Index

- Workflows Management Systems
- Architecture
- Building blocks
- More features
- User Interface
- Security
- CLI
- Demo

WTH is a Workflow Management System ?

A worflow Management system is:

Is a data-centric software (framework) for :

- **Setting up**
- **Performing**
- **Monitoring**

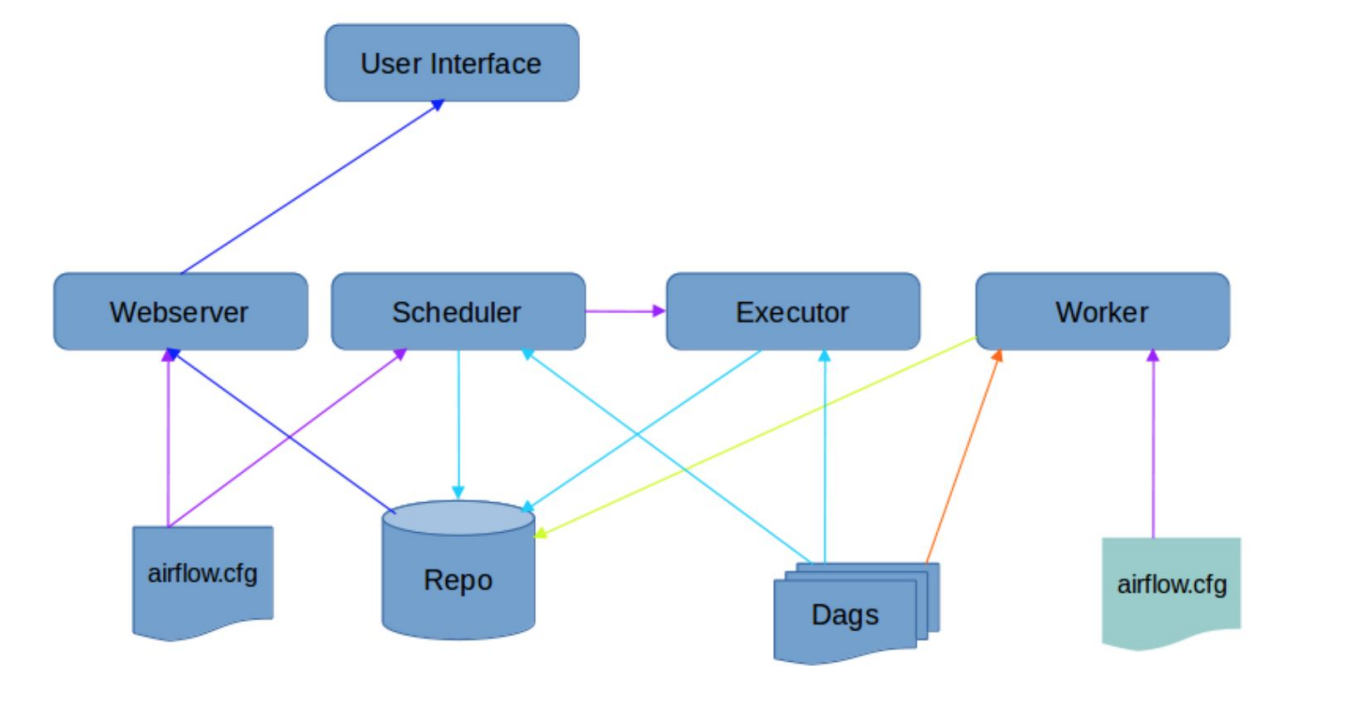
of a defined **sequence of processes** and **tasks**

