
Apache Airflow



Iulia Volkova, xnuinside@gmail.com

Agenda (Day 1)

1. What is Apache Airflow?
2. What is Workflow Manager (orchestrator) and what is ETL
3. Server Components & basic installation, cli
4. DAG, basic DAG params & DAGFile, Tasks
5. DAG Run & Task Instance
6. Operators, Sensors
7. Schedule interval & catch up & execution date
8. Junja2 Templating,
9. Task Statuses
10. Files to play with (homework)
11. Q&A session

Agenda (Day 2)

1. Macros, User Defined Macros, 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. Backfill
10. Customization: UI plugins
11. Airflow in clouds: Google Compose (Airflow in GCP), Astronomer.io
12. Q&A session

What is Apache Airflow?

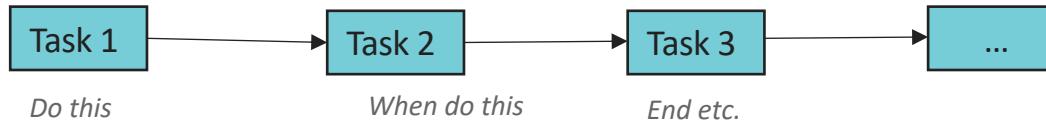


What is Apache Airflow?



It is an workflow manager!
(or orchestrator)

What is workflow or pipeline or DAG (Direct Acyclic Graph)



Alternatives (Orchestrators)



For DS & ML



Similar things for Ops/DevOps/non data workflows

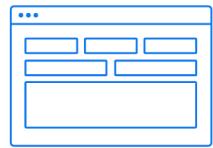


and etc.





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

Airflow DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾ 2018-09-22 10:48:00 UTC ⏪

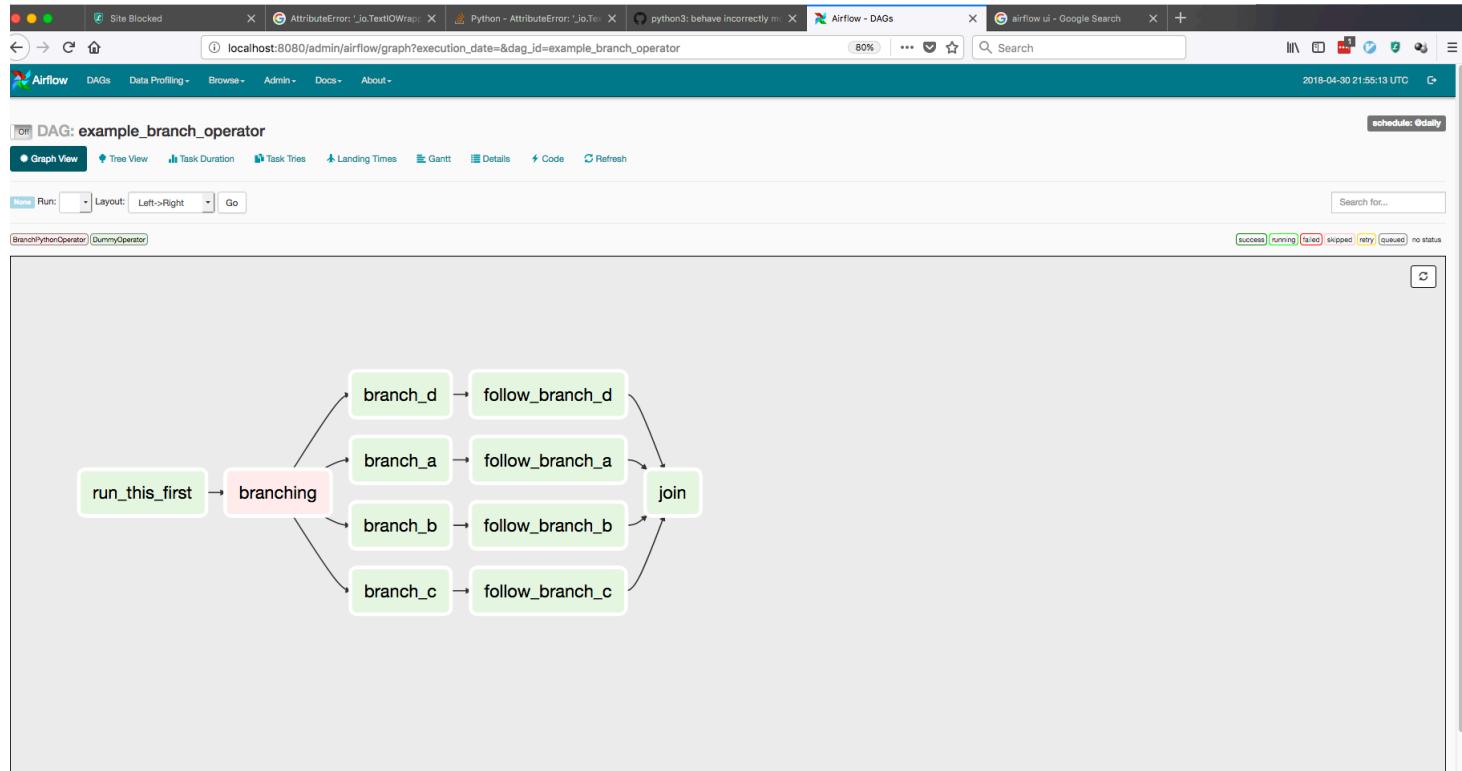
DAGs

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
		example_bash_operator	0 * * * *	airflow	(10)		(3)	
		example_branch_dop_operator_v3	* /1 * * * *	airflow	(10)		(3)	
		example_branch_operator	@daily	airflow	(10)		(3)	
		example_http_operator	1 day, 0:00:00	airflow	(10)		(3)	
		example_kubernetes_executor	None	airflow	(10)		(3)	
		example_passing_params_via_test_command	* /1 * * * *	airflow	(10)		(3)	
		example_python_operator	None	airflow	(10)		(3)	
		example_short_circuit_operator	1 day, 0:00:00	airflow	(10)		(3)	
		example_skip_dag	1 day, 0:00:00	airflow	(10)		(3)	



UI Screens – DAGs View





UI Screens – DAGs Runs, Tree View

On DAG: example_branch_dop_operator_v3

schedule: */1 * * * *

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Refresh Delete

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

BranchPythonOperator DummyOperator

success running failed skipped retry queued no status

[DAG] oper_1 condition oper_2 condition

05:45 06 PM

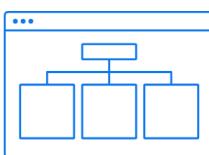


Main Features



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.



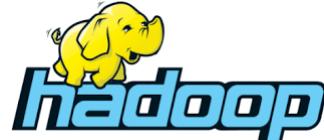
Robust Integrations

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*)



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

```
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
```

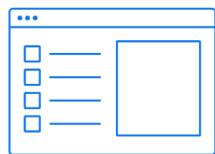


Main Features



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.

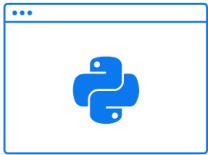


Easy to Use

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>

The screenshot shows the GitHub repository page for 'apache / airflow'. On the left, there's a 'Contributors' section with a count of 1,395, featuring small profile pictures of contributors. Below it are icons for 'XD' and a red circular logo with a white arrow. The main area displays a list of recent commits from 'jtimmins' and others, with details like commit message, author, time ago, and a link to the pull request. The right sidebar includes sections for 'About', 'Readme', 'Apache-2.0 License', and 'Releases'.

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

Official community Slack:

<https://apache-airflow-slack.herokuapp.com/>

List of committers (maintainers):

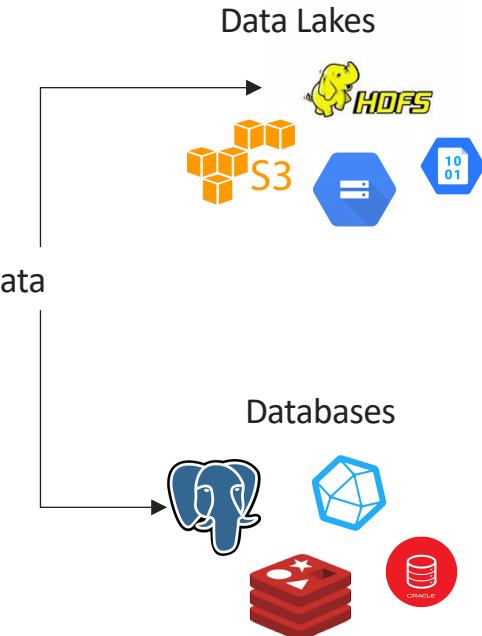
<https://people.apache.org/committers-by-project.html#airflow> (about 40 people)

What is a workflow manager?
(or orchestrator)

