды в прошлом.

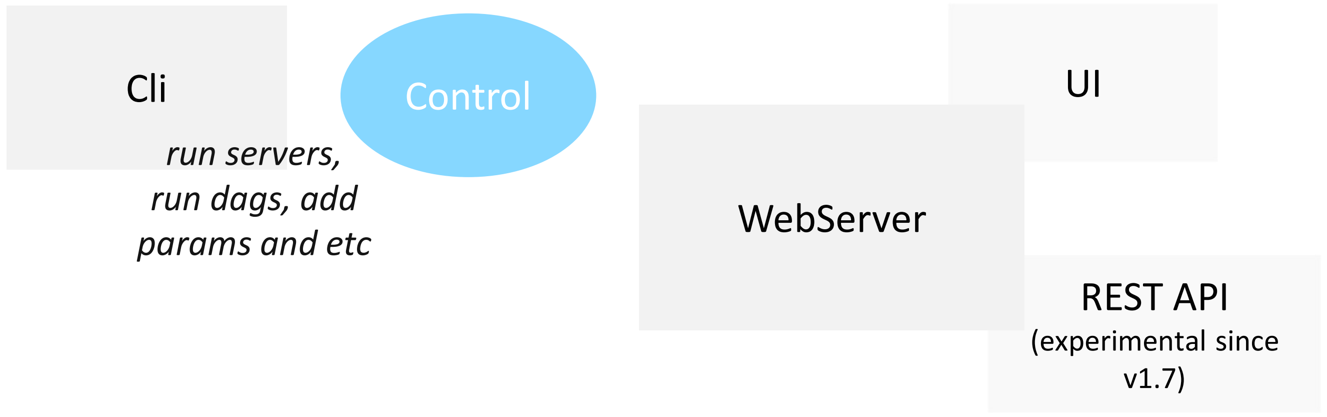
Прежде чем мы создадим первую DAG — запустим Apache Airflow Server

вся информация относительно Apache Airflow актуально для версии 1.10.12 и может отличаться от версии 2.0+

High-level overview of Apache Airflow components



High-level overview of Apache Airflow components



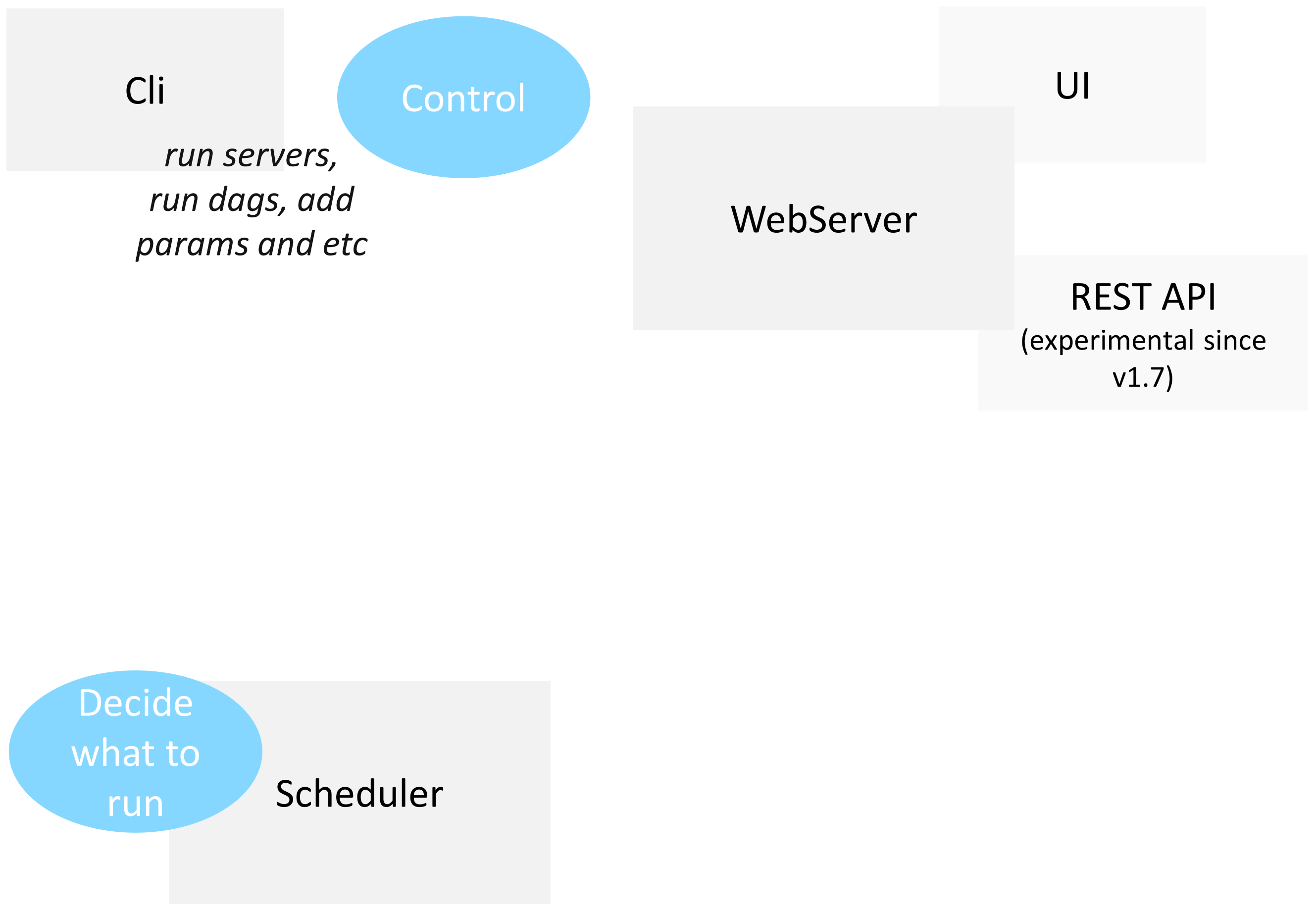
CLI

https://airflow.apache.org/docs/apache-airflow/stable/cli-ref

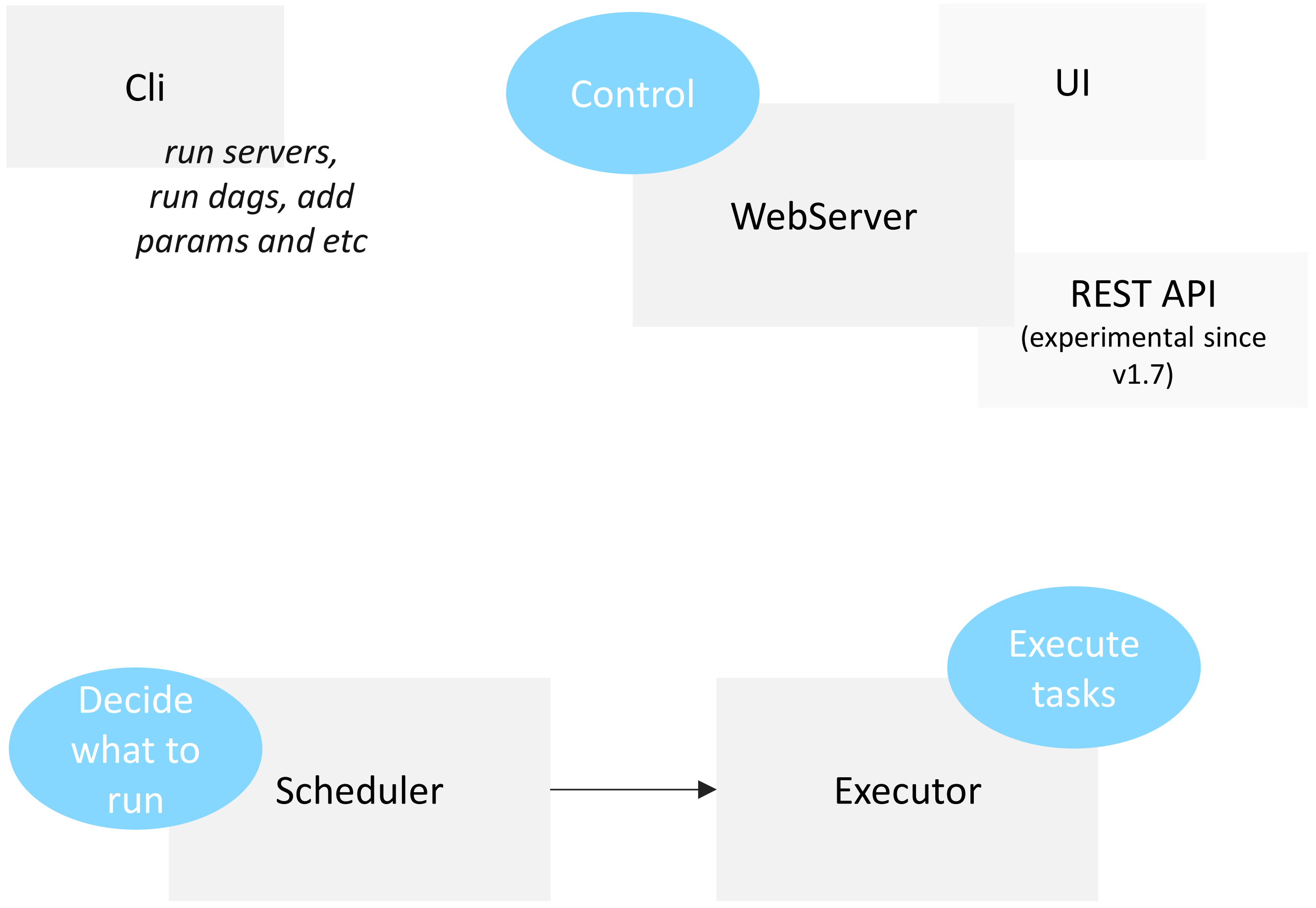
Server commands:

```
airflow initdb
airflow webserver
airflow scheduler and etc.
```