Popular Workflow Management Systems

Airflow Architecture

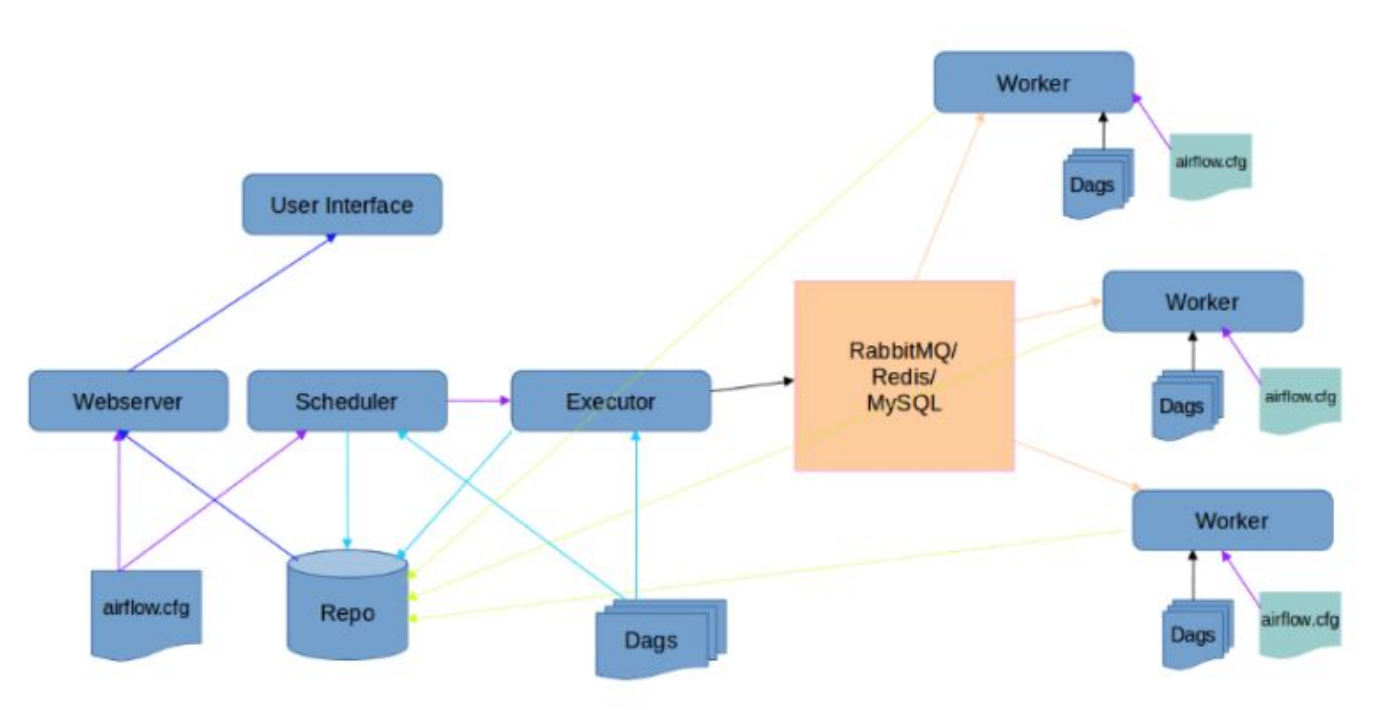
Airflow architecture

SequentialExecutor / LocalExecutor



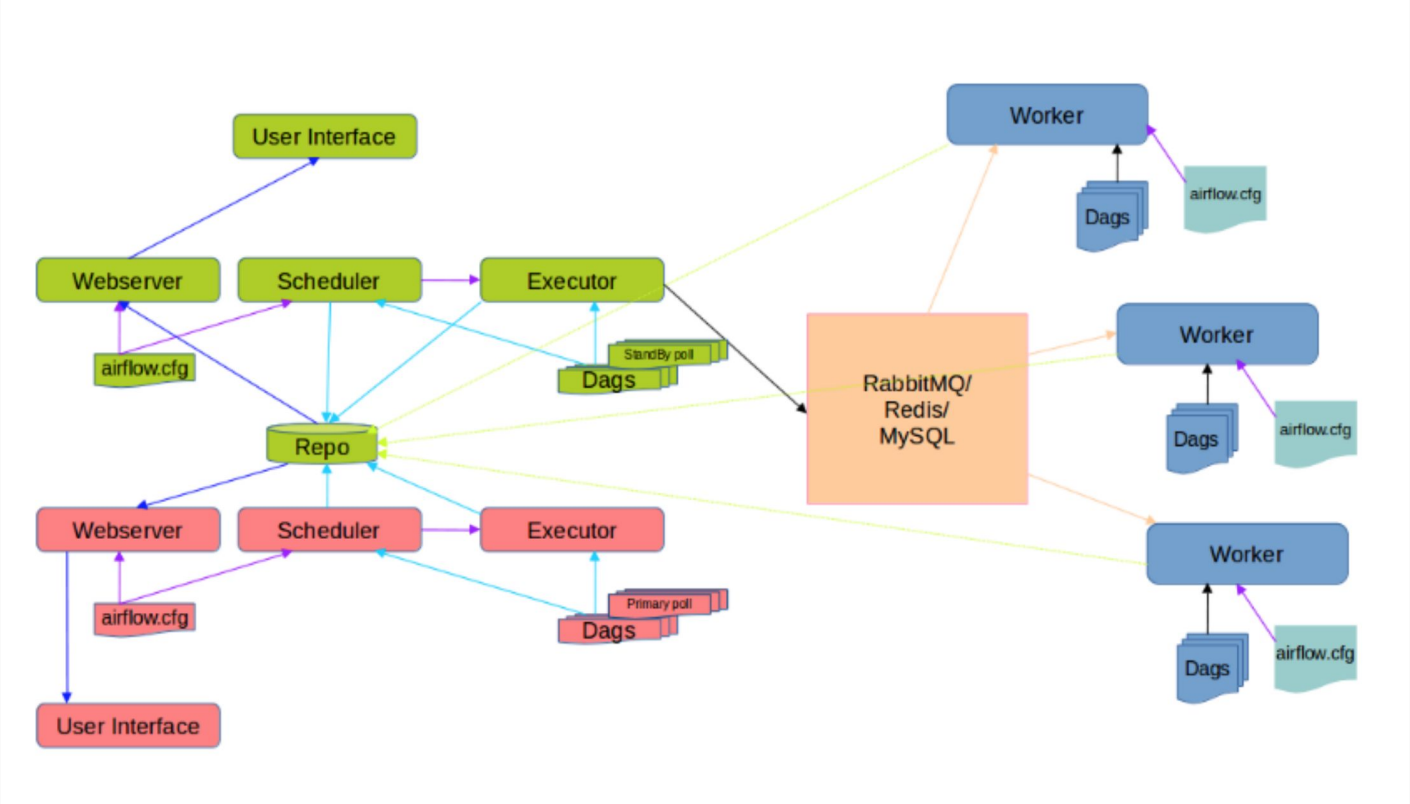
Airflow architecture

CeleryExecutor



Airflow architecture

HA + CeleryExector



Airflow architecture

- MesosExecutor : already available in contrib package
- KubernetesExecutor ??