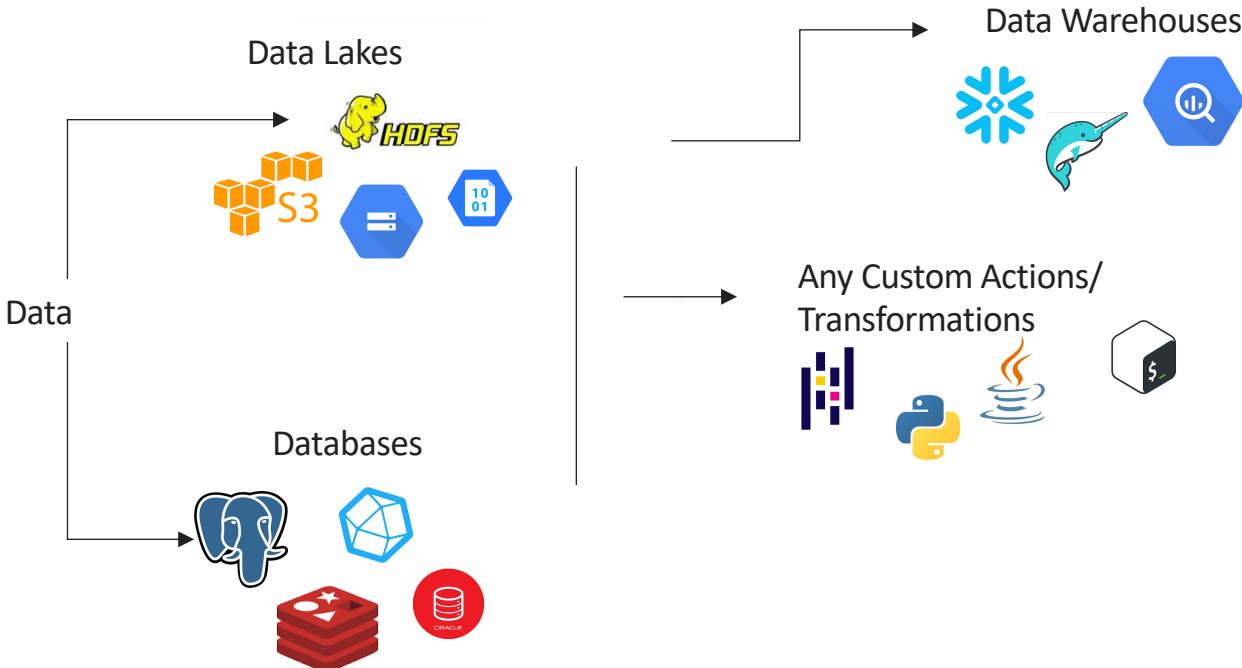
Data Flow



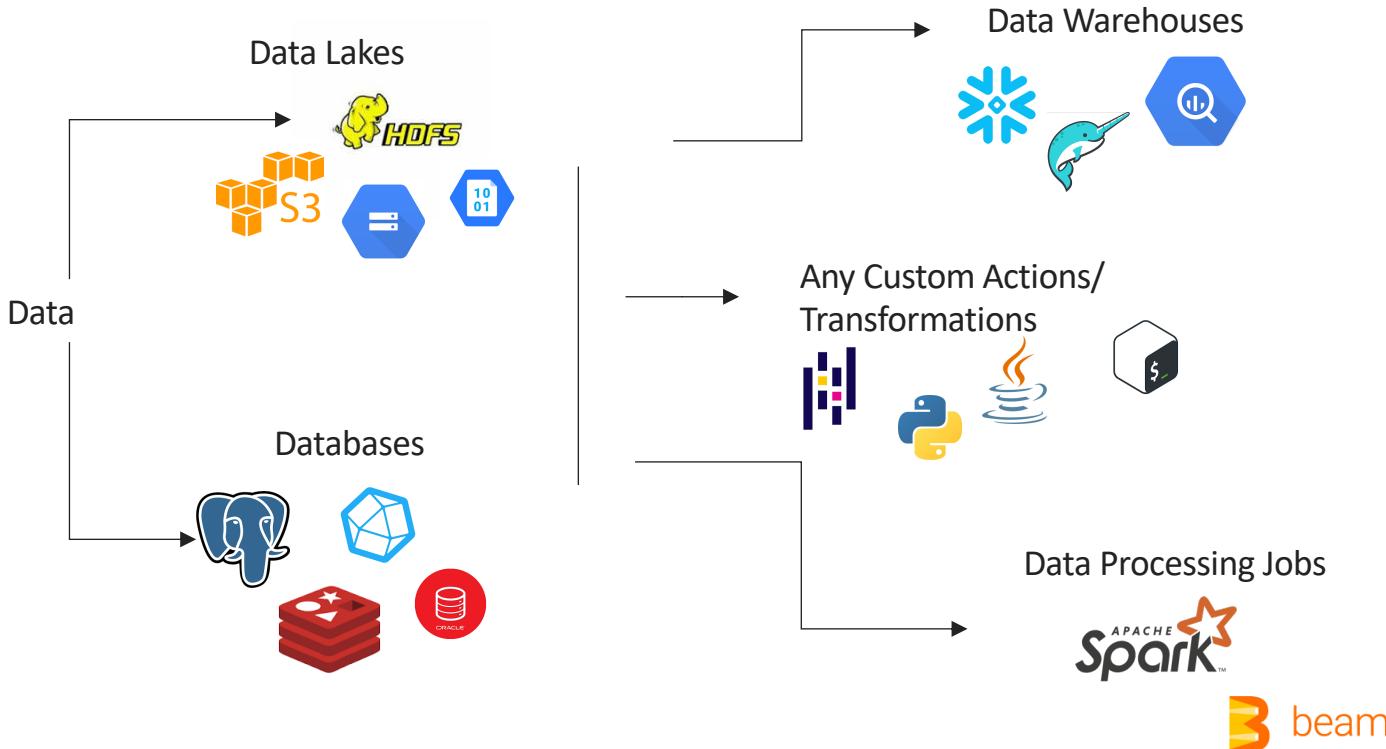
Data Flow



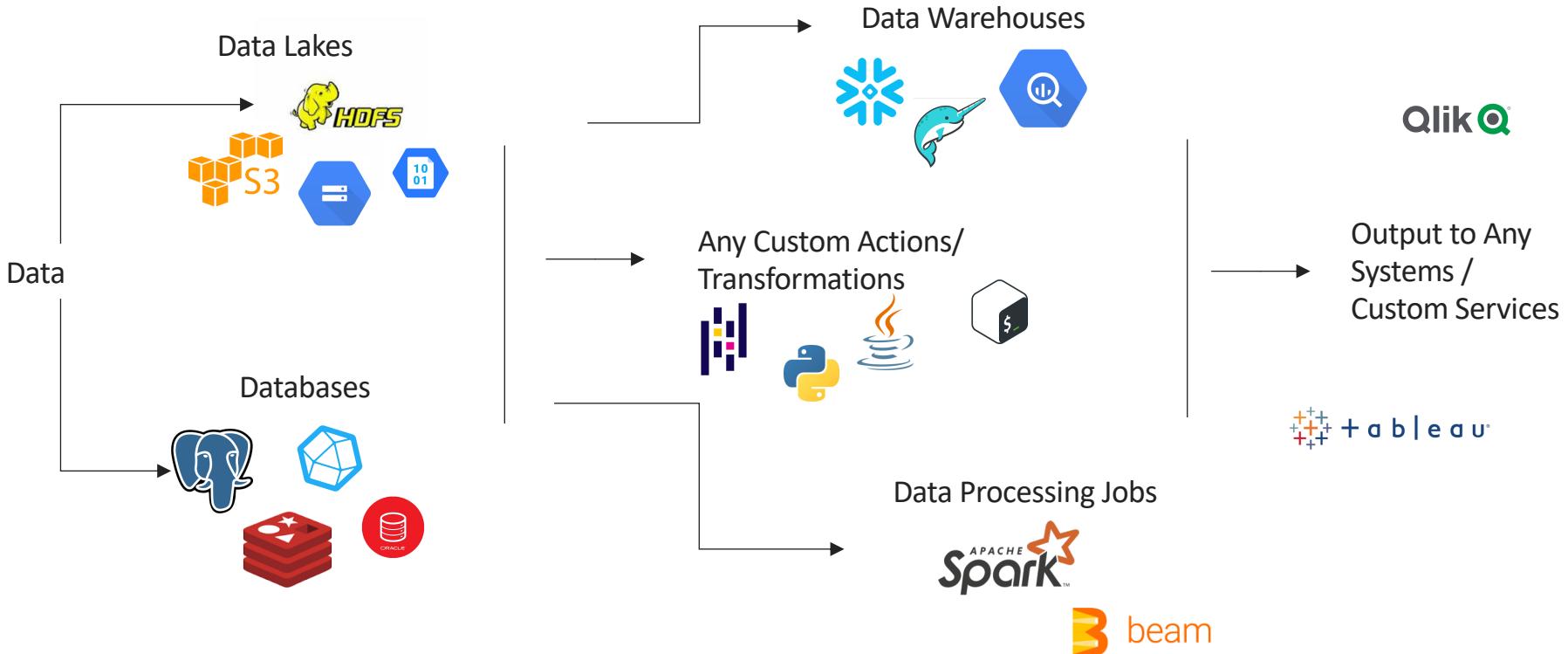
Data Flow



Data Flow



Data Flow



Orchestrator or Workflow manager

Allows you to create Data Pipelines
& describe all steps of your Data Flow:

from **where** to where, **what**, **when** and **how**

- multiple task in **any** sequence (not only classical ETL)

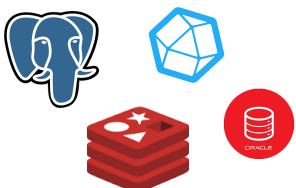
Extract, transform, load

Extract

Data Lakes



Databases



Transform

Any Custom Actions/
Transformations



Data Processing Jobs



Load

Data Warehouses



Output to Any
Systems /
Custom Services



Workflow Capabilities that we needs

1. **Monitoring Dashboard** (what's going on with our pipeline?)

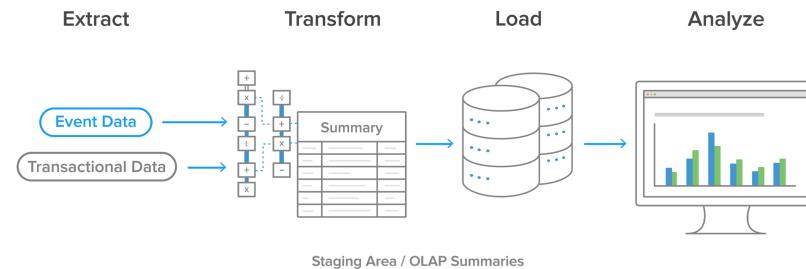
2. **Alerts** (if something wrong

- I must know about it quick)

3. **SLAs** (if we don't have data for the day

- do we have the problem?)

4. **Way to make customization**



And etc.

Pipeline Example in Words

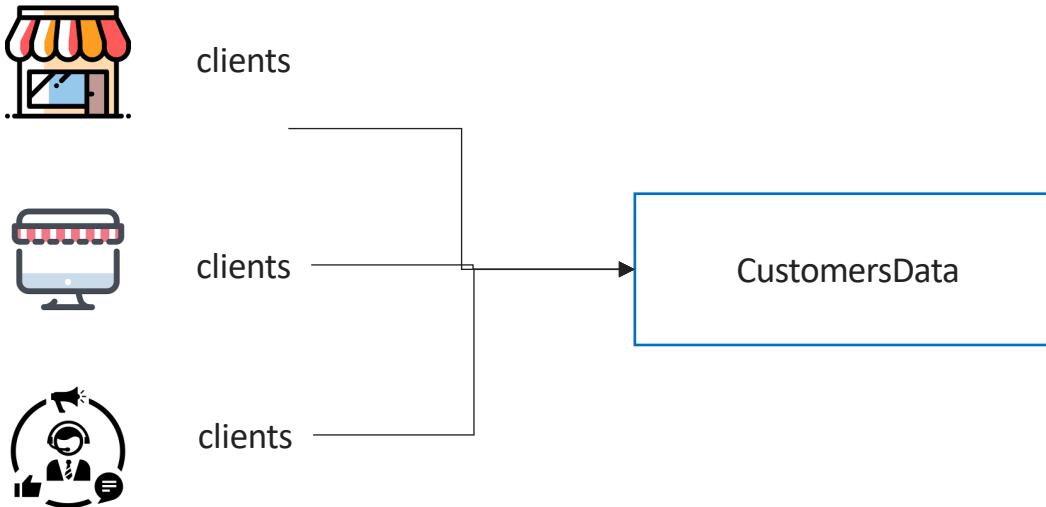
Let's imagine. We work in Data Engineering Team in The Stationery Shop (pens, papers and etc). We have about 1500 offline shops, online shop and direct sales.

We work on the Data Pipeline that assume information about our clients from different sources.

Pipeline Example in Words

Let's imagine. We work in Data Engineering Team in The Stationery Shop (pens, papers and etc). We have about 1500 offline shops, online shop and direct sales.

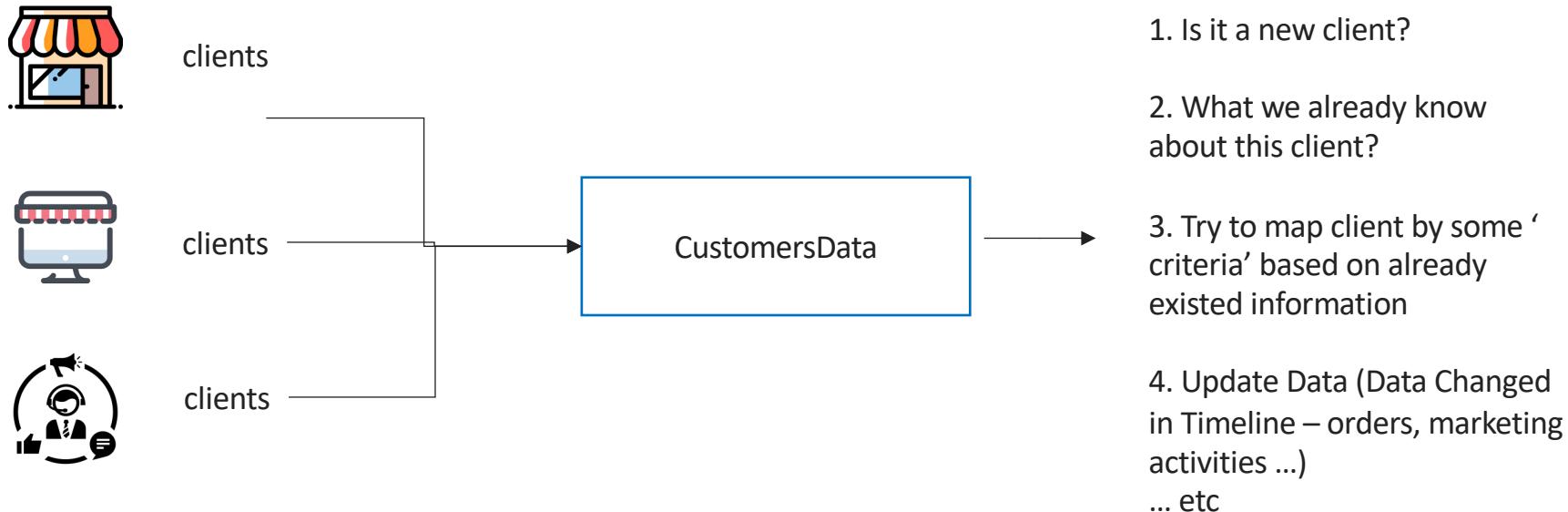
We work on the Data Pipeline that assume information about our clients from different sources.



Pipeline Example in Words

Let's imagine. We work in Data Engineering Team in The Stationery Shop (pens, papers and etc). We have about 1500 offline shops, online shop and direct sales.

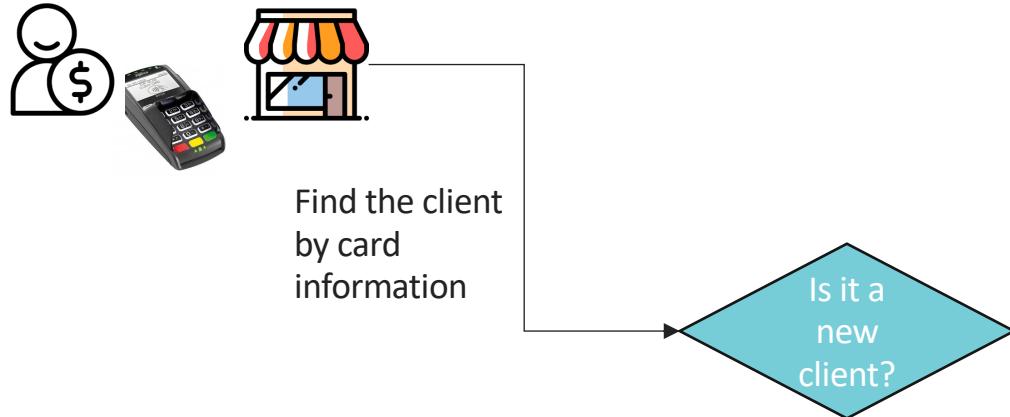
We work on the Data Pipeline that assume information about our clients from different sources.