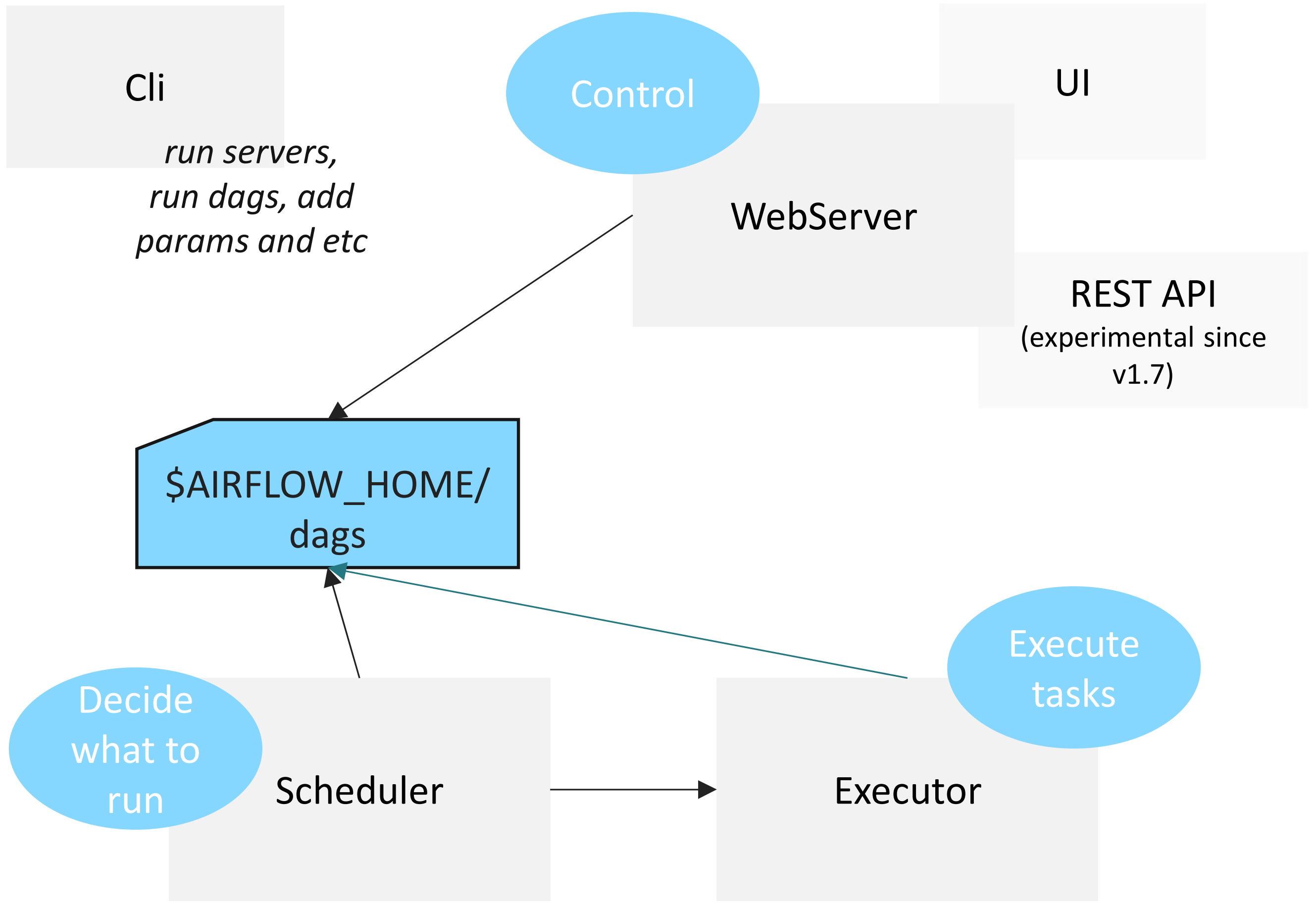
High-level overview of Apache Airflow components


Cli
Control
run servers, run dags, add params and etc
UI
WebServer
REST API (experimental since v1.7)
Decide what to run
Scheduler

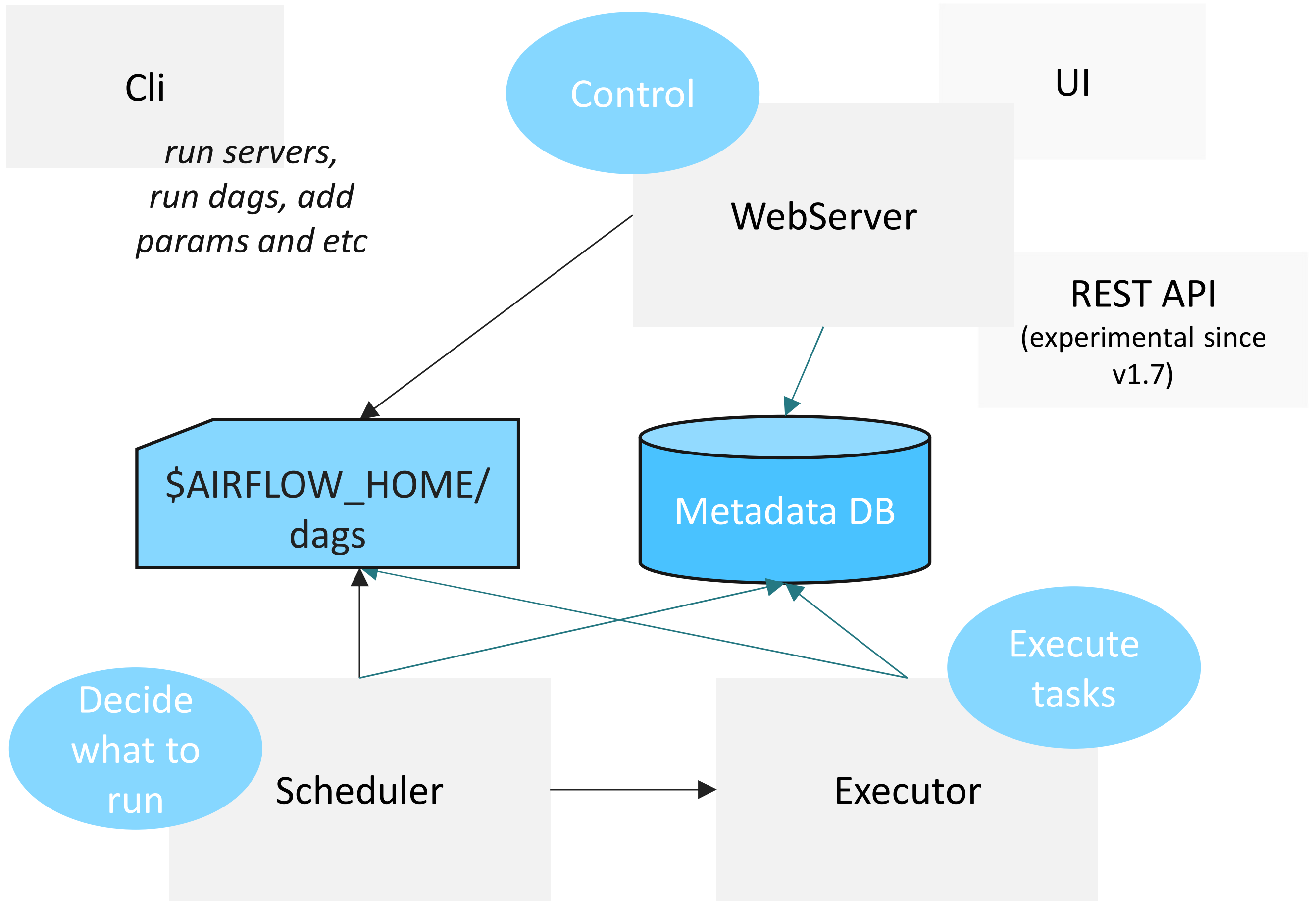
High-level overview of Apache Airflow components