Building blocks

Building blocks

Dags :

- Directed Acyclic Graph
- Is a collection of all the tasks you want to run
- DAGs describe *how* to run a workflow

Building blocks

Dags :

```
1   from airflow.models import DAG
2
3   default_args = {
4       'owner': 'airflow',
5       'start_date': datetime(2018, 3, 3),
6       'schedule_interval' : '0 12 * * *',
7       'depends_on_past': False,
8       'retry_delay': timedelta(minutes=5),
9       'retries': 1,
10      'email': ['                    '],
11      'email_on_failure': True,
12      'email_on_retry': True
13  }
14
15  facebook_insights_dag = DAG(
16      dag_id='facebook_insights_dag',
17      default_args=default_args
18  )
19
20
```

Building blocks

Operators :

- Describes a **single task** in a workflow.
- Determine what actually gets done
- Operators generally run independently (atomic)
- The DAG make sure that operators run in the correct certain **order**
- They may run on completely **different machines**

Building blocks

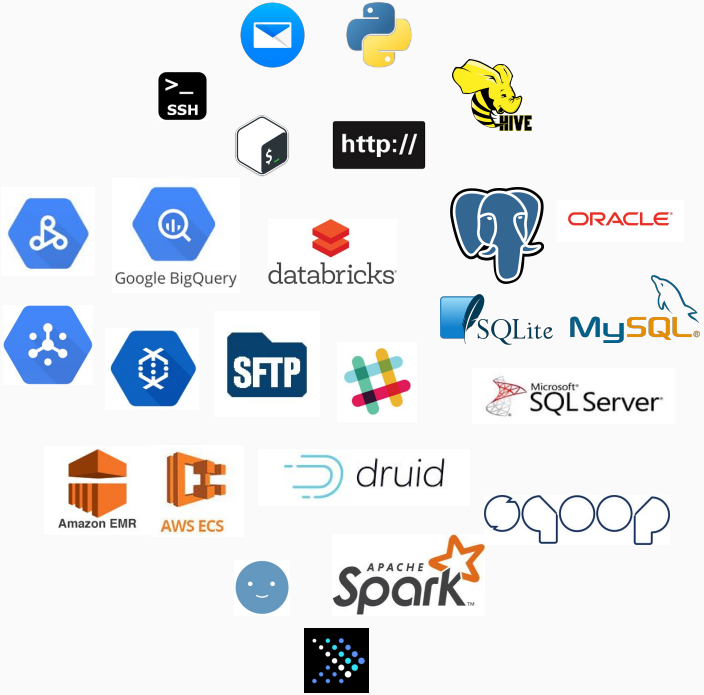
Operators：There are 3 main types of operators:

- **Operators** that performs an **action**, or tell another system to perform an action
- **Transfer** operators move data from one system to another
- **Sensors** are a certain type of operator that will **keep running** until a certain **criterion** is met.
  - Examples include a specific file landing in HDFS or S3.
  - A partition appearing in Hive.
  - A specific time of the day.

Building blocks

Operators :

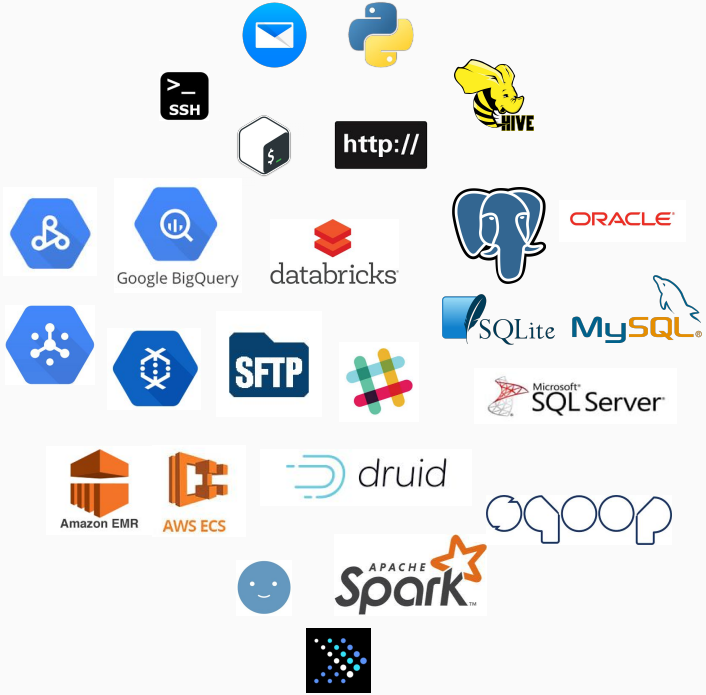
- Operators :
  - BashOperator
  - PythonOperator
  - EmailOperator
  - HTTPOperator
  - MySqlOperator
  - SqliteOperator
  - PostgresOperator
  - MsSqlOperator
  - OracleOperator
  - JdbcOperator
  - DockerOperator
  - HiveOperator
  - SlackOperator



Building blocks

Operators :

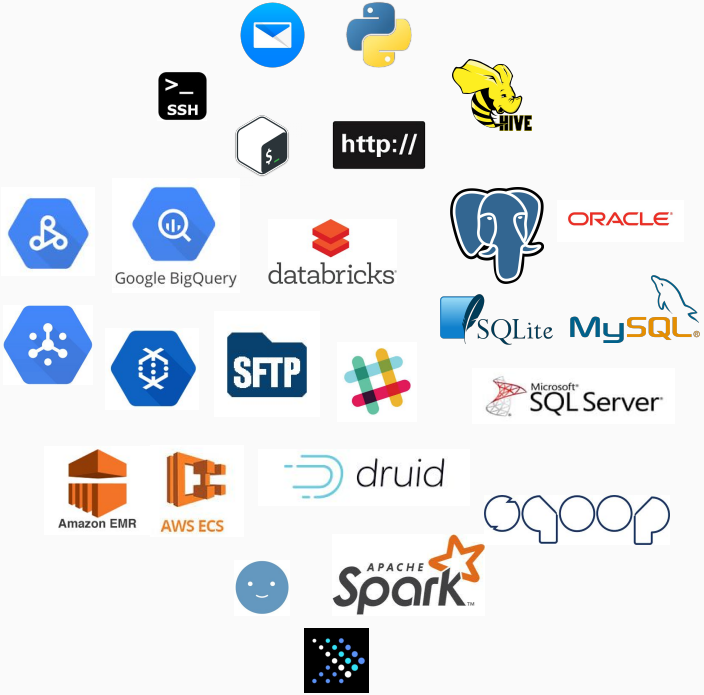
- Transfers :
  - S3FileTransferOperator
  - PrestoToMysqlOperator
  - MySqlToHiveTransfer
  - S3ToHiveTransfer
  - BigQueryToCloudStorageOperator
  - GenericTransfer
  - HiveToDruidTransfer
  - HiveToMySqlTransfer