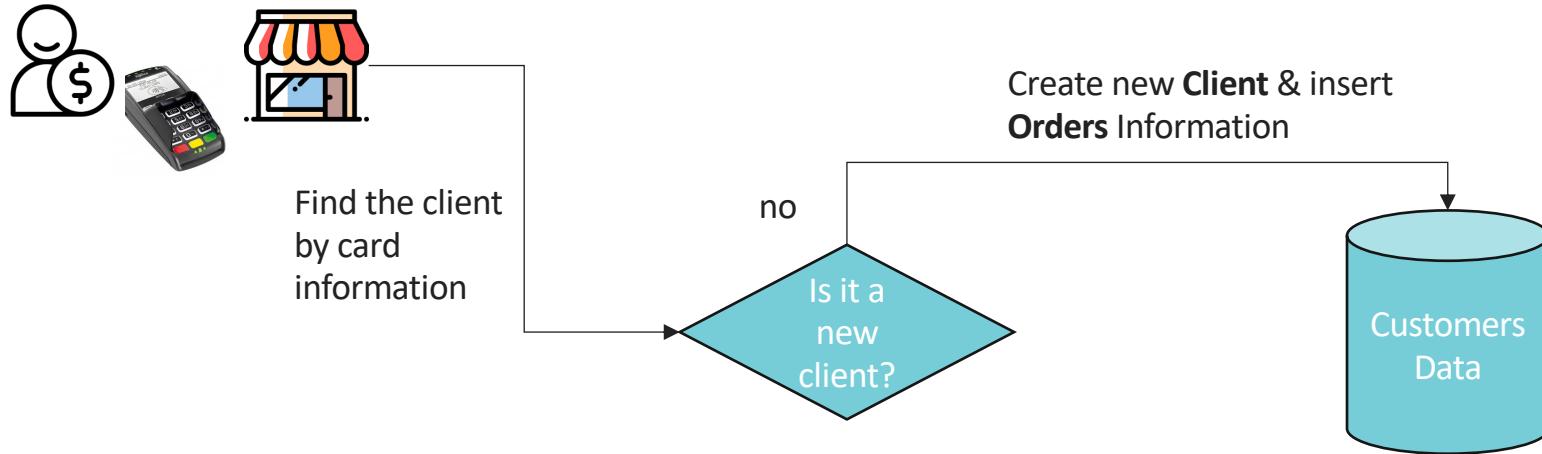
Pipeline – find the client by credit card number



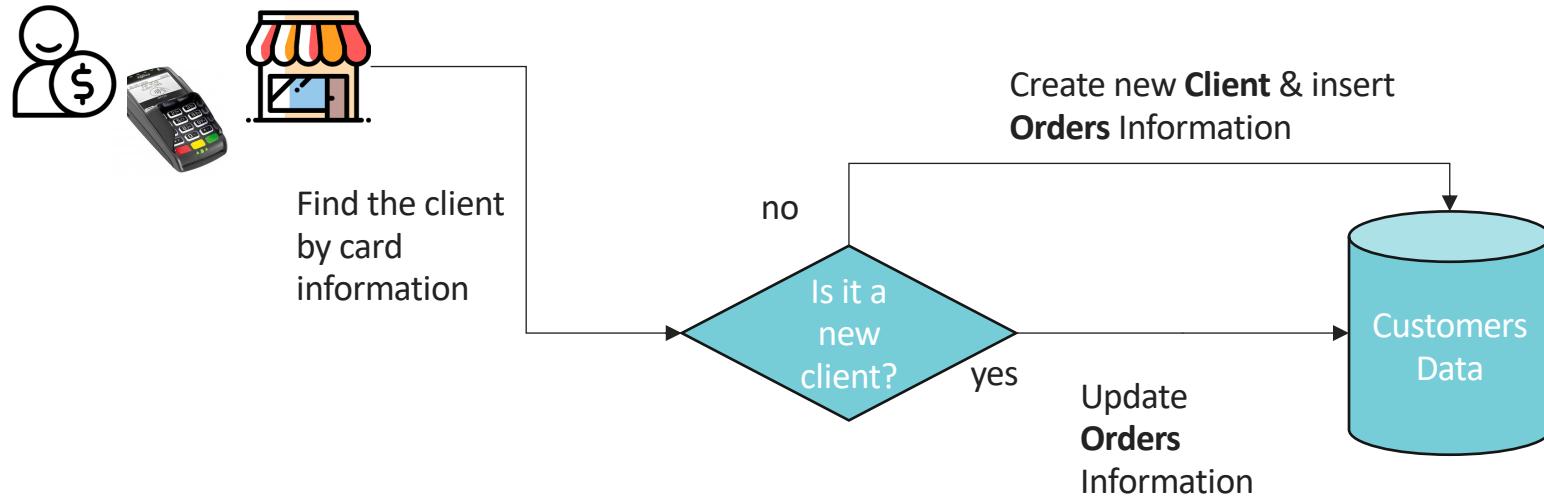
Pipeline – find the client by credit card number



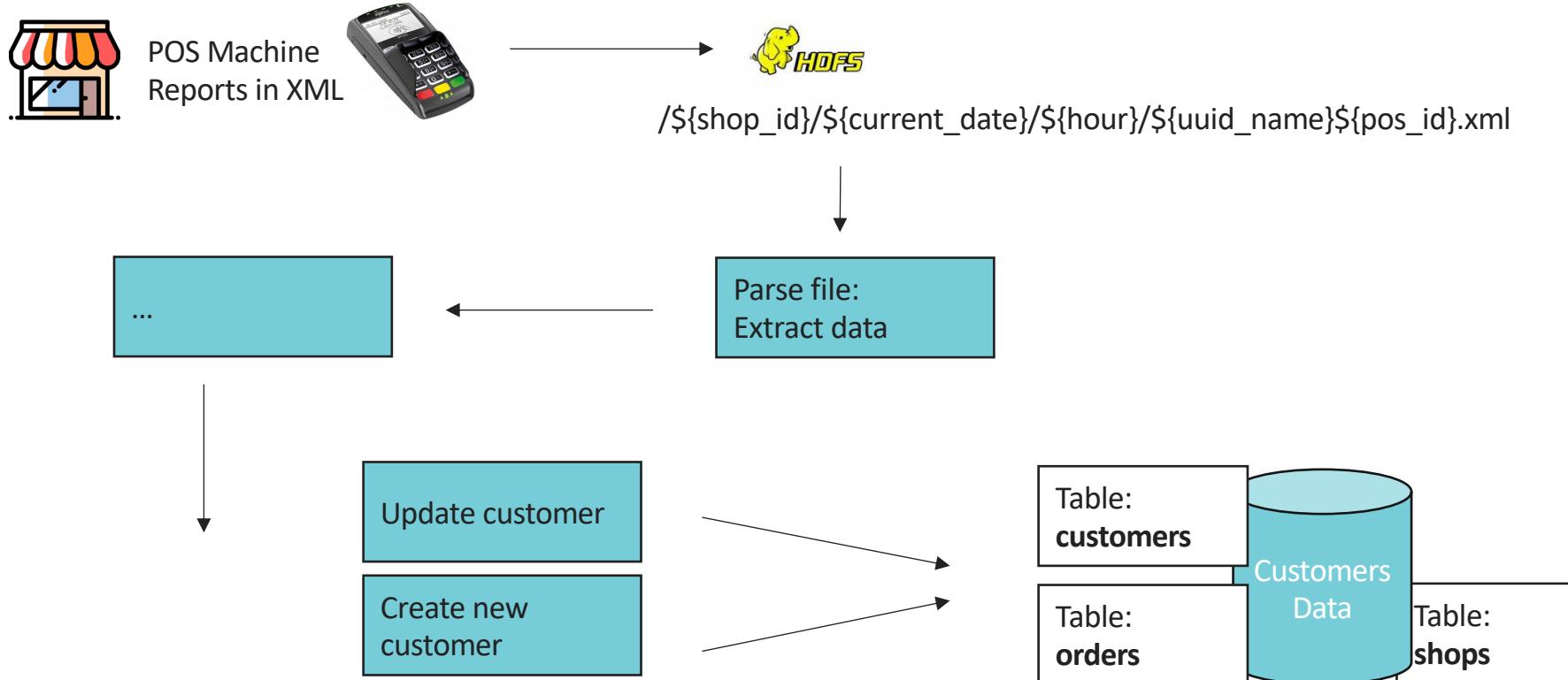
Pipeline – find the client by credit card number



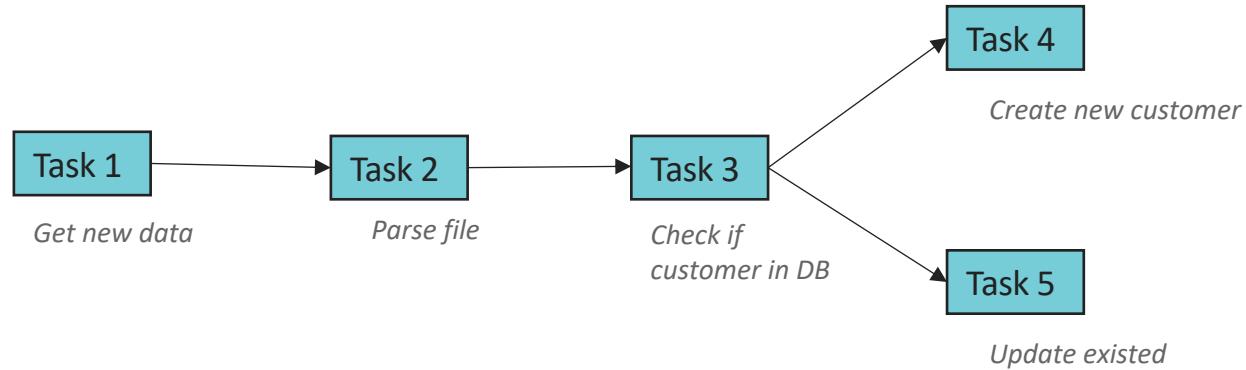
Pipeline – find the client by credit card number



Pipeline – find the client by credit card number



Abstract visualization



Congratulations!
We just described a **Workflow** or a **Pipeline**...

Or **DAG** – main concept
of Apache Airflow



Some key characteristic of Pipelines

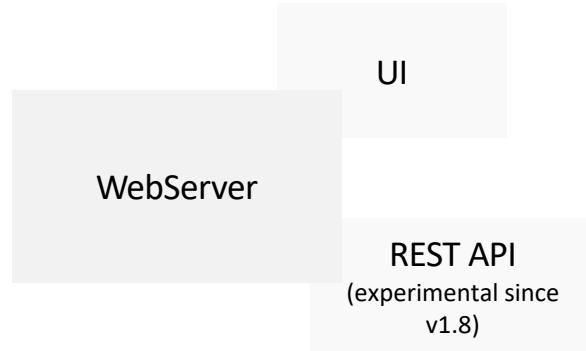
1. **Schedule:** They run in time with different schedule, duration, etc
2. **Triggers:** Pipelines can have Triggers that cause need to run the pipeline
3. **Fails:** Pipelines can fail. We need 1) to know about it 2) to get possible start it from failed place – and this is why your task must be atomic and small
4. **Re-processing:** Sometimes you need reprocess data for whole long periods in past
5. Sometimes fails can be because of network or system issues and you want to have auto **retries**

Before we will create our first DAG –
let's up & run Apache Airflow Server

Disclaimer:

all information relative to Apache Airflow
actual on **version 1.10.12** and can be different on ver 2.0+

High-level overview of Apache Airflow components



High-level overview of Apache Airflow components

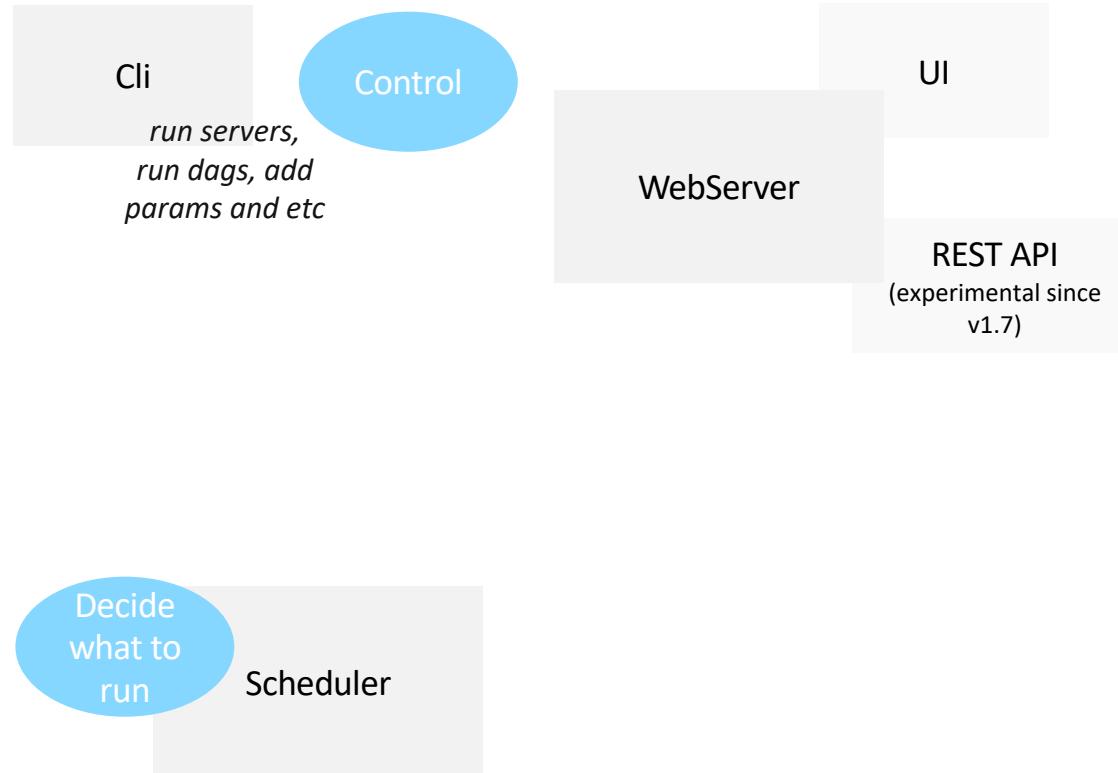


<https://airflow.apache.org/docs/apache-airflow/stable/cli-ref>

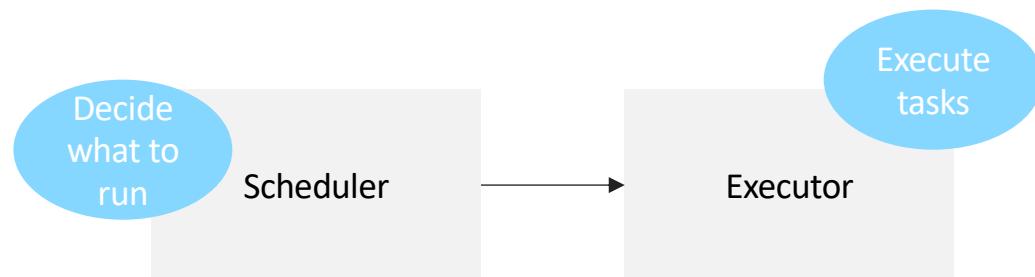
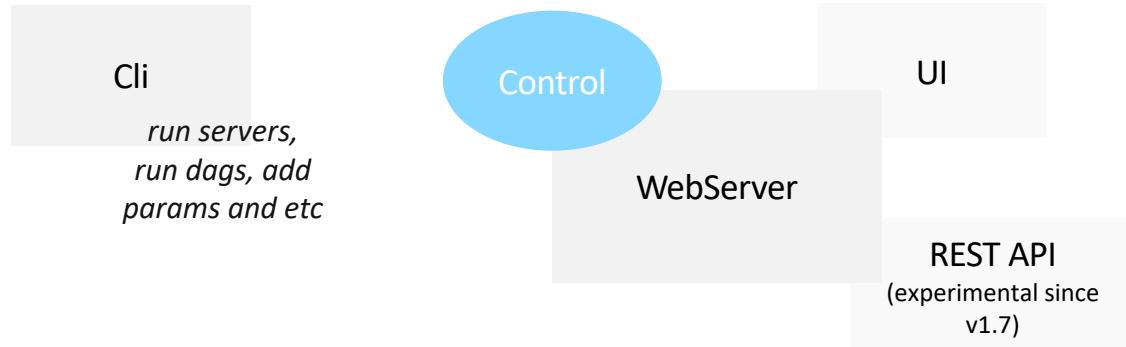
Server commands:

airflow initdb
airflow webserver
airflow scheduler and etc.