Cli
run servers, run dags, add params and etc
Control
UI
WebServer
REST API (experimental since v1.7)
Decide what to run
Scheduler
Executor
Execute tasks

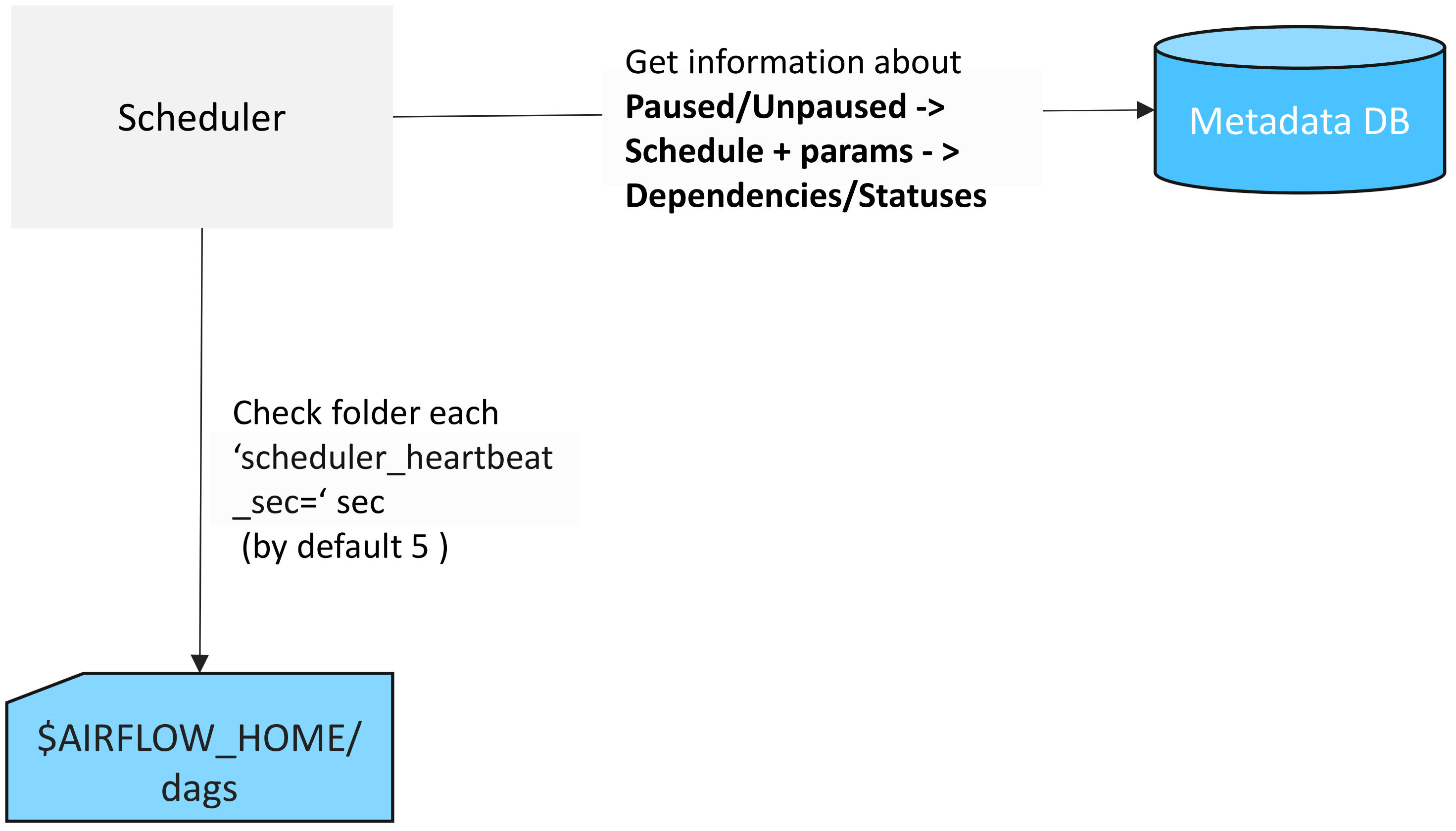
High-level overview of Apache Airflow components



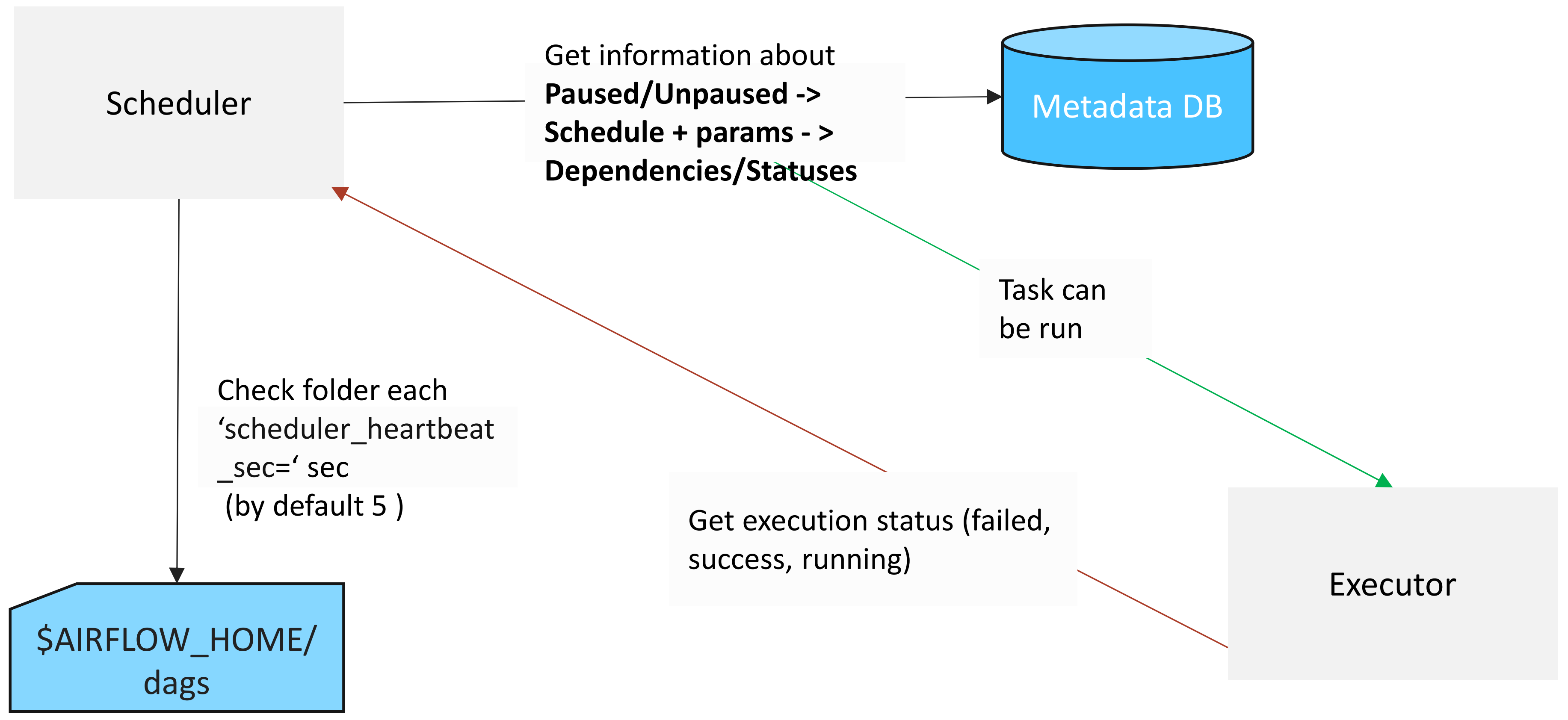
High-level overview of Apache Airflow components