Building blocks

Operators :

- Sensors :
  - ExternalTaskSensor
  - HdfsSensor
  - HttpSensor
  - MetastorePartitionSensor
  - HivePartitionSensor
  - S3KeySensor
  - S3PrefixSensor
  - SqlSensor
  - TimeDeltaSensor
  - TimeSensor
  - WebHdfsSensor



Building blocks

Operators :

```python
def fetch_ads_infos(**kwargs):
    ...
def enrich_campaigns_insights(**kwargs):
    ...

ads_infos_task = PythonOperator(
    task_id='ads_infos_task',
    python_callable=fetch_ads_infos,
    provide_context=True,
    dag=facebook_insights_dag
)

enrich_campaigns_lifetime_insights_task = PythonOperator(
    task_id='enrich_campaigns_lifetime_insights_task',
    op_kwargs={
        'granularity': 'lifetime'
    },
    python_callable=enrich_campaigns_insights,
    provide_context=True,
    dag=facebook_insights_dag
)

ads_infos_task >> enrich_campaigns_lifetime_insights_task
ads_infos_task.set_downstream(enrich_campaigns_lifetime_insights_task)

enrich_campaigns_lifetime_insights_task << ads_infos_task
enrich_campaigns_lifetime_insights_task.set_upstream(ads_infos_task)
```

Building blocks

Tasks : a parameterized instance of an operator

Building blocks

Task Instance : Dag + Task + point in time

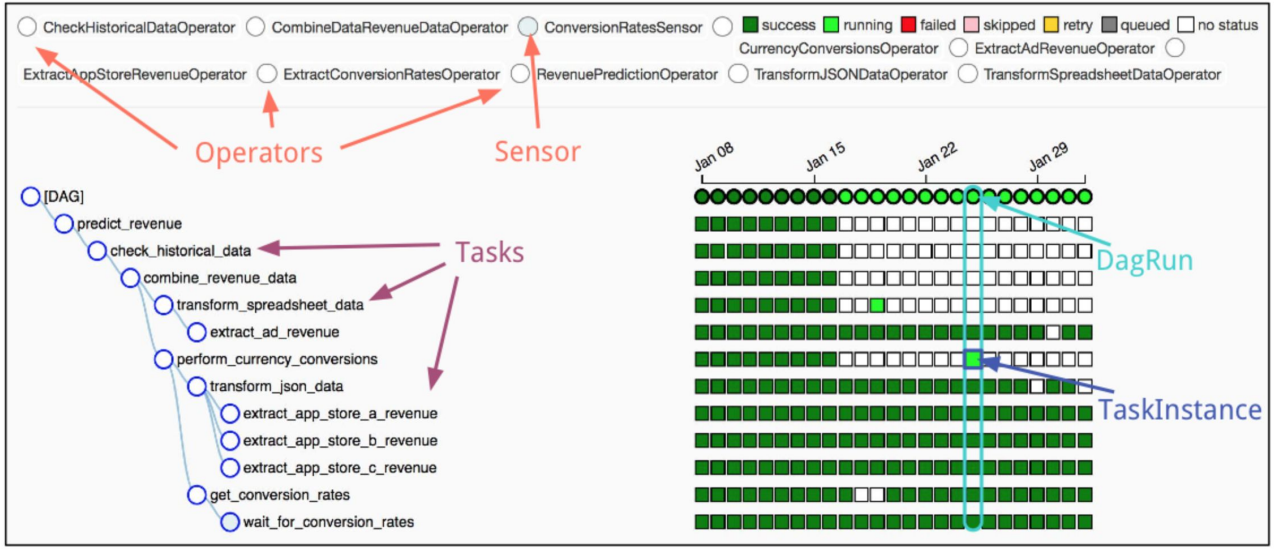
- Specific run of a Task
- A task assigned to a DAG
- Has State associated with a specific run of the DAG
- States : it could be
    - **running**
    - **success**,
    - **failed**
    - **skipped**
    - **up for retry**
    - **...**

Building blocks

Workflows :

- **DAG**: a description of the order in which work should take place
- **Operator**: a class that acts as a **template** for carrying out some work
- **Task**: a **parameterized instance** of an operator
- **Task** Instance: a task that
  - Has been assigned to a DAG
  - Has a state associated with a specific run of the DAG
- By combining **DAGs** and Operators to create **TaskInstances**, you can build **complex workflows**.

Building blocks



More features

More features

- Features :
  - Hooks
  - Connections
  - Variables
  - XComs
  - SLA
  - Pools
  - Queues
  - Trigger Rules
  - ~~Branchings~~
  - ~~SubDags~~

More features

Hooks :

- Interface to external platforms and databases :
  - Hive
  - S3
  - MySQL
  - PostgreSQL
  - HDFS
  - Hive
  - Pig
  - …
- Act as building block for Operators
- Use **Connection** to retrieve authentication informations
- Keep authentication infos out of pipelines.