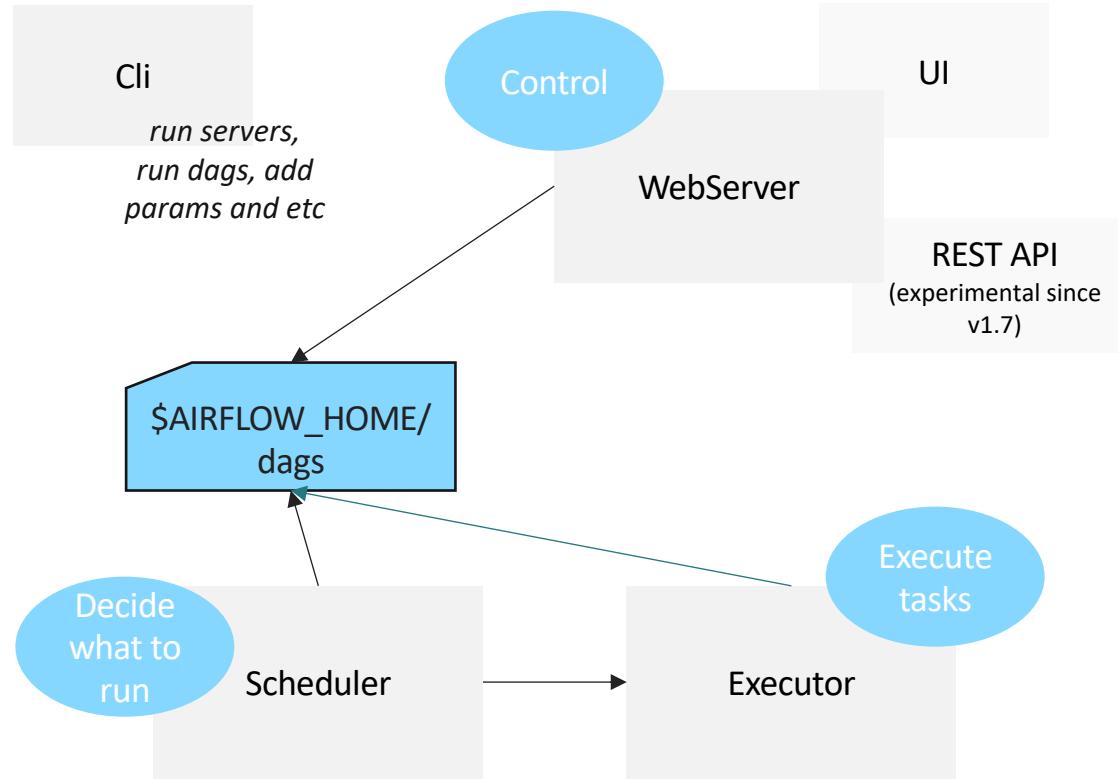
High-level overview of Apache Airflow components



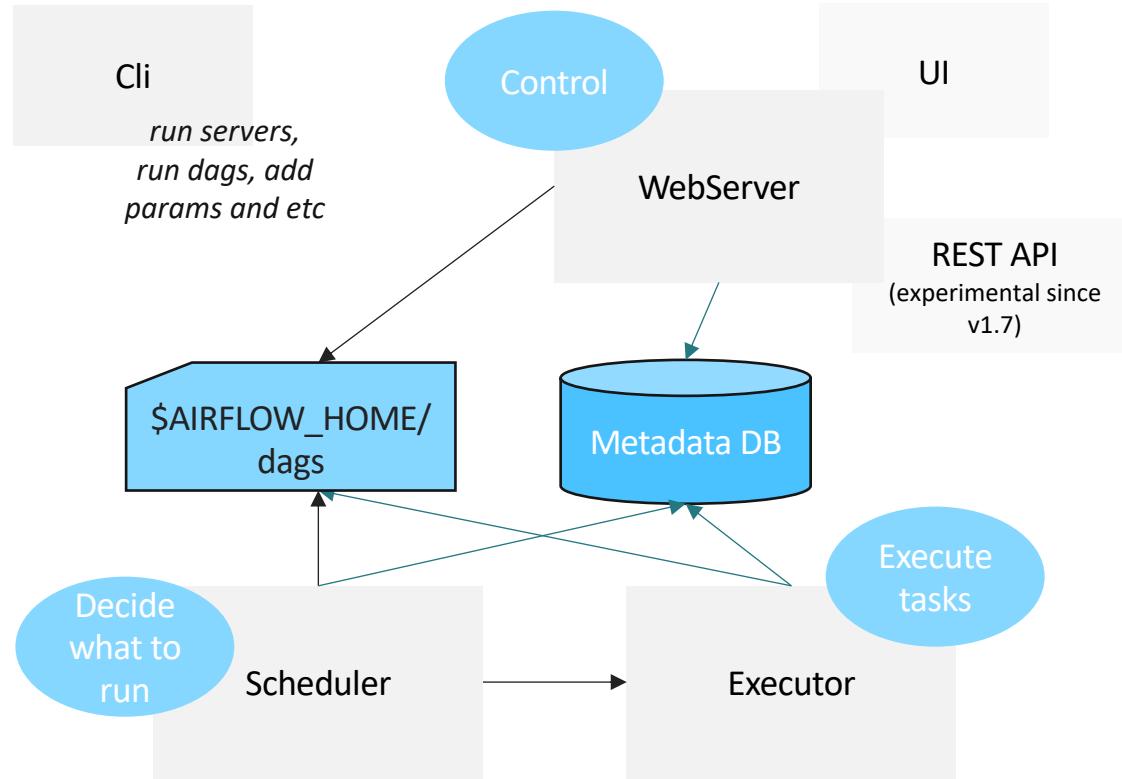
High-level overview of Apache Airflow components



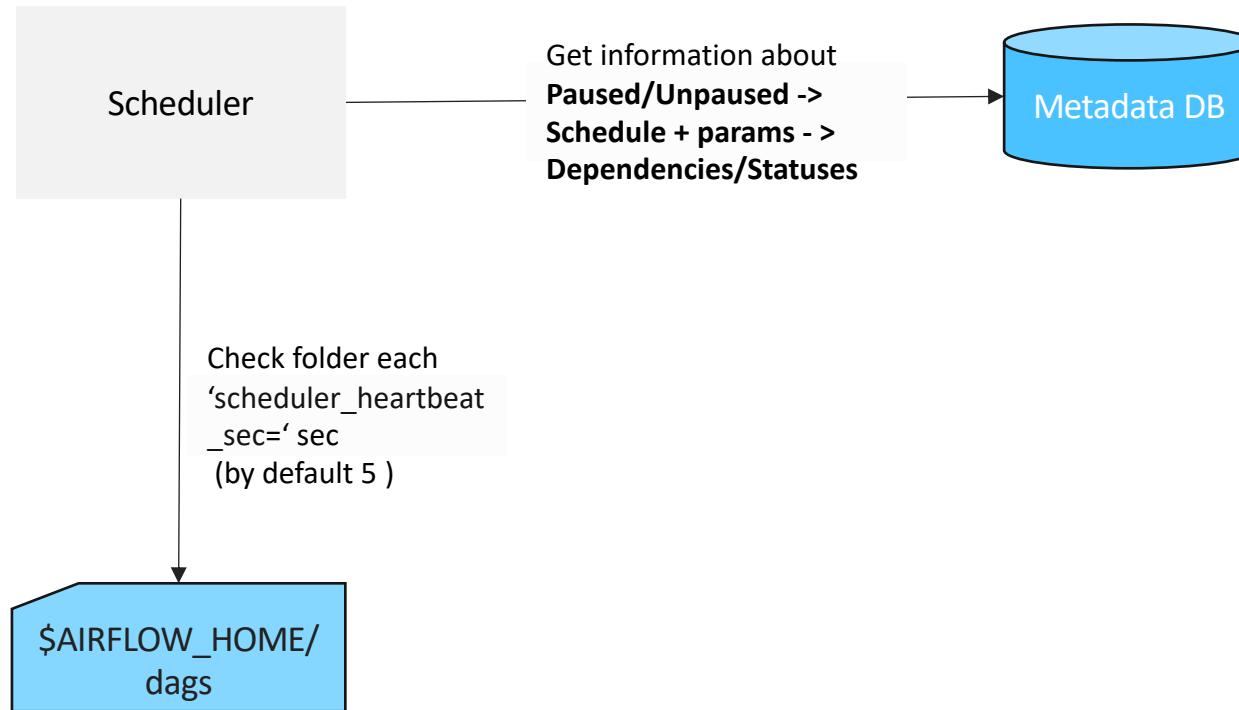
High-level overview of Apache Airflow components



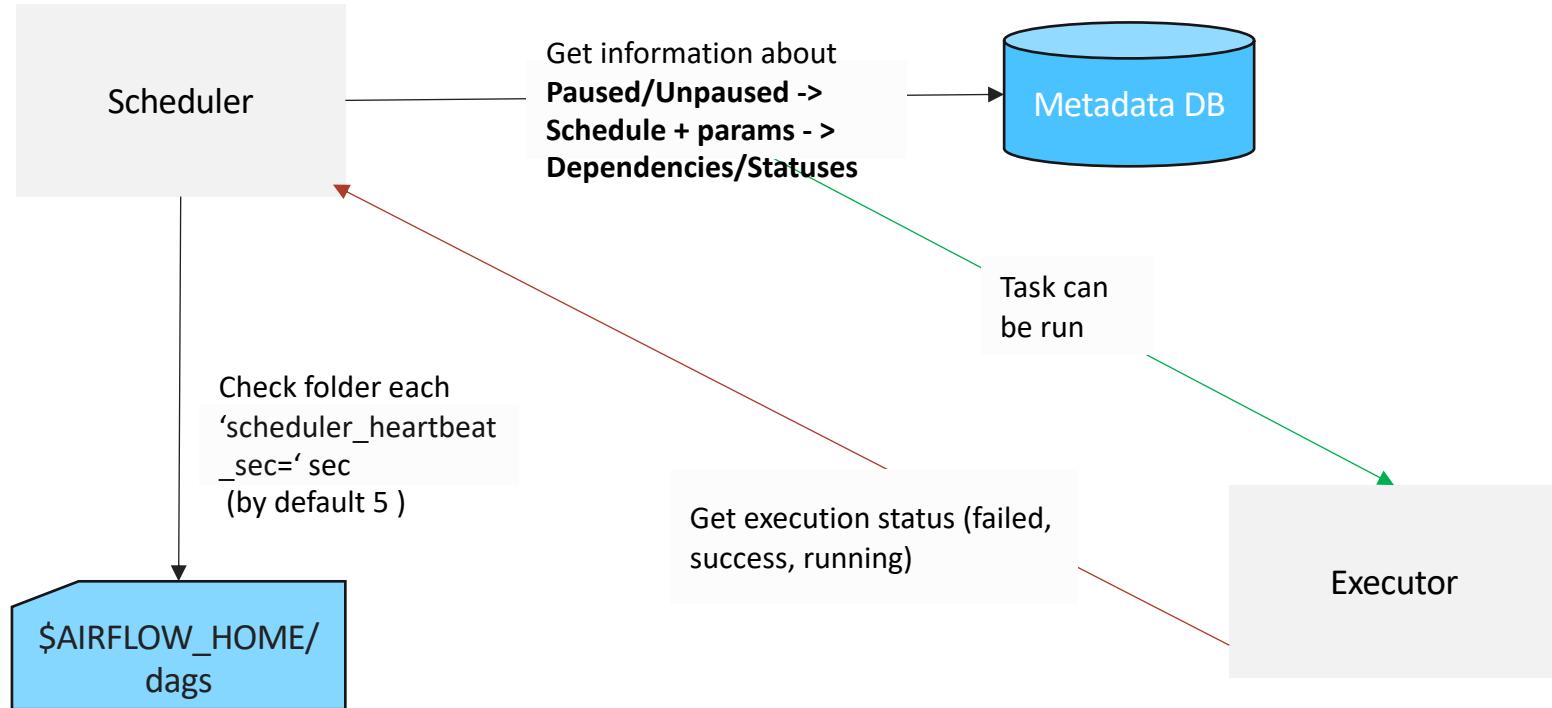
High-level overview of Apache Airflow components