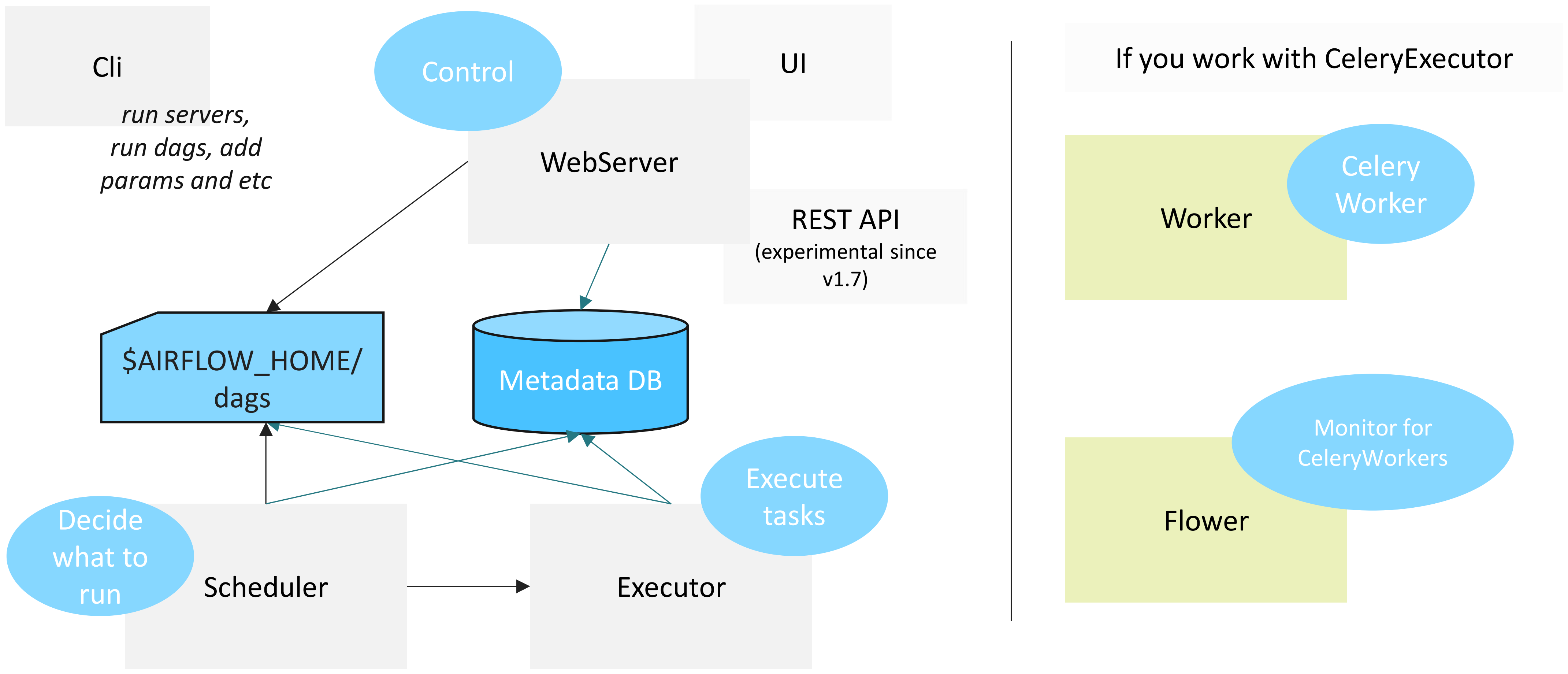
Process of DAG execution



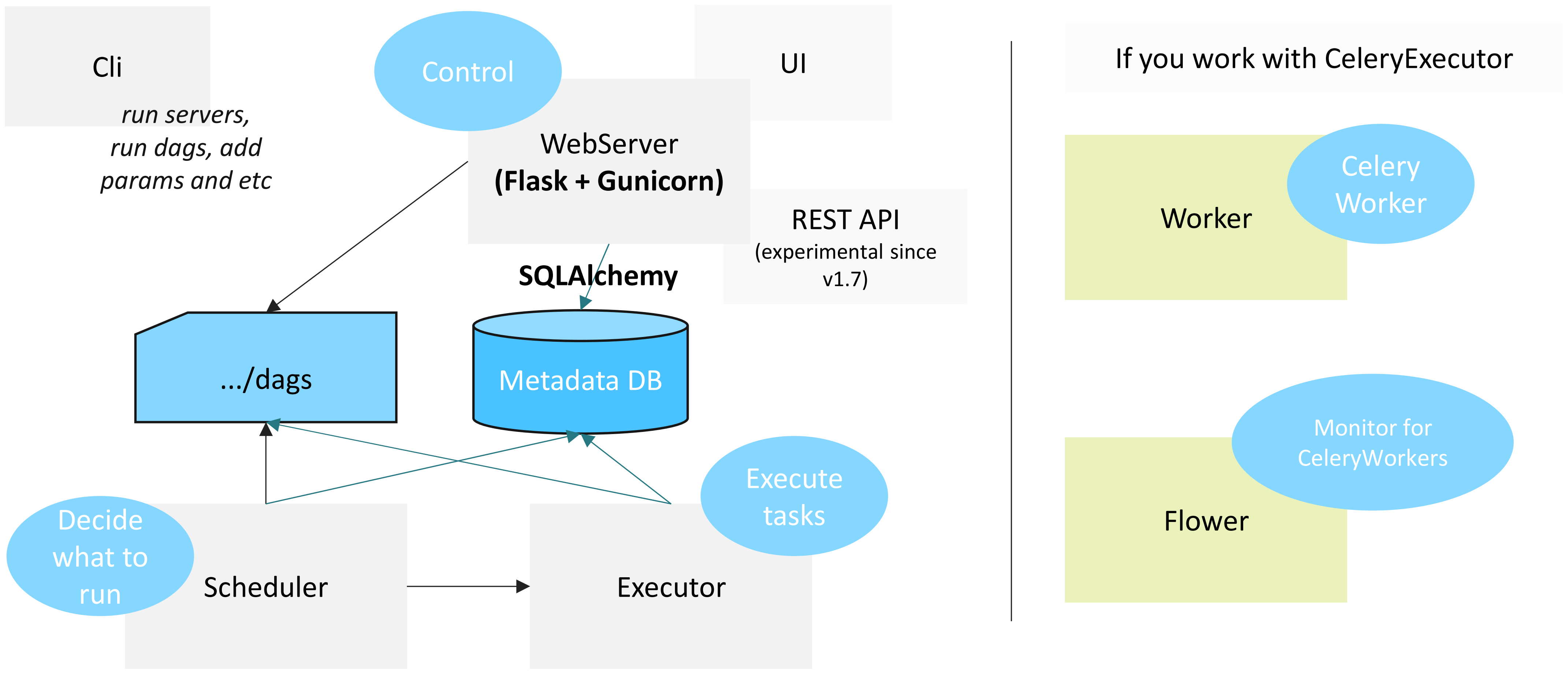
Process of DAG execution



High-level overview of Apache Airflow components


Cli
run servers, run dags, add params and etc
Control
WebServer
UI
REST API (experimental since v1.7)
$AIRFLOW_HOME/dags
Metadata DB
Decide what to run
Scheduler
Executor
Execute tasks
If you work with CeleryExecutor
Worker
Celery Worker
Flower
Monitor for CeleryWorkers

High-level overview of Apache Airflow components



Quick Start

https://airflow.apache.org/docs/stable/start.html#quick-start

```
# install from pypi using pip

pip install apache-airflow

# initialize the database (create all needed tables)


airflow initdb

# start the web server, default port is 8080
airflow webserver -p 8080
# start the scheduler
airflow scheduler
```

```
# airflow needs a home, ~/airflow is
the default, # but you can lay
foundation somewhere else if you
prefer # (optional)

export AIRFLOW_HOME=~/airflow
```

Errors

In **November 2020** after install apache-airflow==1.10.12 if you will try to run ‘**airflow initdb**’ you will get an error:

```
   from attr import fields, resolve_types
ImportError: cannot import name 'resolve_types' from 'attr’
```

To solve it you need install cattrs==1.1.0:

```
$ pip install cattrs==1.1.0
```

Remove DAG examples

1. Set in config option “load_examples = False“ before **airflow initdb**

If you already did ‘airflow initdb’ and want to remove example DAGs

1. Set in config option “load_examples = False“
2. Run **“airflow resetdb”**

Airflow by default

executor = SequentialExecutor

sql_alchemy_conn = sqlite:////Users/iuliia_volkova2/airflow/airflow.db – **only 1 connection**

Extra packages in Installation:

https://airflow.apache.org/docs/apache-airflow/stable/installation.html#extra-packages

Let’s define our first DAG

Create a DAGFile in $AIRFLOW_HOME/dags directory

DAGFile – file with .py that contains words ‘airflow’ and ‘DAG’

If you don’t want Apache Airflow to parse your files:
add it to .airflowignore in DAGs folder

Let’s define our first DAG

```
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import
DummyOperator


with DAG(
  dag_id="consume_new_data_from_pos",
  start_date=datetime(2020, 12, 1),
  schedule_interval=None
  ) as dag:
```

dag_id – unique dag_id (dag name)
start_date – date from that we start process the date
schedule_interval – schedule how we plan to run DAG (daily, hourly and etc)

Let’s define our first DAG

```
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import
DummyOperator


with DAG(
   dag_id="consume_new_data_from_pos",
   start_date=datetime(2020, 12, 1),
   schedule_interval=None
   ) as dag:
```