More features

Connections :

Connection informations to external systems are stored in the airflow metadata Database and managed in the UI

More features

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 15:58 UTC

- Pools
- Configuration
- Users
- Connections
- Variables
- XComs

	druid_ingest_default		druid-overlord	8081		
	emr_default					
	fs_default					
	google_cloud_default	_platform				
	hive_cli_default	hive_cli				
	hiveserver2_default	hiveserver2	localhost	10000		
	http_default	http	https://www.google.com/			
	local_mysql	mysql	localhost			
	metastore_default	hive_metastore	localhost	9083		
	mssql_default	mssql	localhost	1433		
	mysql_default	mysql	localhost			
	postgres_default	postgres	localhost			
	presto_default	presto	localhost	3400		
	redis_default	redis	localhost	6379		
	s3_pr_connection	s3				
	spark_default	spark	yarn			
	sqlite_default	sqlite	/tmp/sqlite_default.db			
	sqoop_default	sqoop	rmdbs			
	ssh_default	ssh	localhost			
	vertica_default	vertica	localhost	5433		
	wasb_default	wasb				
	webhdfs_default	hdfs	localhost	50070		

More features

Exemple de Hook + connection :

```python

def enrich_campaigns_insights(granularity, **kwargs):
    s3hook = S3Hook(aws_conn_id=AWS_AF_CONNECTION_NAME)
    extraction_date = kwargs['execution_date'].strftime('%Y-%m-%d')

    campaigns_infos_path = 'facebook-ads-and-campaigns/referential/campaigns_infos.json'
    campaigns_insights_path = f'facebook-ads-and-campaigns/data/campaigns_{granularity}_{extraction_date}.json'
    output_path = f'facebook-ads-and-campaigns/data/enriched_campaigns_{granularity}_{extraction_date}.json'

    campaigns_infos_select_cols = ['account_id','id','start_time','stop_time','updated_time','status','created_time','effective_status']

    campaigns_infos_content = s3hook.read_key(campaigns_infos_path, S3_BUCKET)
    campaigns_insights_content = s3hook.read_key(campaigns_insights_path, S3_BUCKET)
    campaigns_referential_df = pd.read_json(campaigns_infos_content, lines=True)
    campaigns_insights_df = pd.read_json(campaigns_insights_content, lines=True)

    enriched_campaigns_df = campaigns_insights_df.merge(campaigns_referential_df[campaigns_infos_select_cols],
        how='inner',
        left_on=['account_id','campaign_id'],
        right_on=['account_id','id'])

    jl_data = enriched_campaigns_df.to_json(orient='records', lines=True)
    s3hook.load_string(
        jl_data,
        output_path,
        S3_BUCKET,
        replace=True,
        encrypt=False)
```

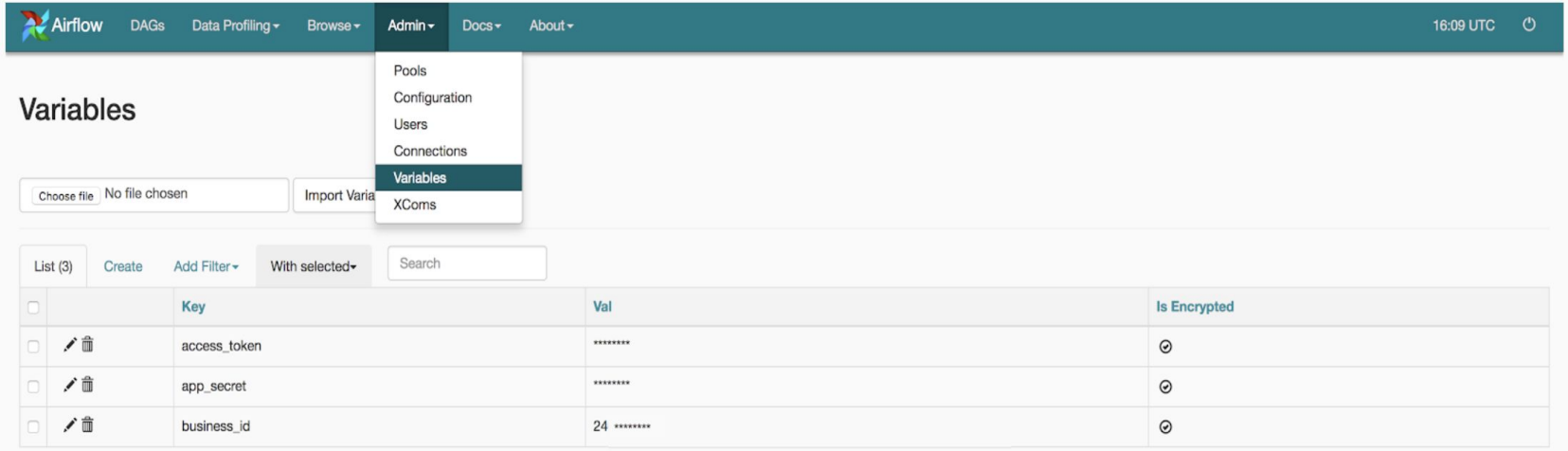
More features

Variables :

- A generic way to store and retrieve arbitrary content or settings as a simple key value store within Airflow.
- Variables can be listed, created, updated and deleted from the UI (Admin -> Variables), code or CLI.
- While your pipeline code definition and most of your constants and variables should be defined in code and stored in source control, it can be useful to have some variables or configuration items accessible and modifiable through the UI.

```
1  from airflow.models import Variable
2  foo = Variable.get("foo")
3  bar = Variable.get("bar", deserialize_json=True)
```

More features



More features

XCom or Cross-communication:

- Let tasks **exchange** messages allowing **shared state**.
- Defined by a **key**, **value**, and **timestamp**.
- Also **track** attributes like the task/DAG that created the XCom and when it should become visible.
- **Any object** that can be **pickled** can be used as an XCom value.