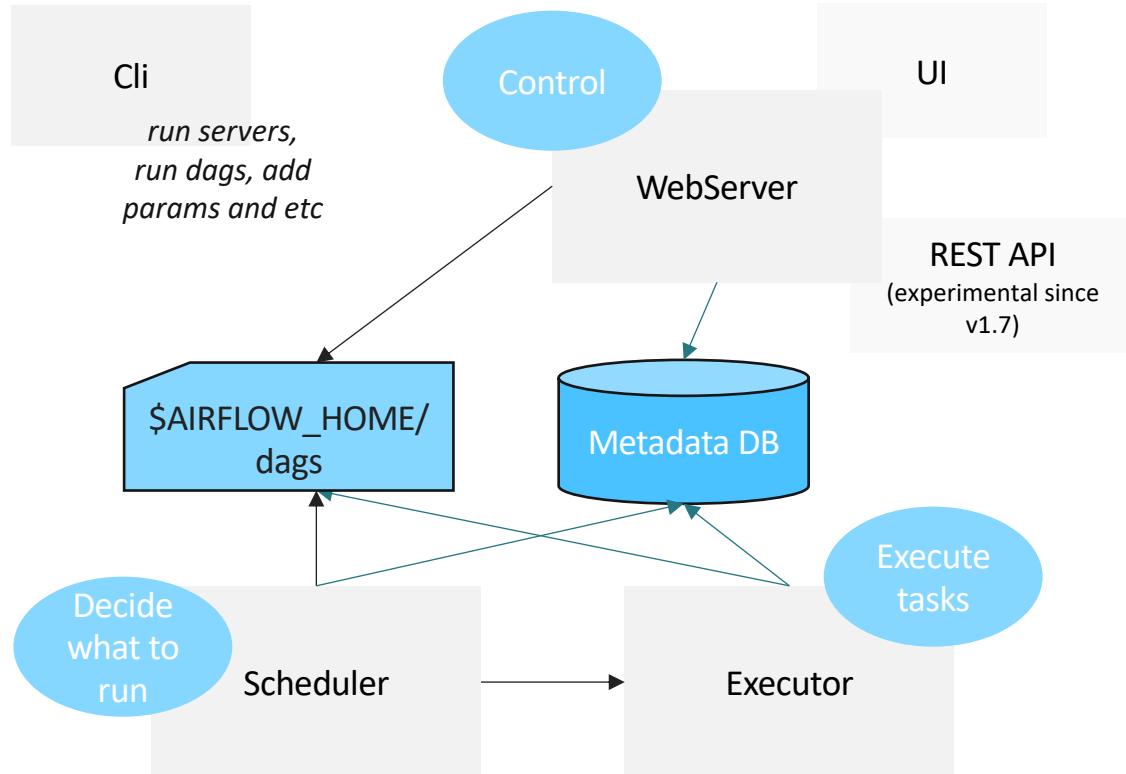
Process of DAG execution



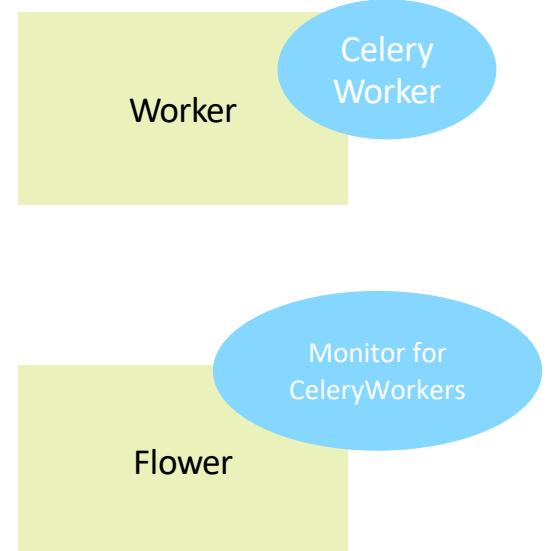
Process of DAG execution



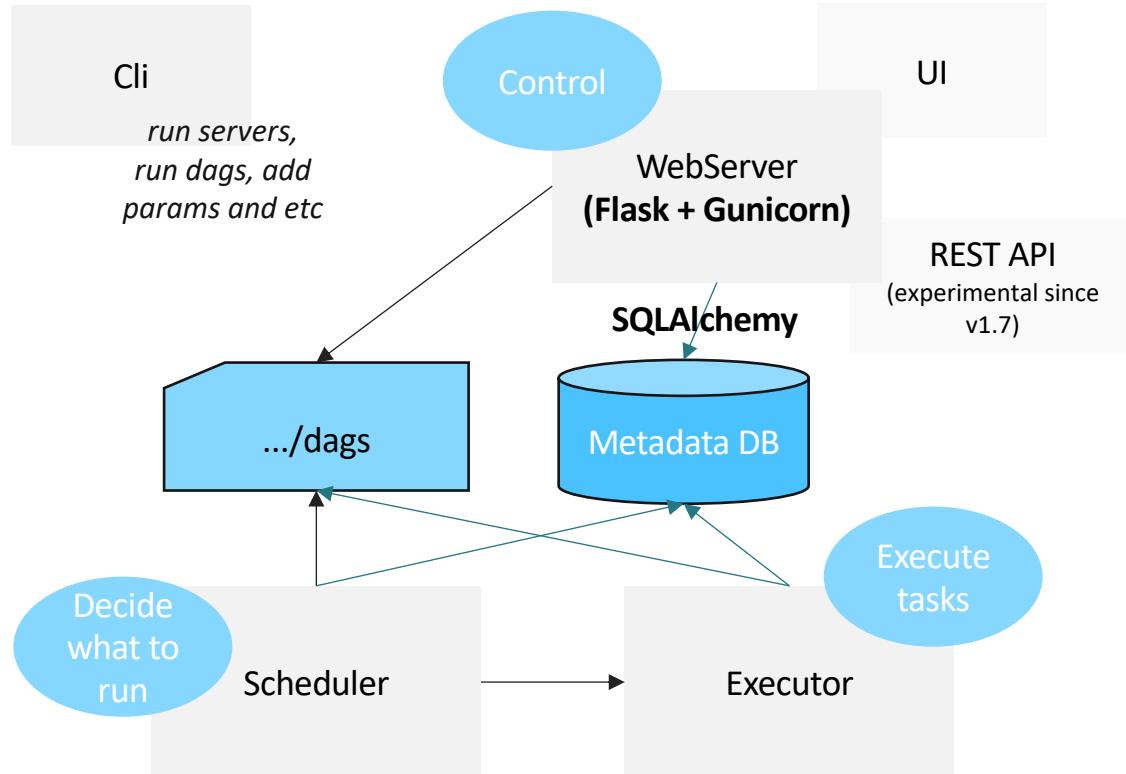
High-level overview of Apache Airflow components



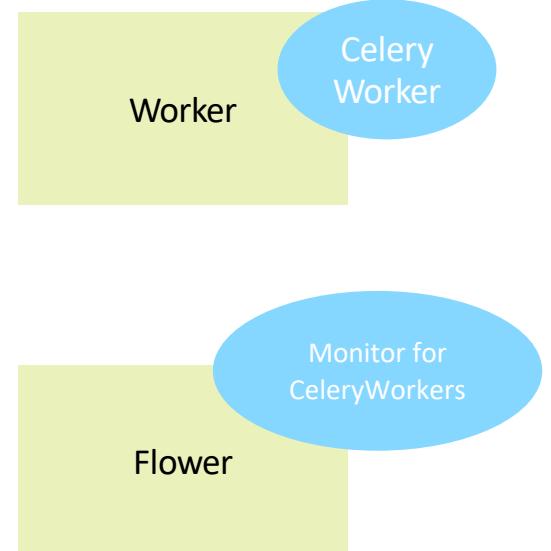
If you work with CeleryExecutor



High-level overview of Apache Airflow components



If you work with CeleryExecutor



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

The screenshot shows the Airflow UI interface. At the top, there is a navigation bar with links for Airflow, DAGs, Data Profiling, Browse, Admin, Docs, and About. The main content area is titled "DAGs". On the left, a sidebar lists "DAG" and "consume_new_data_from_pos". The main panel displays the DAG details for "consume_new_data_from_pos". The DAG is currently "Off". The title bar includes "Graph View", "Tree View", "Task Duration", "Task Tries", "Landing Times", "Gantt", "Details", and "Code" buttons. Below the title, there are filters for "None", "Base date: 2020-12-03 22:50:50", "Number of runs: 25", "Run: [dropdown]", "Layout: Left->Right", and a "Go" button. A "DummyOperator" task is listed with status indicators: "scheduled" (orange), "skipped" (pink), and "upstream_failed" (yellow). The DAG structure shows two tasks: "get_new_data" and "parse_file".

Define a sequence of tasks

```
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
```

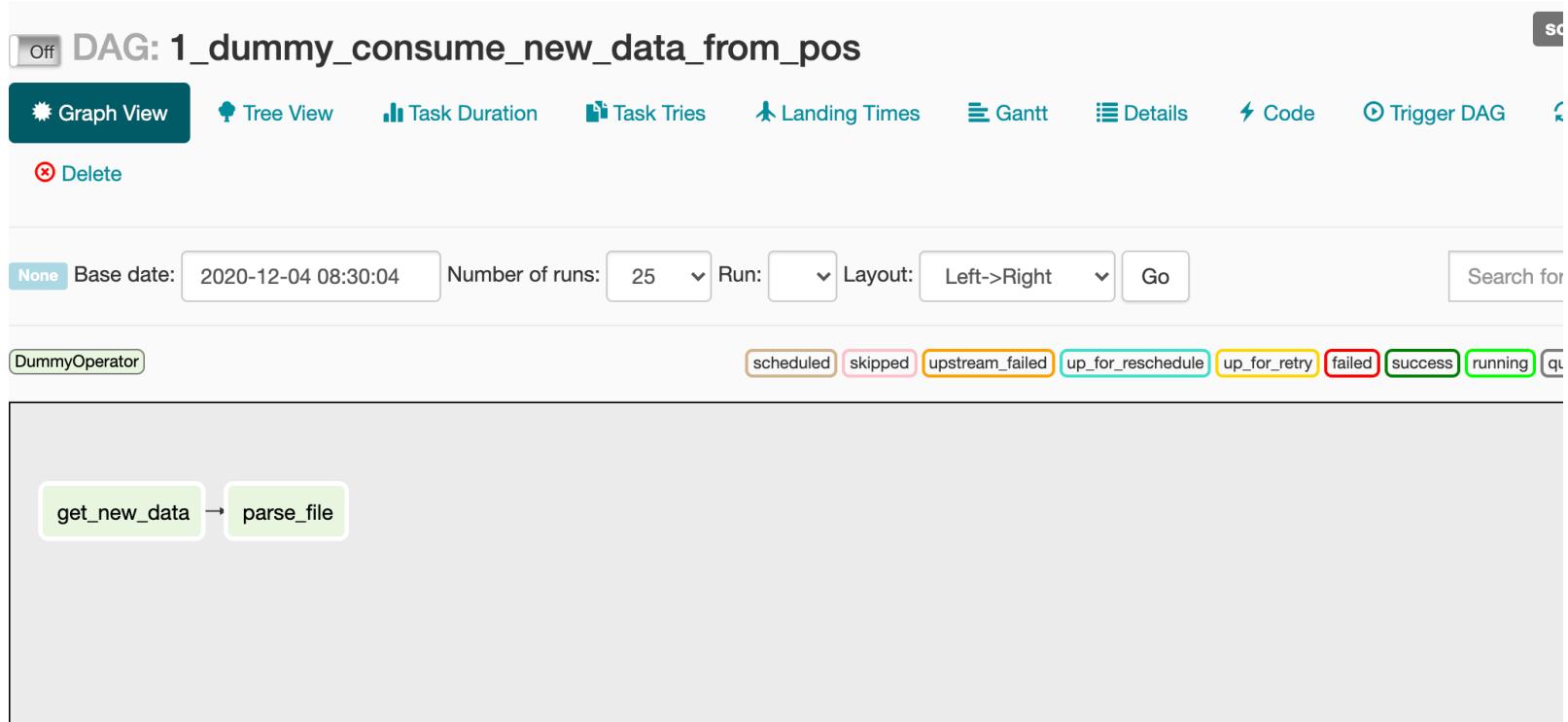
set_downstream

>>

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")