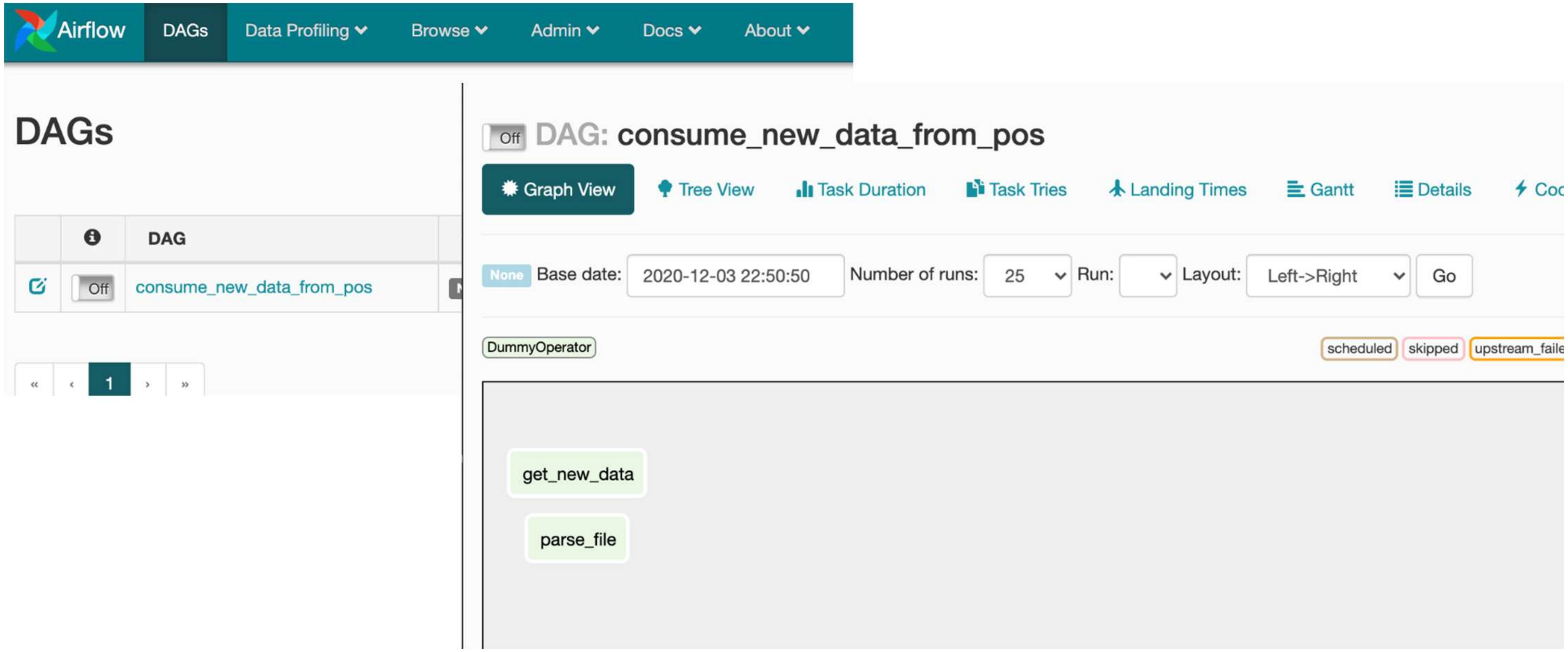
dag_id – unique dag_id (dag name)
start_date – date from that we start process the date
schedule_interval – schedule how we plan to run DAG (daily, hourly and etc)

Add tasks to the DAG

```
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator


with DAG(
  dag_id="consume_new_data_from_pos",
  start_date=datetime(2020, 12, 1),
  schedule_interval=None
  ) as dag:

  get_new_data = DummyOperator(task_id="get_new_data")

  parse_file = DummyOperator(task_id="parse_file")
```

task_id – unique task_id, mandatory to all Operators
DummyOperator – operator that does nothing (useful to prototype pipeline)

Let’s check the UI



Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About

DAGs

		DAG
	Off	consume_new_data_from_pos

« ‹ 1 › »

Off DAG: consume_new_data_from_pos

Graph View | Tree View | Task Duration | Task Tries | Landing Times | Gantt | Details | Coc

None Base date: 2020-12-03 22:50:50 Number of runs: 25 Run: Layout: Left->Right Go

DummyOperator

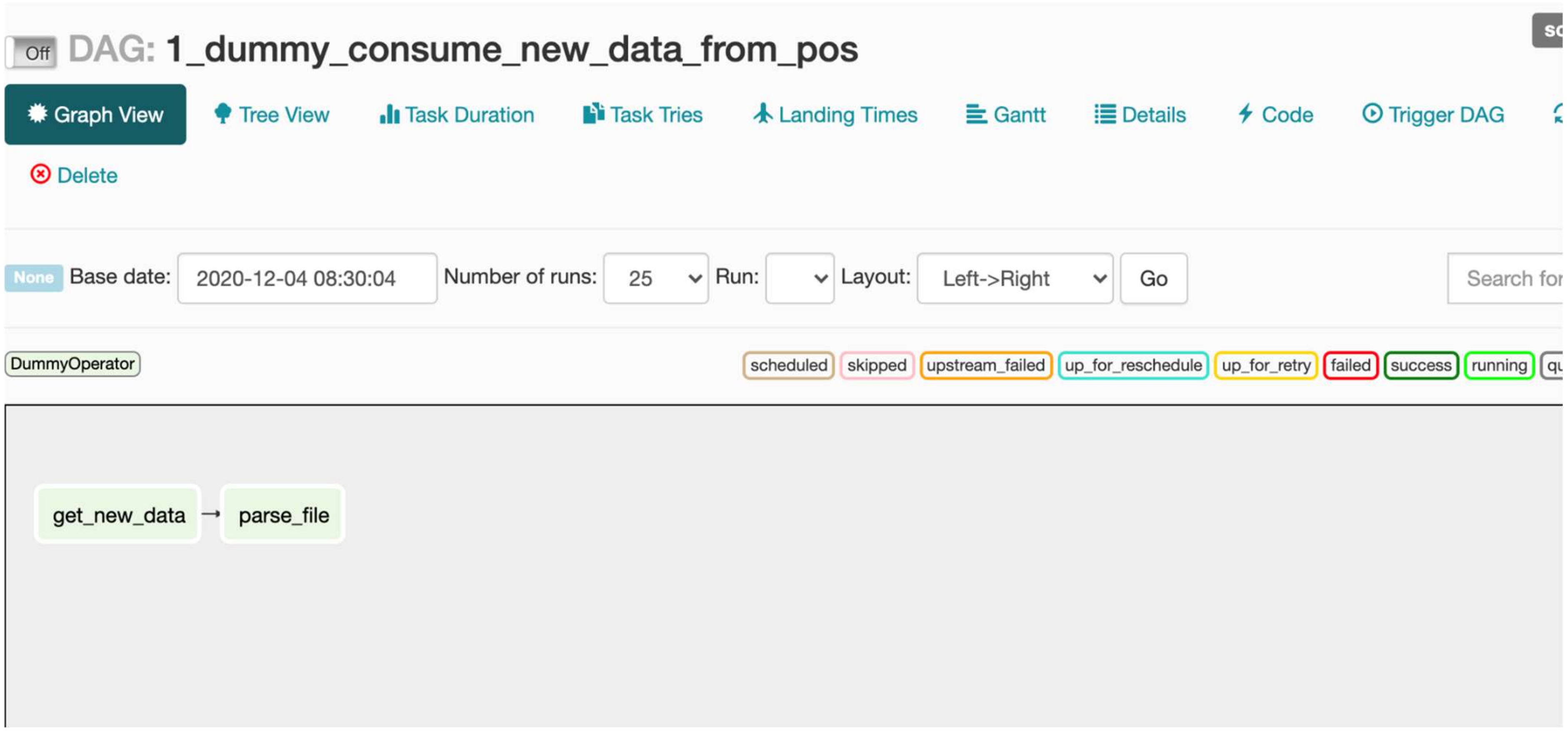
scheduled skipped upstream_faile

Define a sequence of tasks

```python
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator


with DAG(
  dag_id="consume_new_data_from_pos",
  start_date=datetime(2020, 12, 1),
  schedule_interval=None
  ) as dag:

  get_new_data = DummyOperator(task_id="get_new_data")

  parse_file = DummyOperator(task_id="parse_file")

  get_new_data >> parse_file
```

set_downstream	>>
set_upstream	<<

Define a sequence of tasks



Define a sequence of tasks

```
[task1, task2, task3]   >>  task4 - allowed
```

```
task4  >> [task1, task2, task3] – allowed
task5  >> [task1, task2, task3]
```

```
[task1, task2, task3] >> [task4, task5] – not allowed
```