XComs can be :

- Pushed (sent) :
  - Calling `xcom_push()`
  - If a task return a value (from its operator `execute()` method) or from a PythonOperator's `python_callable`
- Pulled (received) : calling `xcom_pull()`

More features

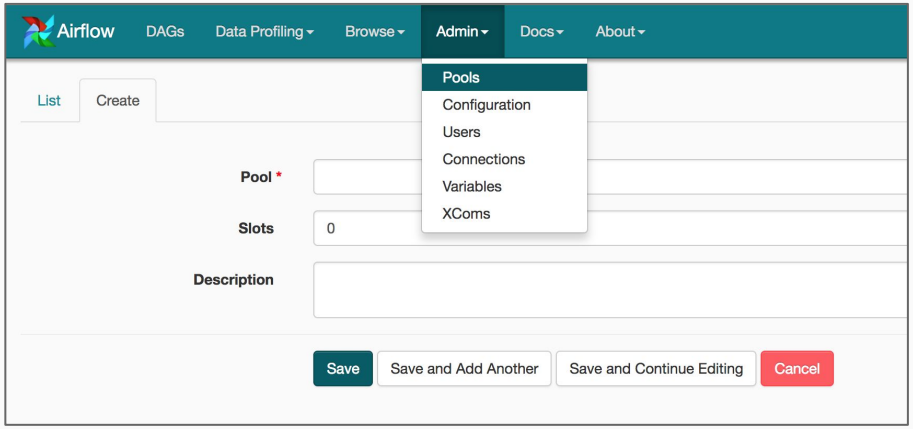
SLA :

- Service Level Agreements, or time by which a task or DAG should have succeeded,
- Can be set at a task level as a timedelta.
- An alert email is sent detailing the list of tasks that missed their SLA.

More features

Pools :

- Some systems can get overwhelmed when too many processes hit them at the same time.
- **Limit the execution parallelism** on arbitrary sets of tasks.



More features

Pools :

```
1  aggregate_db_message_job = BashOperator(
2      task_id='aggregate_db_message_job',
3      execution_timeout=timedelta(hours=3),
4      pool='ep_data_pipeline_db_msg_agg',
5      bash_command=aggregate_db_message_job_cmd,
6      dag=dag)
7  aggregate_db_message_job.set_upstream(wait_for_empty_queue)
```

More features

Queues : (only on CeleryExecutors) :

- Every Task can be assigned a specific queue name
- By default, both worker and tasks are assigned with the **default_queue** queue
- Workers can be assigned **multiple queues**
- Very useful feature when specialized workers are needed (GPU, Spark…)

More features

Trigger Rules:

Though the normal workflow behavior is to trigger tasks when all their directly upstream tasks have succeeded, Airflow allows for more complex dependency settings.

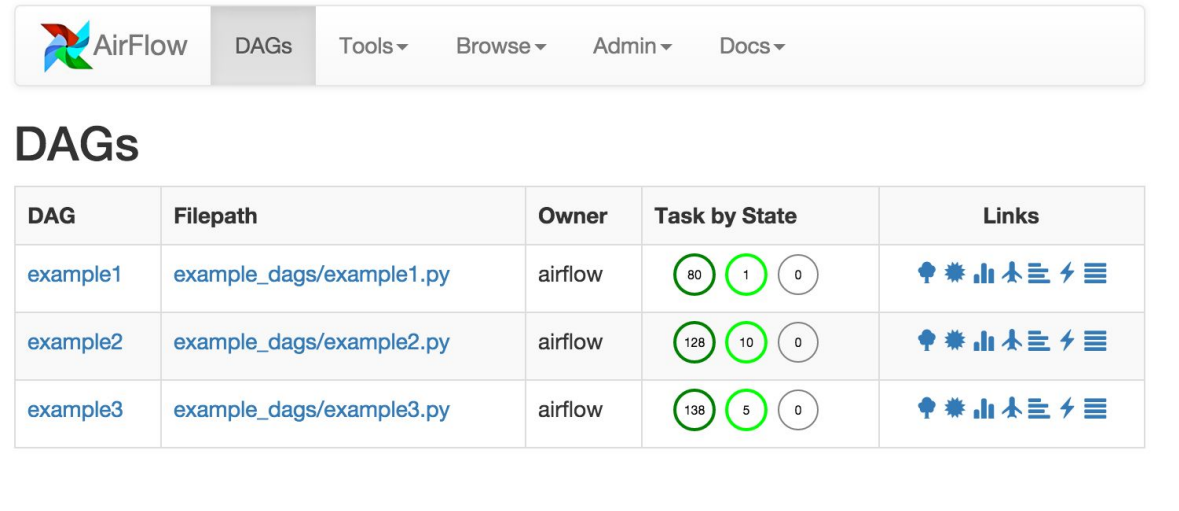
All operators have a trigger_rule argument which defines the rule by which the generated task get triggered. The default value for trigger_rule is all_success and can be defined as “trigger this task when all directly upstream tasks have succeeded”. All other rules described here are based on direct parent tasks and are values that can be passed to any operator while creating tasks:

- **all_success**: (default) all parents have succeeded
- **all_failed**: all parents are in a failed or upstream_failed state
- **all_done**: all parents are done with their execution
- **one_failed**: fires as soon as at least one parent has failed, it does not wait for all parents to be done • one_success: fires as soon as at least one parent succeeds, it does not wait for all parents to be done • dummy: dependencies are just for show, trigger at will.