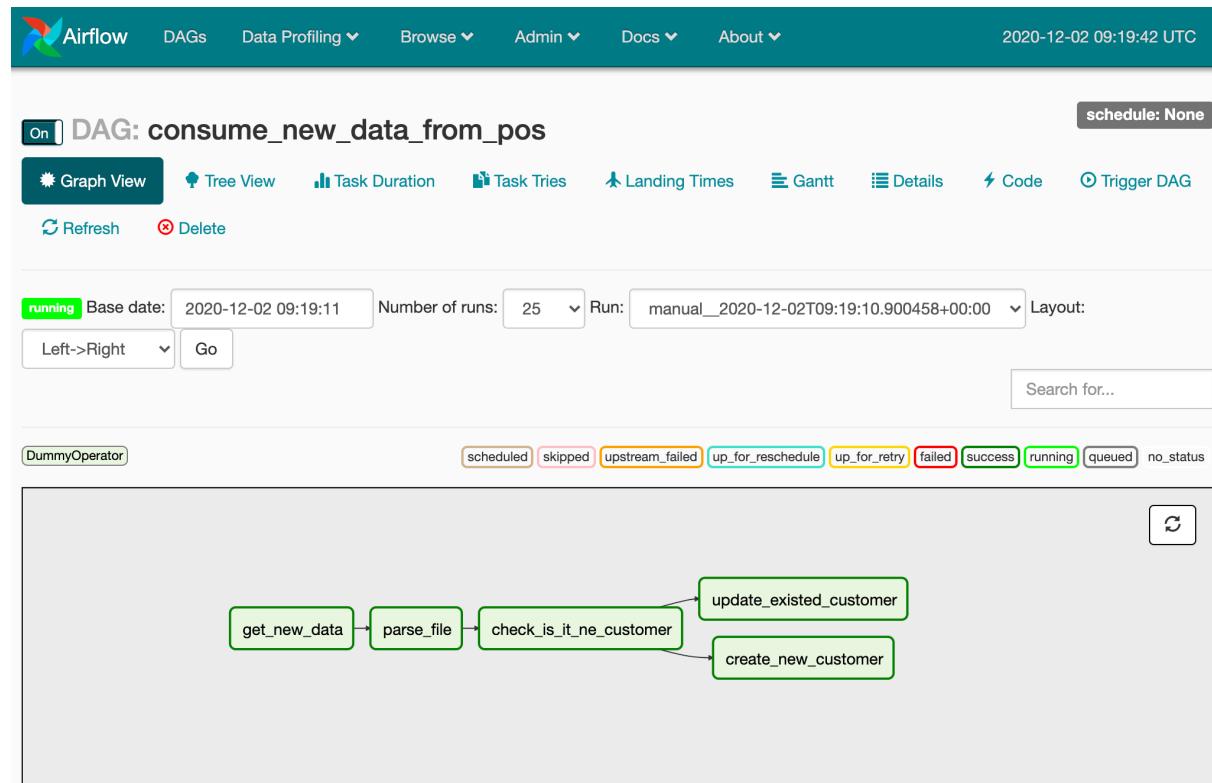
    check_is_it_ne_customer = DummyOperator(task_id="check_is_it_ne_customer")

    create_new_customer = DummyOperator(task_id="create_new_customer")

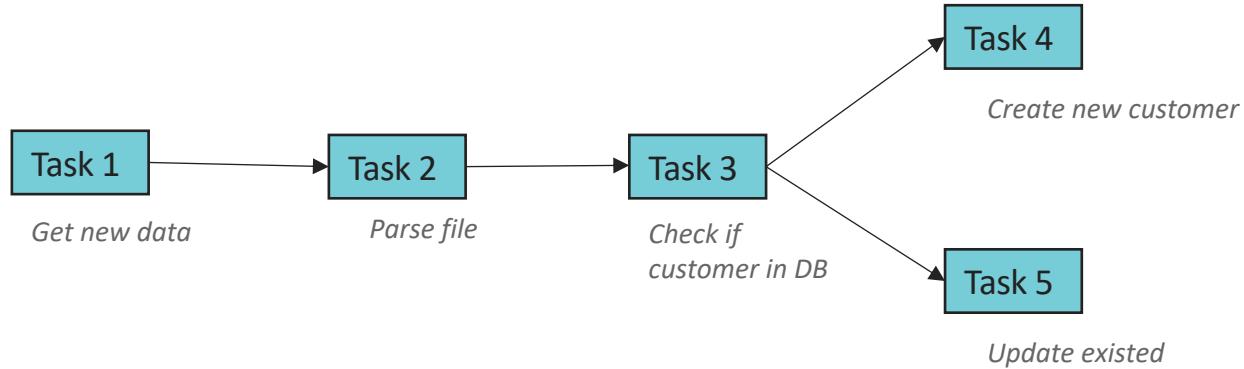
    update_existed_customer = DummyOperator(task_id="update_existed_customer")

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

Apache Airflow UI



DAG – Directed Acyclic Graph



What can be a Task?

Operators

Just DO right now
– report about completion

Sensors

Poke(wait) condition
until try

What can be a Task?

Operators

Just DO right now

– report about completion

Examples:

- [FileToGoogleCloudStorageOperator](#)
- [MySqlOperator](#)
- [AWSAthenaOperator](#)

...

Sensors

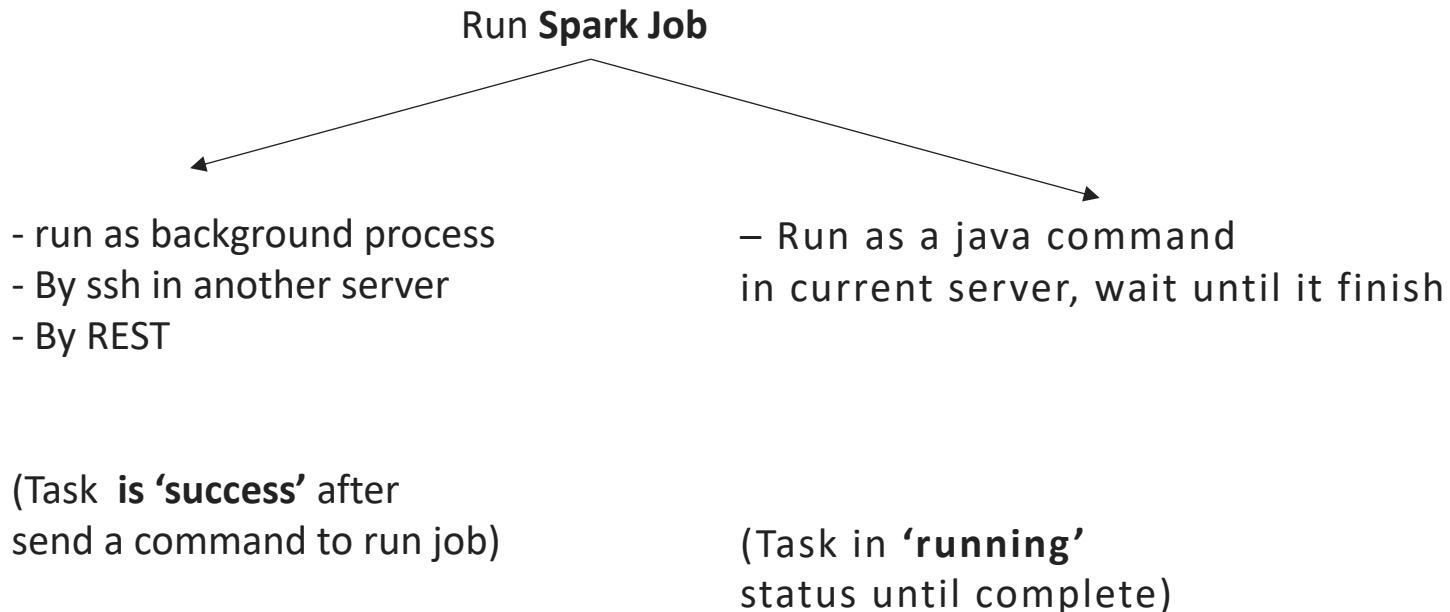
Poke(wait) condition
until try

Examples:

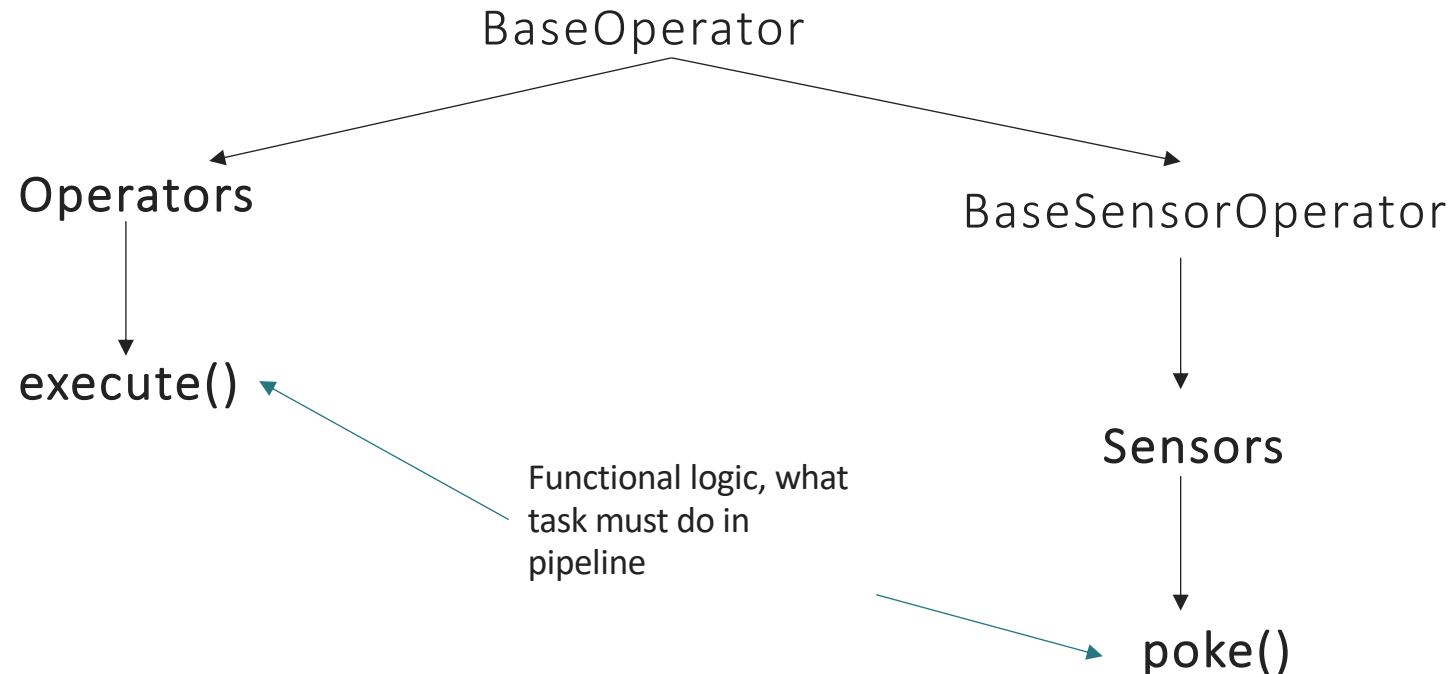
- [HdfsSensor](#)
- [HttpSensor](#)
- [SqlSensor](#)

...

Moment of Task Completion



What can be a Task?



Let's define our primitive Operator

```
from typing import Union, Iterable, Dict

from airflow.models import BaseOperator, SkipMixin

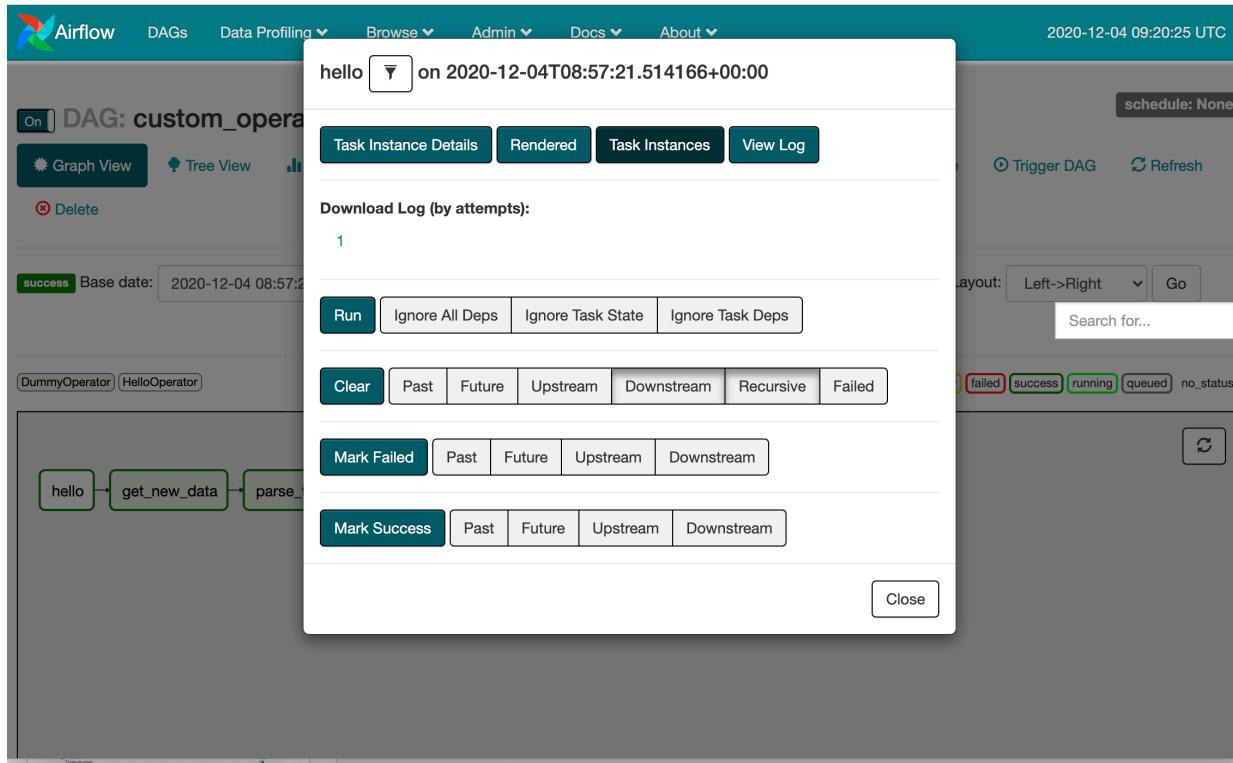
class HelloOperator(BaseOperator, SkipMixin):

    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

The screenshot shows the Airflow UI interface. At the top, there is a navigation bar with links for DAGs, Data Profiling, Browse, Admin, Docs, and About, along with a timestamp of 2020-12-04 09:32:41 UTC. Below the navigation bar, a message indicates a task instance has been deleted. The main content area displays a task instance for a DAG named 'hello' at the time 2020-12-04 08:57:21. The interface includes tabs for Task Instance Details, Rendered Template, Log (which is currently selected), and XCom. The Log section shows log entries categorized by attempt number (attempt 1). The log details the execution of a custom operator named 'HelloOperator' for the task 'hello'. It shows the operator being executed, the task runner starting the process, and the task running successfully, outputting 'Hello, World!'.