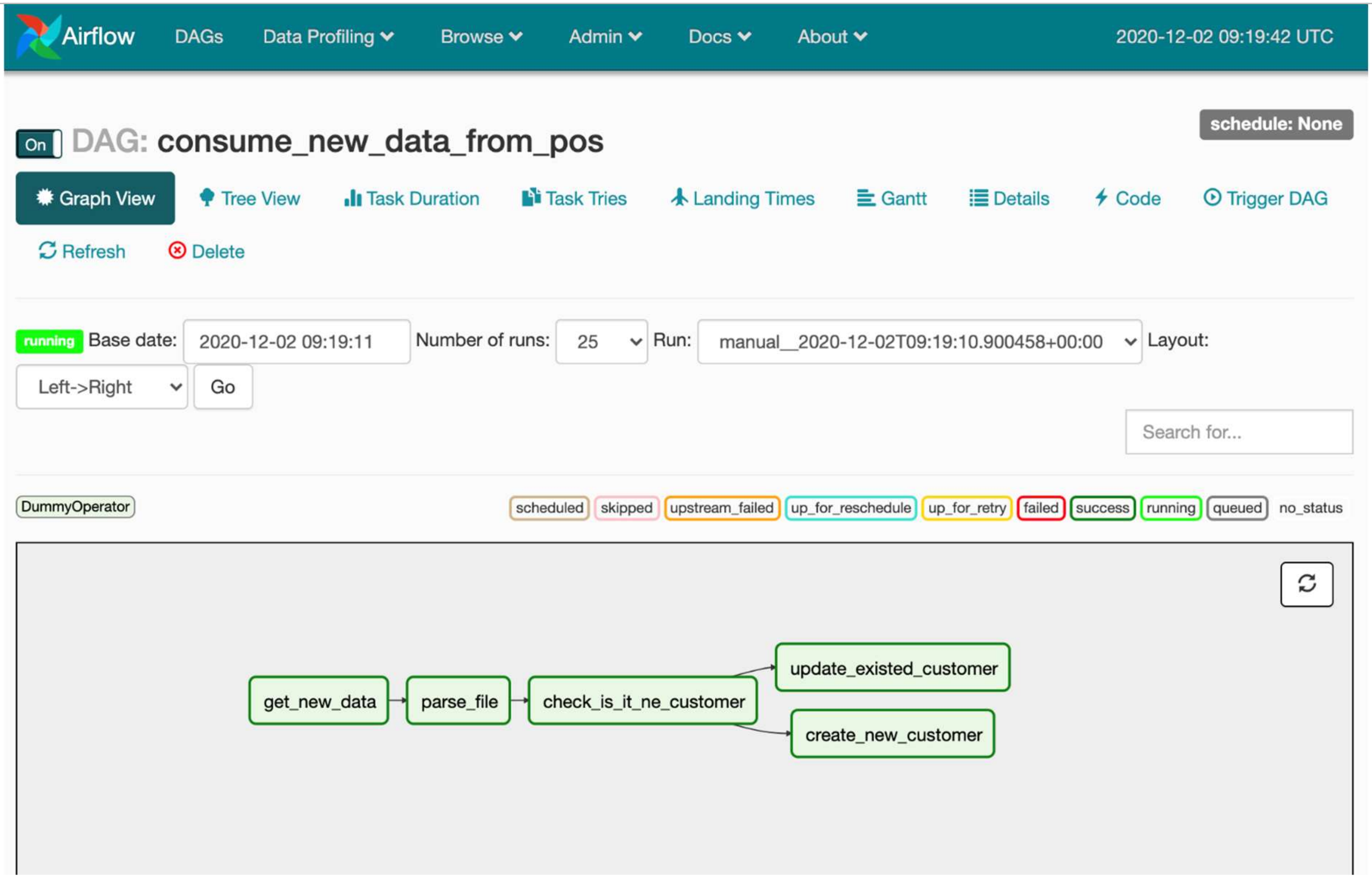
```
[task4, task5] >> [task1, task2, task3] – not allowed
```

```
unsupported operand type(s) for >>: 'list' and 'list'
```

Let’s define the full DAG

```
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator


with DAG(
  dag_id="consume_new_data_from_pos",
  start_date=datetime(2020, 12, 1),
  schedule_interval=None
  ) as dag:
  get_new_data = DummyOperator(task_id="get_new_data")

  parse_file = DummyOperator(task_id="parse_file")

  check_is_it_ne_customer =
DummyOperator(task_id="check_is_it_ne_customer")

  create_new_customer = DummyOperator(task_id="create_new_customer")

  update_existed_customer =
DummyOperator(task_id="update_existed_customer")

  get_new_data >> parse_file >> check_is_it_ne_customer >>
[create_new_customer, update_existed_customer]
```

Apache Airflow UI


Airflow DAGs Data Profiling Browse Admin Docs About 2020-12-02 09:19:42 UTC

On DAG: consume_new_data_from_pos

schedule: None

Graph View · Tree View · Task Duration · Task Tries · Landing Times · Gantt · Details · Code · Trigger DAG

Refresh · Delete

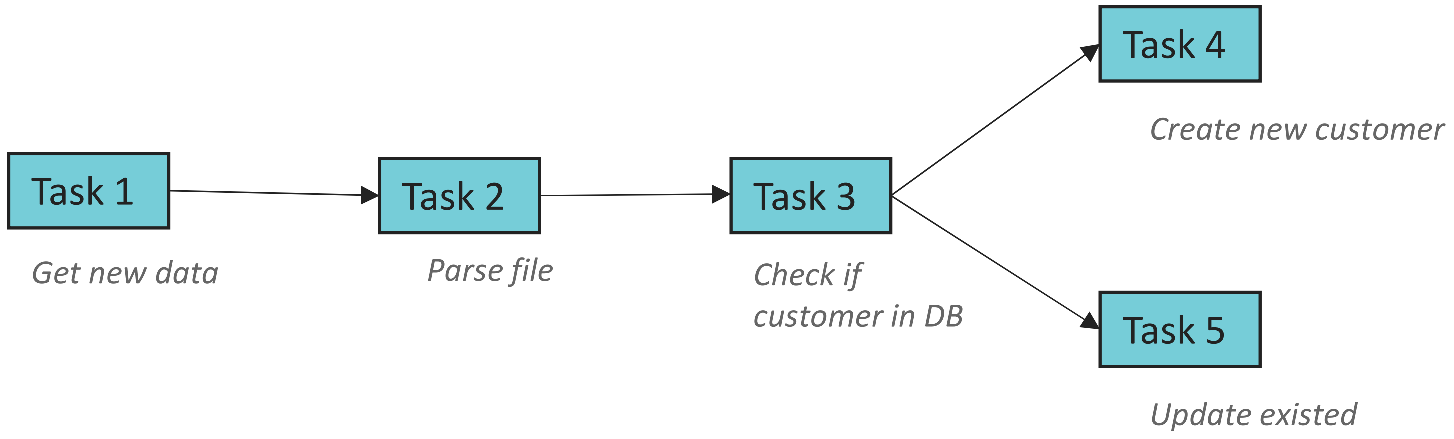
running Base date: 2020-12-02 09:19:11 Number of runs: 25 Run: manual__2020-12-02T09:19:10.900458+00:00 Layout: Left->Right Go

Search for...

DummyOperator

scheduled skipped upstream_failed up_for_reschedule up_for_retry failed success running queued no_status

DAG – Directed Acyclic Graph



What can be a Task?

Operators

Просто СДЕЛАЙТЕ прямо сейчас
- отчет о завершении

Sensors

Poke (ожидание) условие
пока не выполнено

What can be a Task?

Operators

Просто СДЕЛАЙТЕ прямо сейчас - отчет о завершении

Examples:

- FileToGoogleCloudStorageOperator
- MySqlOperator
- AWSAthenaOperator

…

Sensors

Poke (ожидание) условие пока не выполнено

Examples:

- HdfsSensor
- HttpSensor
- SqlSensor

…

Moment of Task Completion

Run **Spark Job**

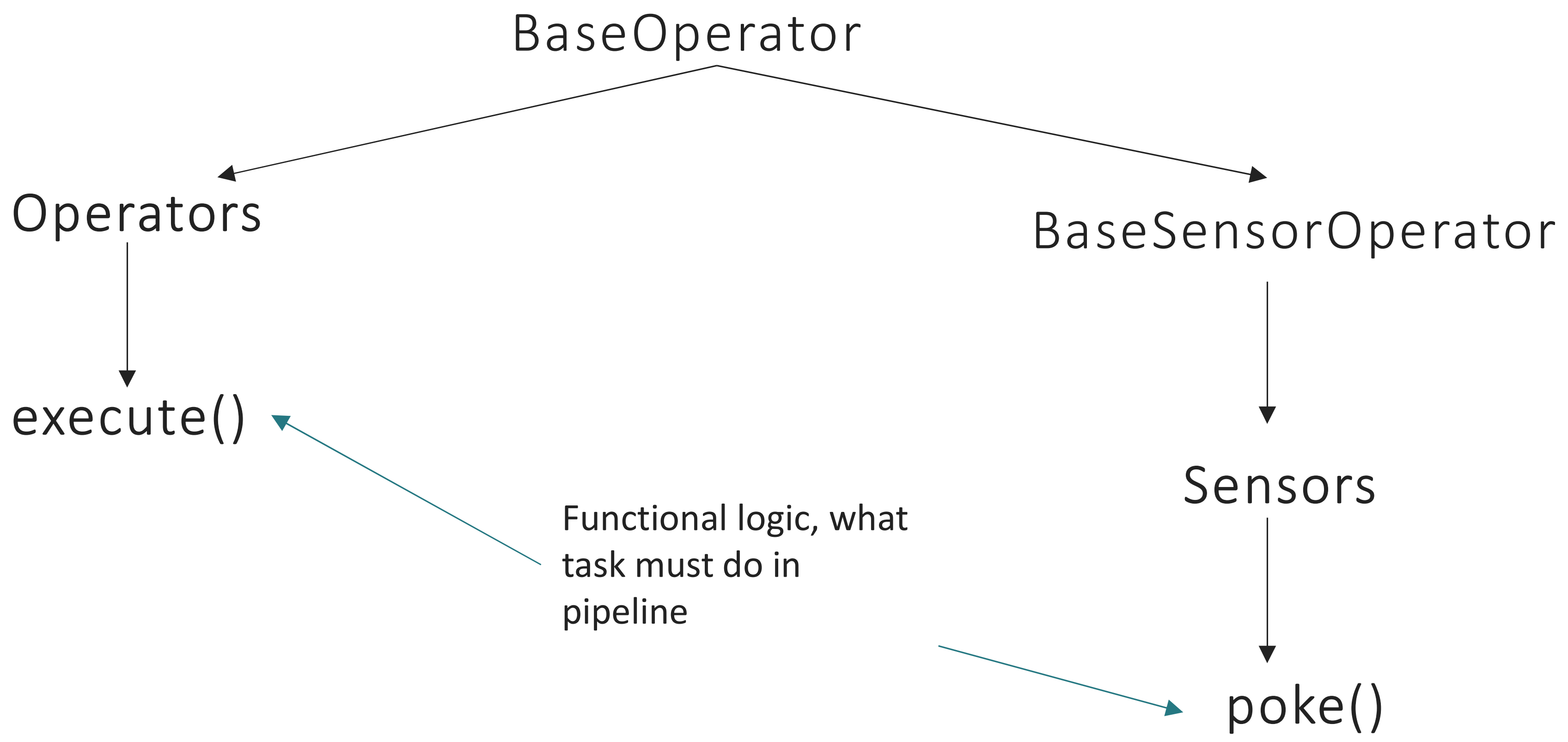
- run as background process
- By ssh in another server
- By REST

(Task **is 'success'** after
send a command to run job)

– Run as a java command
in current server, wait until it finish

(Task in **'running'**
status until complete)

What can be a Task?



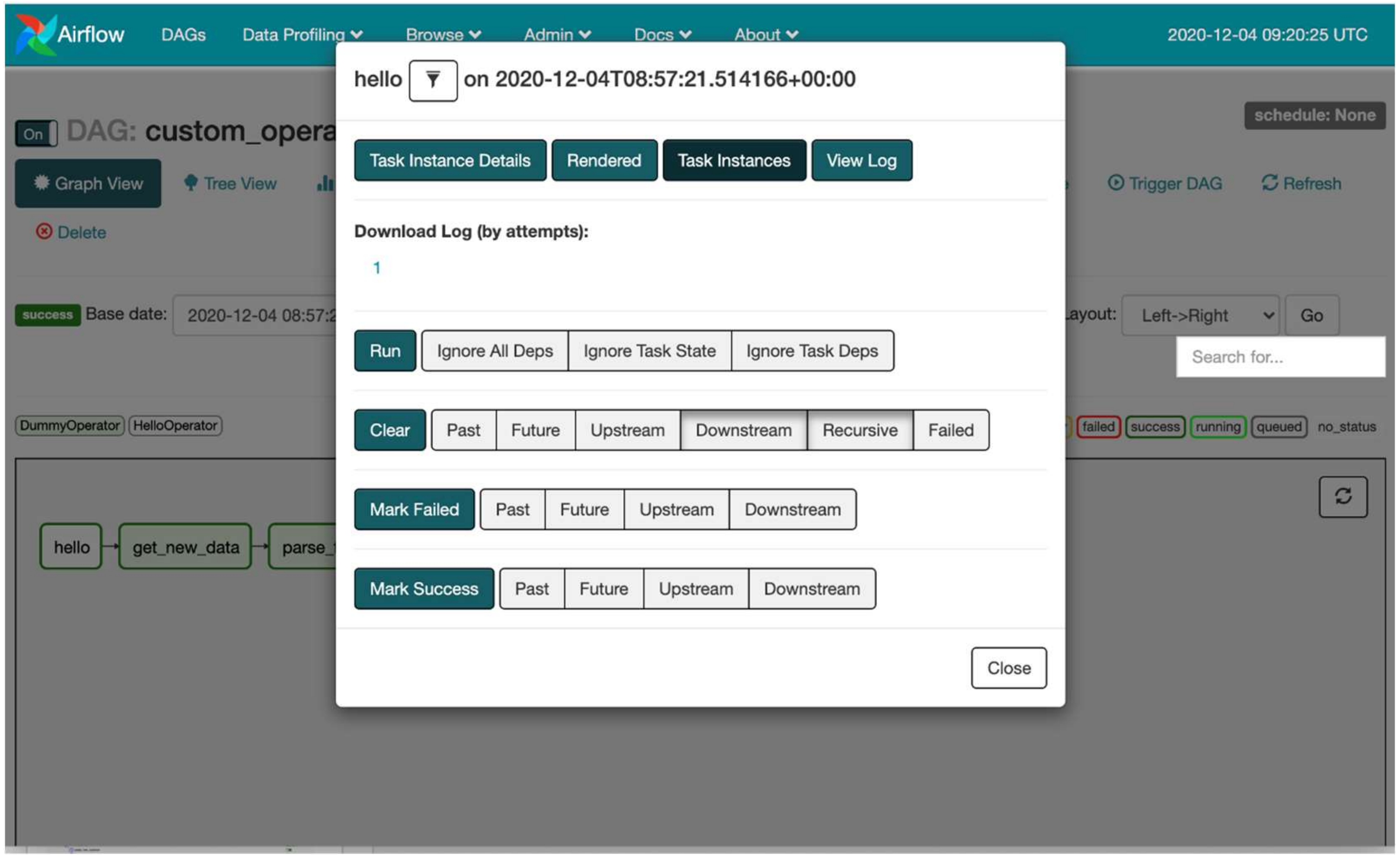
Let’s define our primitive Operator

```
from typing import Union, Iterable, Dict

from airflow.models import BaseOperator, SkipMixin


class HelloOperator(BaseOperator, SkipMixin):

  def execute(self, context):
      self.logger.info("Hello, World!")
```

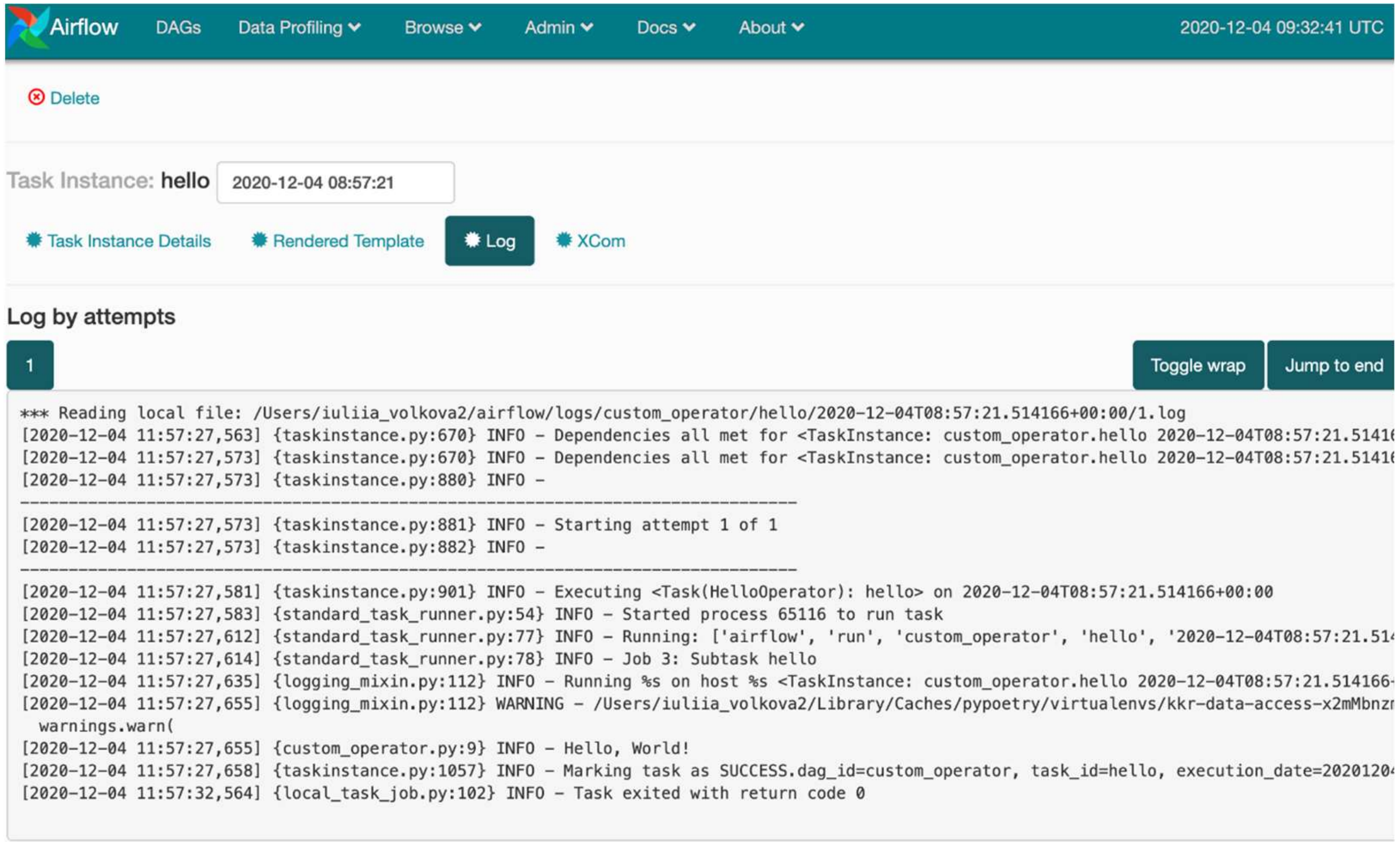
And put module with it to $AIRFLOW_HOME/dags directory