User Interface

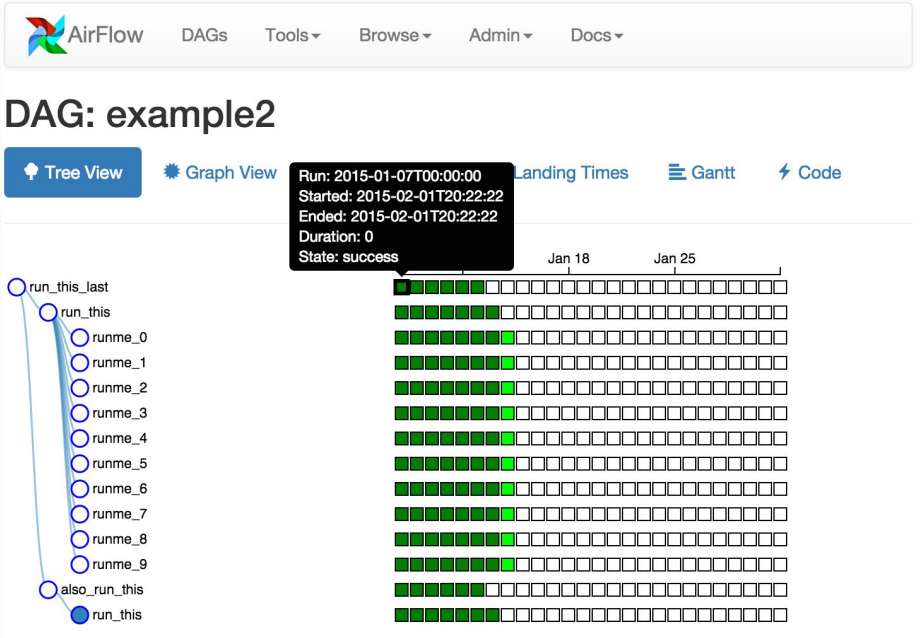
User Interface

Dags view :



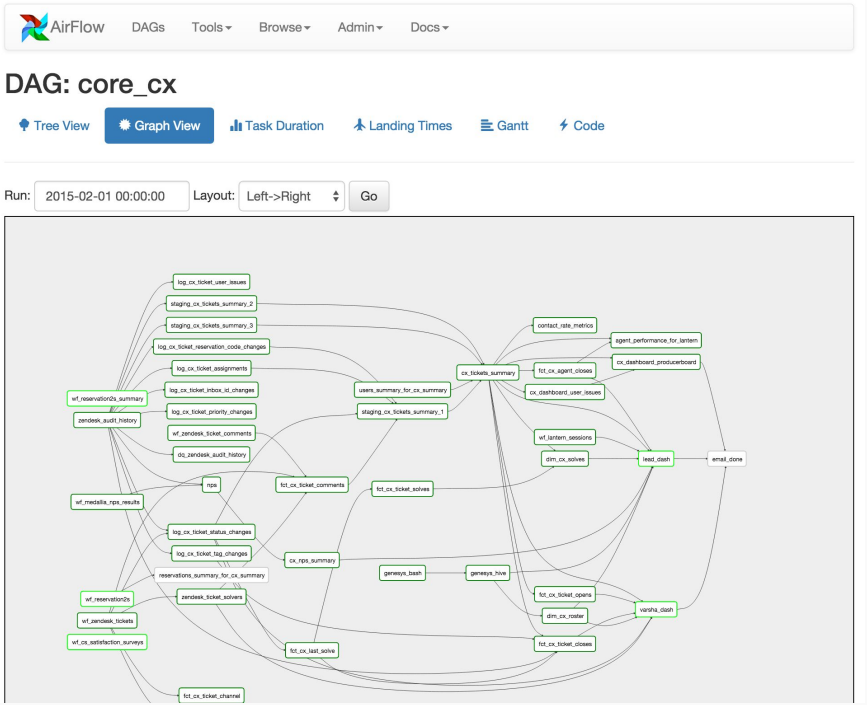
User Interface

Tree view :



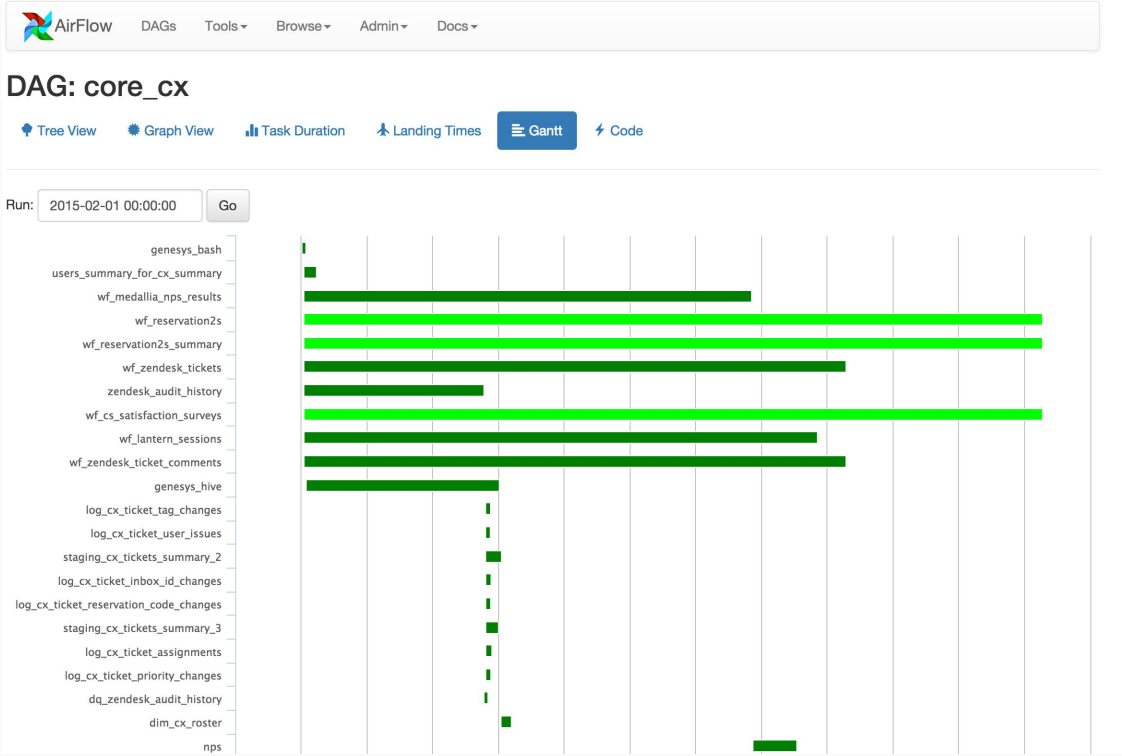
User Interface

Graph view :