```
*** Reading local file: /Users/iuliiia_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 2020-12-04T08:57:21.514166+00:00>
[2020-12-04 11:57:27,573] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.514166+00:00>
[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', '2020-12-04T08:57:21.514166+00:00']
[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+00:00>
[2020-12-04 11:57:27,655] {logging_mixin.py:112} WARNING - /Users/iuliiia_volkova2/Library/Caches/pypoetry/virtualenvs/kkr-data-access-x2mMbnzr
  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=20201204T08:57:21.514166+00:00
[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

The screenshot shows the Airflow UI interface. At the top, there is a navigation bar with links for DAGs, Data Profiling, Browse, Admin, Docs, and About, along with a timestamp of 2020-12-04 09:32:41 UTC. Below the navigation bar, there is a red 'Delete' button. The main content area displays a task instance named 'hello' from December 4, 2020, at 08:57:21. There are four tabs: Task Instance Details, Rendered Template, Log (which is selected and highlighted in green), and XCom. The Log tab shows the log output for the task instance. The log output is as follows:

```
*** Reading local file: /Users/iuliiia_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 2020-12-04T08:57:21.514166+00:00>
[2020-12-04 11:57:27,573] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: custom_operator.hello 2020-12-04T08:57:21.514166+00:00>
[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', '2020-12-04T08:57:21.514166+00:00']
[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+00:00>
[2020-12-04 11:57:27,655] {logging_mixin.py:112} WARNING - /Users/iuliiia_volkova2/Library/Caches/pypoetry/virtualenvs/kkr-data-access-x2mMbnzr
  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=20201204T08:57:21.514166+00:00
[2020-12-04 11:57:32,564] {local_task_job.py:102} INFO - Task exited with return code 0
```

Let's play with Tasks menu

1. Task Details
2. Task logs
3. Task clean up / change status
4. Let's check **Browse** menu also

What we already saw in Airflow

1. Server components
2. How to define DAG with minimal params
3. How to define tasks
4. What is Operator & Sensors
5. How to define custom Operator
6. Tasks Details & Logs in UI
7. Browse menu in UI

Dynamical DAGs generation: DAGFile & multiple DAGs in one file

DAGFile != DAG

```
...
def create_dag( ... ) -> DAG:
    ...

# build a dag for each number in range(10)
for n in range(1, 10):
    dag_id = 'hello_world_{}'.format(str(n))
    params = {'dag_id': dag_id,
              'schedule_interval': None,
              'start_date': datetime(2020, 12, 1)}
    dag_number = n
    globals()[dag_id] = create_dag(dag_number, params)
```

DAGFile & multiple DAGs in one file

DAGs

		DAG	Schedule	Owner	Recent Tasks	Last Run	D
<input checked="" type="checkbox"/>		consume_new_data_from_pos		airflow	2	2020-12-03 22:53	
<input checked="" type="checkbox"/>		hello_world_1		airflow			
<input checked="" type="checkbox"/>		hello_world_2		airflow			
<input checked="" type="checkbox"/>		hello_world_3		airflow			
<input checked="" type="checkbox"/>		hello_world_4		airflow			
<input checked="" type="checkbox"/>		hello_world_5		airflow			
<input checked="" type="checkbox"/>		hello_world_6		airflow			
<input checked="" type="checkbox"/>		hello_world_7		airflow			
<input checked="" type="checkbox"/>		hello_world_8		airflow			
<input checked="" type="checkbox"/>		hello_world_9		airflow			

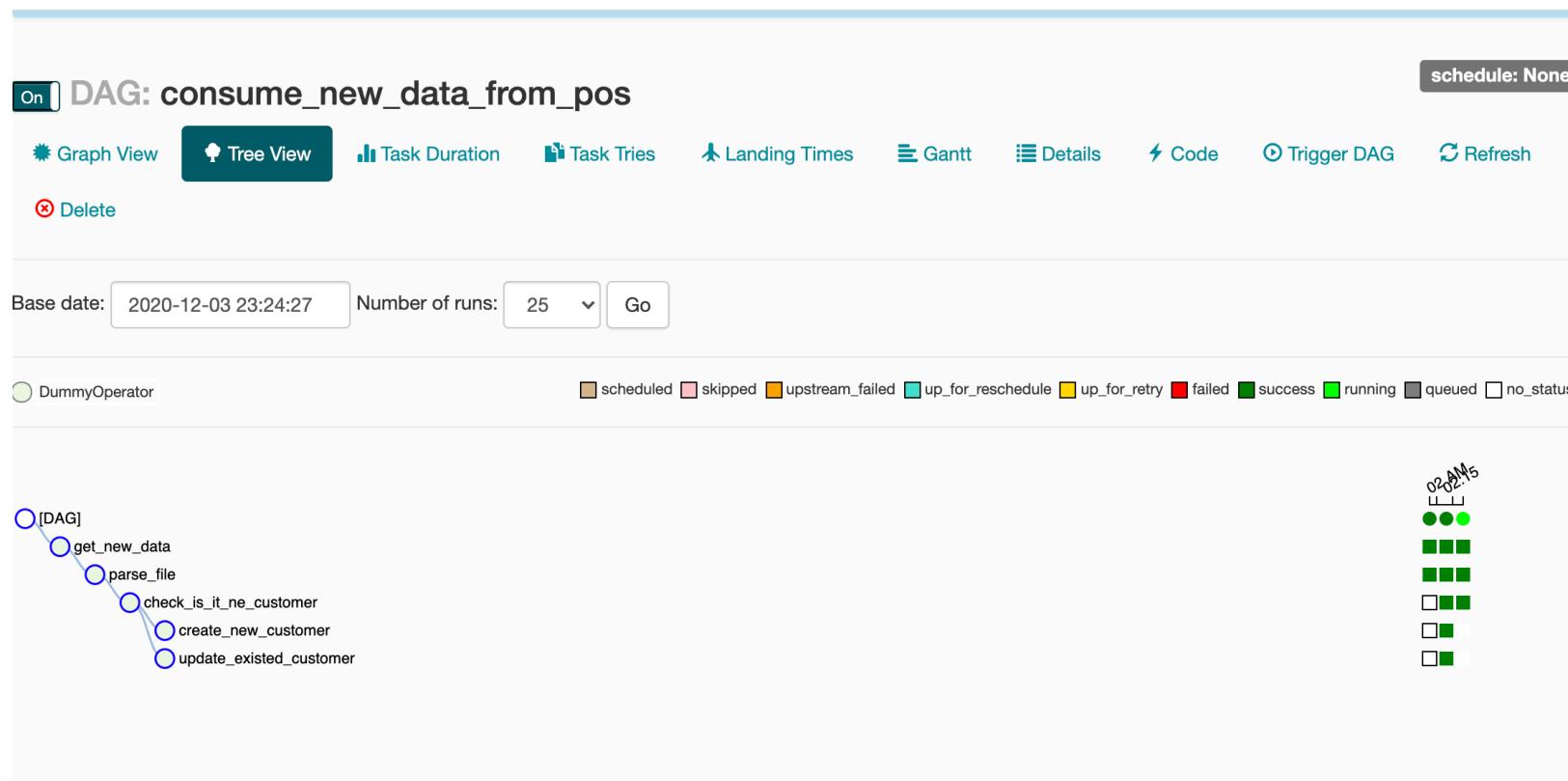
Open in UI

Dynamical DAGs generation: read properties from yaml or JSON

1. Simple read & parse with Pure Python code – custom config/DAG readers
2. Existed third-party libraries/solutions, for example:

<https://github.com/rambler-digital-solutions/airflow-declarative>

DAG Runs & Task Instances



DAG Runs & Task Instances



DAGs

Data Profiling ▾

Browse ▾

Admin ▾

Docs ▾

About ▾

2020-12-03 23:24:54 UTC

Dag Runs

List (3)		Create	Add Filter ▾	With selected ▾	Search: dag_id, state, run_id	
		State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		success	consume_new_data_from_pos	12-03T23:24:27.035263+00:00	manual__2020-12-03T23:24:27.035263+00:00	<input checked="" type="checkbox"/>
<input type="checkbox"/>		success	consume_new_data_from_pos	12-03T23:23:52.997410+00:00	manual__2020-12-03T23:23:52.997410+00:00	<input checked="" type="checkbox"/>
<input type="checkbox"/>		success	consume_new_data_from_pos	12-03T22:53:20.570274+00:00	manual__2020-12-03T22:53:20.570274+00:00	<input checked="" type="checkbox"/>

<http://localhost:8080/admin/dagrun/>

Let's change the schedule_interval

Set `schedule_interval="@daily"` or `"0 0 * * *"`

Airflow support **CRON** expressions:

<https://airflow.apache.org/docs/apache-airflow/stable/dag-run.html#cron-presets>

`@daily` – just cron preset from Airflow

Let's change the schedule_interval

Airflow DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾ 2020-12-04 09:57:19 UTC

On DAG: schedule_daily_dag schedule: 0 0 * * *

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

Base date: 2020-12-01 00:00:00 Number of runs: 25 Go

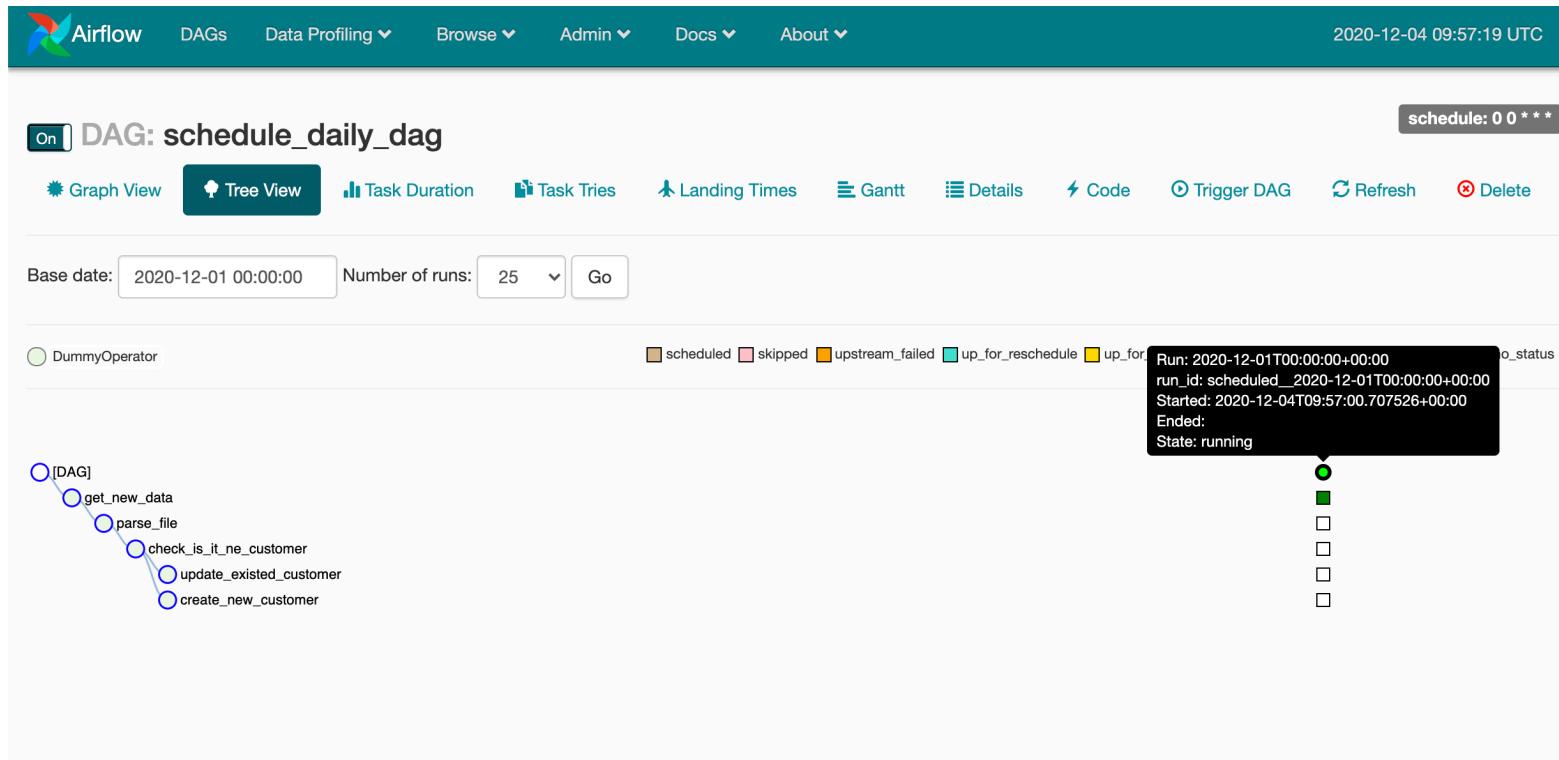
DummyOperator

[DAG] get_new_data parse_file check_is_it_ne_customer update_existed_customer create_new_customer

Run: 2020-12-01T00:00:00+00:00
run_id: scheduled__2020-12-01T00:00:00+00:00
Started: 2020-12-04T09:57:00.707526+00:00
Ended:
State: running

```
graph TD; DAG([DAG]) --> get_new_data((get_new_data)); get_new_data --> parse_file((parse_file)); parse_file --> check_is_it_ne_customer((check_is_it_ne_customer)); check_is_it_ne_customer --> update_existed_customer((update_existed_customer)); update_existed_customer --> create_new_customer((create_new_customer))
```

Let's change the schedule_interval



Let's change the schedule_interval

The screenshot shows the Airflow web interface with the following details:

- Header:** Airflow logo, DAGs, Data Profiling, Browse, Admin, Docs, About, and the timestamp 2020-12-04 10:01:36 UTC.
- Section:** Dag Runs
- Table Headers:** List (7), Create, Add Filter, With selected, and a search bar.
- Table Data:** A table with columns: State, Dag Id, Execution Date, Run Id, and External Trigger.
- Rows:** There are seven rows, each with a checkbox, edit icon, state (success), dag id, execution date, run id, and external trigger status.
- Highlighted Row:** The last three rows, which all have 'schedule_daily_dag' as the dag_id, are highlighted with a blue border around their 'Execution Date' column values: 12-03T00:00:00+00:00, 12-02T00:00:00+00:00, and 12-01T00:00:00+00:00.