Airflow add $AIRFLOW_HOME/dags to PYTHONPATH
 so everything inside it you can use with import

Check the UI that all works good

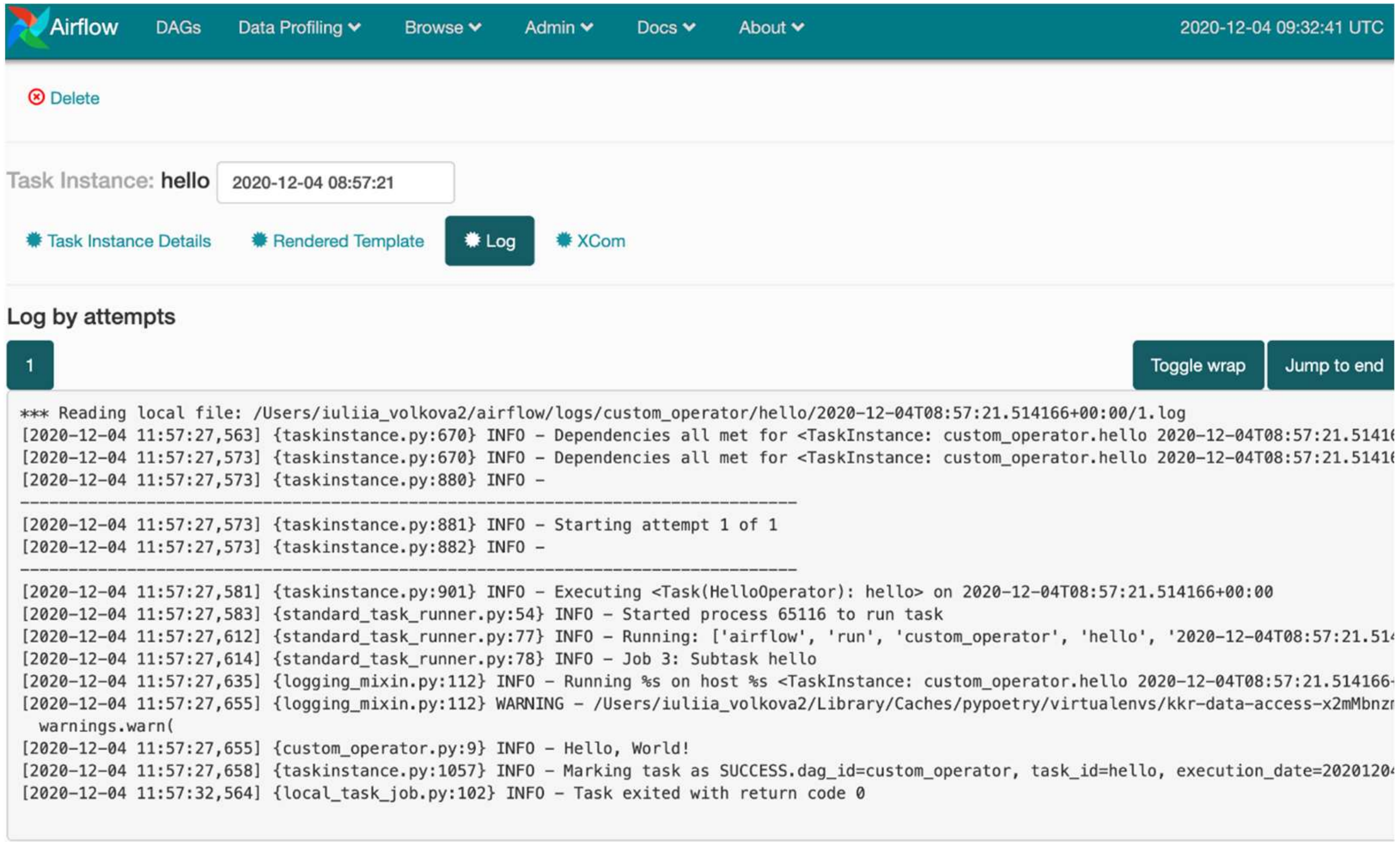


Check the UI that all works good



```
*** Reading local file: /Users/iuliia_volkova2/airflow/logs/custom_operator/hello/2020-12-04T08:57:21.514166+00:00/1.log
[2020-12-04 11:57:27,563] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.51416
[2020-12-04 11:57:27,573] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.51416
[2020-12-04 11:57:27,573] {taskinstance.py:880} INFO -
--------------------------------------------------------------------------------
[2020-12-04 11:57:27,573] {taskinstance.py:881} INFO - Starting attempt 1 of 1
[2020-12-04 11:57:27,573] {taskinstance.py:882} INFO -
--------------------------------------------------------------------------------
[2020-12-04 11:57:27,581] {taskinstance.py:901} INFO - Executing <Task(HelloOperator): hello> on 2020-12-04T08:57:21.514166+00:00
[2020-12-04 11:57:27,583] {standard_task_runner.py:54} INFO - Started process 65116 to run task
[2020-12-04 11:57:27,612] {standard_task_runner.py:77} INFO - Running: ['airflow', 'run', 'custom_operator', 'hello', '2020-12-04T08:57:21.51
[2020-12-04 11:57:27,614] {standard_task_runner.py:78} INFO - Job 3: Subtask hello
[2020-12-04 11:57:27,635] {logging_mixin.py:112} INFO - Running %s on host %s <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.514166
[2020-12-04 11:57:27,655] {logging_mixin.py:112} WARNING - /Users/iuliia_volkova2/Library/Caches/pypoetry/virtualenvs/kkr-data-access-x2mMbnz
  warnings.warn(
[2020-12-04 11:57:27,655] {custom_operator.py:9} INFO - Hello, World!
[2020-12-04 11:57:27,658] {taskinstance.py:1057} INFO - Marking task as SUCCESS.dag_id=custom_operator, task_id=hello, execution_date=2020120
[2020-12-04 11:57:32,564] {local_task_job.py:102} INFO - Task exited with return code 0
```

Check the UI that all works good



```
*** Reading local file: /Users/iuliia_volkova2/airflow/logs/custom_operator/hello/2020-12-04T08:57:21.514166+00:00/1.log
[2020-12-04 11:57:27,563] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.5141
[2020-12-04 11:57:27,573] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.5141
[2020-12-04 11:57:27,573] {taskinstance.py:880} INFO -
--------------------------------------------------------------------------------
[2020-12-04 11:57:27,573] {taskinstance.py:881} INFO - Starting attempt 1 of 1
[2020-12-04 11:57:27,573] {taskinstance.py:882} INFO -
--------------------------------------------------------------------------------
[2020-12-04 11:57:27,581] {taskinstance.py:901} INFO - Executing <Task(HelloOperator): hello> on 2020-12-04T08:57:21.514166+00:00
[2020-12-04 11:57:27,583] {standard_task_runner.py:54} INFO - Started process 65116 to run task
[2020-12-04 11:57:27,612] {standard_task_runner.py:77} INFO - Running: ['airflow', 'run', 'custom_operator', 'hello', '2020-12-04T08:57:21.51
[2020-12-04 11:57:27,614] {standard_task_runner.py:78} INFO - Job 3: Subtask hello
[2020-12-04 11:57:27,635] {logging_mixin.py:112} INFO - Running %s on host %s <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.514166
[2020-12-04 11:57:27,655] {logging_mixin.py:112} WARNING - /Users/iuliia_volkova2/Library/Caches/pypoetry/virtualenvs/kkr-data-access-x2mMbnz
  warnings.warn(
[2020-12-04 11:57:27,655] {custom_operator.py:9} INFO - Hello, World!
[2020-12-04 11:57:27,658] {taskinstance.py:1057} INFO - Marking task as SUCCESS.dag_id=custom_operator, task_id=hello, execution_date=2020120
[2020-12-04 11:57:32,564] {local_task_job.py:102} INFO - Task exited with return code 0
```