Let's change the schedule_interval

The screenshot shows the Airflow interface with the following details:

- Header:** Airflow logo, DAGs, Data Profiling, Browse, Admin, Docs, About, and timestamp 2020-12-04 10:03:11 UTC.
- Section Title:** Task Instances.
- Table Headers:** List (30), Add Filter, With selected, and a search bar for dag_id, task_id, state.
- Table Columns:** State, Dag Id, Task Id, Execution Date, Operator, Start Date, and End Date.
- Data Rows:** There are five rows of task instances. The first row is for a custom operator 'hello' in the 'custom_operator' DAG. The subsequent four rows are for tasks in the 'schedule_daily_dag' DAG: 'update_existed_customer', 'create_new_customer', 'check_is_it_ne_customer', and 'parse_file'. All tasks show a 'success' status and were run at 12-03T00:00:00+00:00.

	State	Dag Id	Task Id	Execution Date	Operator	Start Date	End Date
<input type="checkbox"/>	success	custom_operator	hello ▾	12-04T08:57:21.514166+00:00	HelloOperator	12-04T08:57:27.563872+00:00	12-04T08:57:27.658
<input type="checkbox"/>	success	schedule_daily_dag	update_existed_customer ▾	12-03T00:00:00+00:00	DummyOperator	12-04T09:58:10.950436+00:00	12-04T09:58:10.950
<input type="checkbox"/>	success	schedule_daily_dag	create_new_customer ▾	12-03T00:00:00+00:00	DummyOperator	12-04T09:58:10.947758+00:00	12-04T09:58:10.947
<input type="checkbox"/>	success	schedule_daily_dag	check_is_it_ne_customer ▾	12-03T00:00:00+00:00	DummyOperator	12-04T09:57:56.908326+00:00	12-04T09:57:56.908
<input type="checkbox"/>	success	schedule_daily_dag	parse_file ▾	12-03T00:00:00+00:00	DummyOperator	12-04T09:57:42.927844+00:00	12-04T09:57:42.927

Execution_date, start_date & schedule_interval

Triggered Manually:

execution_date - triggering datetime (if DAG was triggered manually)

Scheduled:

execution_date - date for that we process the **Data**, not real start or task

start_date - date of data that we want to process

if start_date (2020, 12, 1) and schedule “@daily” “@hourly” 00:01

**For the first time DAG will run on 2020, 12, 2, execution_date will be 2020, 12, 1,
but start_date of tasks will contain real execution datetime**

Let's change the schedule_interval

Set `schedule_interval="@daily"` or `"0 0 * * *"`

Airflow support **CRON** expressions:

<https://airflow.apache.org/docs/apache-airflow/stable/dag-run.html#cron-presets>

`@daily` – just cron preset from Airflow

Catch Up

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

with DAG(
    dag_id="no_catch_up_schedule_daily_dag",
    start_date=datetime(2020, 12, 1),
    schedule_interval="0 0 * * *",
    catchup=False
) as dag:
```

:param **catchup**: Perform scheduler catchup (or only run latest)?
Defaults to True

Catch Up

Screenshot of the Airflow UI showing Dag Runs and a detailed view of a specific run.

The top navigation bar shows the Airflow logo, DAGs, Data Profiling, Browse, Admin, Docs, and About. The date is 2020-12-04 10:16:34 UTC. A red box highlights the timestamp "2020-12-04 10:16:43 UTC" in the top right corner of the header.

The main page title is "Dag Runs". Below it are buttons for "List (8)", "Create", "Add Filter", and "With selected". A search bar is present. The table has columns: State, Dag Id, Execution Date, Run Id, and External Trigger.

	State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		success custom_operator	12-04T08:57:21.514166+00:00	manual__2020-12-04T08:57:21.514166+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T23:24:27.035263+00:00	manual__2020-12-03T23:24:27.035263+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T23:23:52.997410+00:00	manual__2020-12-03T23:23:52.997410+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T22:53:20.570274+00:00	manual__2020-12-03T22:53:20.570274+00:00	
<input type="checkbox"/>		success schedule_daily_dag	12-03T00:00:00+00:00	scheduled__2020-12-03T00:00:00+00:00	
<input type="checkbox"/>		success no_catch_up_schedule_daily_dag	12-03T00:00:00+00:00	scheduled__2020-12-03T00:00:00+00:00	
<input type="checkbox"/>		success schedule_daily_dag	12-02T00:00:00+00:00	scheduled__2020-12-02T00:00:00+00:00	
<input type="checkbox"/>		success schedule_daily_dag	12-01T00:00:00+00:00	scheduled__2020-12-01T00:00:00+00:00	

A tooltip for the "schedule_daily_dag" run on December 3rd shows the following details:

```
Run: 2020-12-03T00:00:00+00:00
run_id: scheduled__2020-12-03T00:00:00+00:00
Started: 2020-12-04T10:14:57.477411+00:00
Ended: 2020-12-04T10:16:01.544920+00:00
State: success
```

The bottom right corner of the page shows the number 92.

Catch Up

Screenshot of the Airflow UI showing Dag Runs and a detailed view of a specific run.

The top navigation bar shows the Airflow logo, DAGs, Data Profiling, Browse, Admin, Docs, and About. The date is 2020-12-04 10:16:34 UTC. A red box highlights the timestamp "2020-12-04 10:16:43 UTC" in the top right corner of the header.

The main page title is "Dag Runs". Below it are buttons for "List (8)", "Create", "Add Filter", and "With selected". A search bar is present. The table has columns: State, Dag Id, Execution Date, Run Id, and External Trigger.

	State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		success custom_operator	12-04T08:57:21.514166+00:00	manual__2020-12-04T08:57:21.514166+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T23:24:27.035263+00:00	manual__2020-12-03T23:24:27.035263+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T23:23:52.997410+00:00	manual__2020-12-03T23:23:52.997410+00:00	
<input type="checkbox"/>		success consume_new_data_from_pos	12-03T22:53:20.570274+00:00	manual__2020-12-03T22:53:20.570274+00:00	
<input type="checkbox"/>		errored schedule_daily_dag	12-03T00:00:00+00:00	scheduled__2020-12-03T00:00:00+00:00	
<input type="checkbox"/>		success no_catch_up_schedule_daily_dag	12-03T00:00:00+00:00	scheduled__2020-12-03T00:00:00+00:00	
<input type="checkbox"/>		success schedule_daily_dag	12-02T00:00:00+00:00	scheduled__2020-12-02T00:00:00+00:00	
<input type="checkbox"/>		success schedule_daily_dag	12-01T00:00:00+00:00	scheduled__2020-12-01T00:00:00+00:00	

A tooltip for the "schedule_daily_dag" run on December 3rd shows the following details:

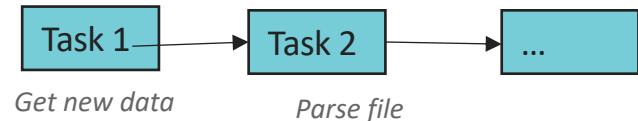
```
Run: 2020-12-03T00:00:00+00:00
run_id: scheduled__2020-12-03T00:00:00+00:00
Started: 2020-12-04T10:14:57.477411+00:00
Ended: 2020-12-04T10:16:01.544920+00:00
State: success
```

The bottom right corner of the page shows the number 93.

Parametrize tasks – Jinja2 Template



`/${shop_id}/${current_date}/${hour}/${uuid_name}${pos_id}.xml`



Let simplify it for study pipeline:

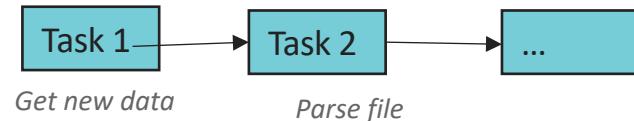
Use - FS, not HDFS and fix shop_id and file name data.json :

`../shop123/{{ current_date }}/${hour}/data.json`

`{'param_name': value}`

Parametrize tasks – Jinja2 Template

Get new data – Sensor that wait for the file,
to avoid failed ('parse_file') task



```
from airflow.contrib.sensors.file_sensor import FileSensor  
  
from airflow.operators.dummy_operator import DummyOperator
```

Parametrize tasks – Jinja2 Template

Get new data – Sensor that wait for the file,
to avoid failed ('parse_file') task



```
from airflow.contrib.sensors.file_sensor import FileSensor
```

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

FileSensor expect **filepath** arg with path to poke

Parametrize tasks – Jinja2 Template

```
from datetime import datetime
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.contrib.sensors.file_sensor import FileSensor

with DAG(
    dag_id="consume_new_data_from_pos_read_and_parse",
    start_date=datetime(2020, 12, 1),
    schedule_interval="0 * * * *"
) as dag:
    get_new_data = FileSensor(task_id="get_new_data",
                               filepath="../shop123/${current_date}/${hour}/data.json")
```

`${current_date} & ${hour} – we need somehow send dynamic variables inside it`

Parametrize tasks – Jinja2 Template

Use Jinja2 Templates!

<https://airflow.apache.org/docs/apache-airflow/stable/concepts.html#jinja-template>

Available macros: <https://airflow.apache.org/docs/apache-airflow/stable/macros-ref.html#macros-reference>

Parametrize tasks – Jinja2 Template

Airflow DAGs Data Profiling ▾ Browse ▾ Admin ▾ Docs ▾ About ▾ 2020-12-04 10:46:14 UTC

On DAG: file_sensor_consume_new_data schedule: 0 * * * * 2020-12-04 10:46:38 UTC

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Trigger DAG Refresh Delete Base date: 2020-12-01 01:00:00 Number of runs: 25 Go schedule: 0 * * * *

DummyOperator FileSensor details Code Trigger DAG Refresh Delete

[DAG] get_new_data parse_file check_is_it_new_customer update_existed_customer create_new_customer

Legend: scheduled skipped upstream_failed up_for_reschedule up_for_retry failed success running queued no_status

ew_data/2020-12-01T00:00:00+00:00/1.log
file_sensor_consume_new_data.get_new_data 2020-12-01T00:00:00+00:00/1.log
[2020-12-04 13:45:53,899] {taskinstance.py:670} INFO - Dependencies all met for <TaskInstance: file_sensor_consume_new_data.get_new_data 2020-12-01T00:00:00+00:00>
[2020-12-04 13:45:53,899] {taskinstance.py:880} INFO -

[2020-12-04 13:45:53,899] {taskinstance.py:881} INFO - Starting attempt 1 of 1
[2020-12-04 13:45:53,899] {taskinstance.py:882} INFO -

[2020-12-04 13:45:53,903] {taskinstance.py:901} INFO - Executing <Task(FileSensor): get_new_data> on 2020-12-01T00:00:00+00:00
[2020-12-04 13:45:53,905] {standard_task_runner.py:54} INFO - Started process 75473 to run task
[2020-12-04 13:45:53,933] {standard_task_runner.py:77} INFO - Running: ['airflow', 'run', 'file_sensor_consume_new_data', 'get_new_data', '2020-12-01T00:00:00+00:00']
[2020-12-04 13:45:53,935] {standard_task_runner.py:78} INFO - Job 5: Subtask get_new_data
[2020-12-04 13:45:53,961] {logging_mixin.py:112} INFO - Running %s on host %s <TaskInstance: file_sensor_consume_new_data.get_new_data 2020-12-01T00:00:00+00:00>
[2020-12-04 13:45:54,138] {file_sensor.py:60} INFO - Poking for file ../../shop123/20201201/{\$hour}/data.json

Toggle wrap Jump to end

Last questions of Day 1

1. UI view changes (Operators colorizing)
2. Statuses - what are they mean?
3. What to do with **hours**?

Statuses

■ scheduled ■ skipped ■ upstream_failed ■ up_for_reschedule ■ up_for_retry ■ failed ■ success ■ running ■ queued ■ no_status

Sensors status

Statuses

`mode``{ poke | reschedule }```



Task is running non-stop on worker

Task re-schedule
(free worker cpu, free pool to other tasks)

Let's try it & knows that to do with hours in the Day2



The end of the Day 1

Homework

- Check Apache Airflow docs web-site <https://airflow.apache.org/docs/stable/start.html#quick-start>
- Do apache airflow Quick install (see links on the slides) & copy lecture DAGs in DAG folder – unpause them and see different behaviour
- Run through Apache Aiflow source code & invistigare Dag, Task and Base Operator classes

Lecture DAGs:

https://github.com/xnuinside/airflow_lectures

Clone Airflow in Docker Compose with Celery:

https://github.com/xnuinside/airflow_in_docker_compose

Check that components are in docker-compose and needed to be defined to run with CeleryExecutor & try to up & run.