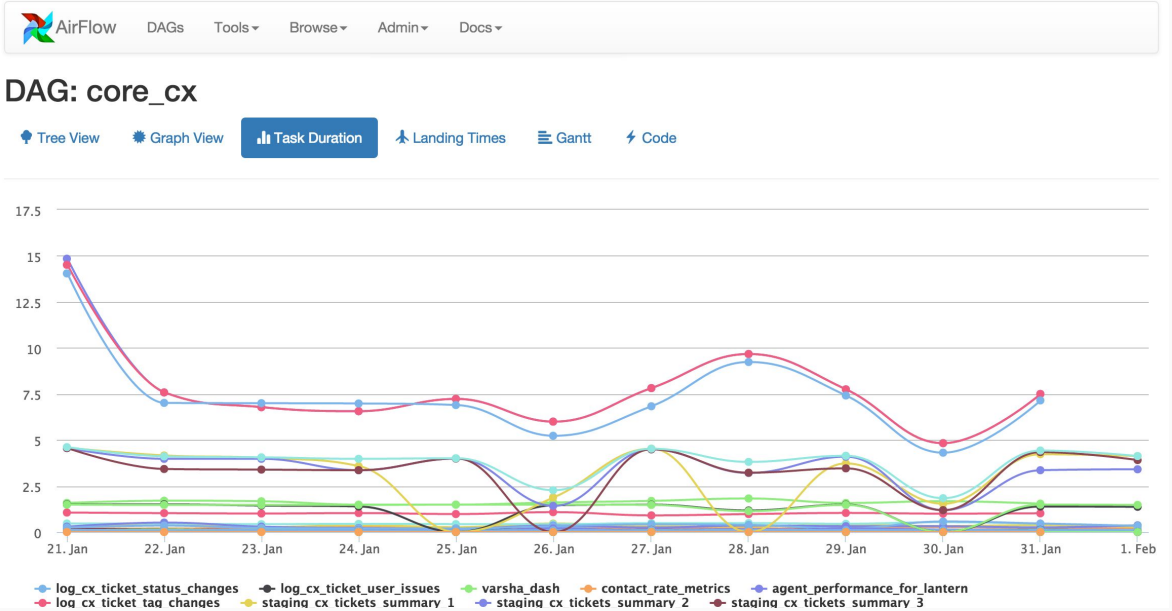
User Interface

Gantt view :



User Interface

Task duration :



User Interface

Data Profiling : SQL Queries

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 05:04 UTC

Ad Hoc Query

airflow_db | Run! | .csv

```
select * from task_instance limit 10;
```

Show 100 entries

Search:

task_id	dag_id	execution_date	start_date	end_date	duration	state	try_number	hostname	unixname	job_id	pool	queue	priority_weight	operator
convert_to_csv_task	facebook_insights_dag	2018-03-20	2018-03-21 16:38:52.541027	2018-03-21 16:39:09.613790	17.072763	success	1	OKACHAs-MacBook-Pro.local	IOKACHA	452		default	1	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-17	2018-03-21 16:22:00.403601	2018-03-21 16:26:53.214537	292.810936	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	357		default	3	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-16	2018-03-21 16:22:00.964350	2018-03-21 16:27:11.234319	310.269969	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	359		default	3	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-19	2018-03-21 16:22:00.702286	2018-03-21 16:27:11.709757	311.007471	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	358		default	3	PythonOperator
campaigns_infos_task	facebook_insights_dag	2018-03-20	2018-03-21 16:20:51.355984	2018-03-21 16:21:31.989539	40.633555	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	330		default	4	PythonOperator
campaigns_infos_task	facebook_insights_dag	2018-03-18	2018-03-21 16:20:51.400999	2018-03-21 16:21:34.013295	42.612296	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	333		default	4	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-20	2018-03-21 16:22:01.067421	2018-03-21 16:27:13.030706	311.963285	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	360		default	3	PythonOperator
enrich_ads_daily_insights_task	facebook_insights_dag	2018-03-15	2018-03-21 16:22:53.110321	2018-03-21 16:23:13.671750	20.561429	success	1	OKACHAs-MacBook-Pro.local	IOKACHA	382		default	2	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-14	2018-03-21 16:23:25.384377	2018-03-21 16:28:03.737195	278.352818	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	401		default	3	PythonOperator
campaign_lifetime_insights_task	facebook_insights_dag	2018-03-15	2018-03-21 16:23:52.229709	2018-03-21 16:28:07.791497	255.561788	success	5	OKACHAs-MacBook-Pro.local	IOKACHA	419		default	3	PythonOperator

Showing 1 to 10 of 10 entries

Previous | 1 | Next

User Interface

Data Profiling : Charts

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 05:09 UTC

List | Create | Edit

Label
label
Can include {{ templated_fields }} and {{ macros }}

Owner
The chart's owner, mostly used for reference and filtering in the list view.

Conn Id *
airflow_db
Source database to run the query against

Chart Type
Bar Chart
The type of chart to be displayed

Show Datatable
Whether to display an interactive data table under the chart.

X Is Date
Whether the X axis should be casted as a date field. Expect most intelligible date formats to get casted properly.

Y Log Scale
Whether to use a log scale for the Y axis.

Display the SQL Statement
Whether to display the SQL statement as a collapsible section in the chart page.

Chart Height
600
Height of the chart, in pixels.

SQL Layout
SELECT series, x, y FROM ...
Defines the layout of the SQL that the application should expect. Depending on the tables you are sourcing from, it may make more sense to pivot / unpivot the metrics.

SQL
Can include {{ templated_fields }} and {{ macros }}.

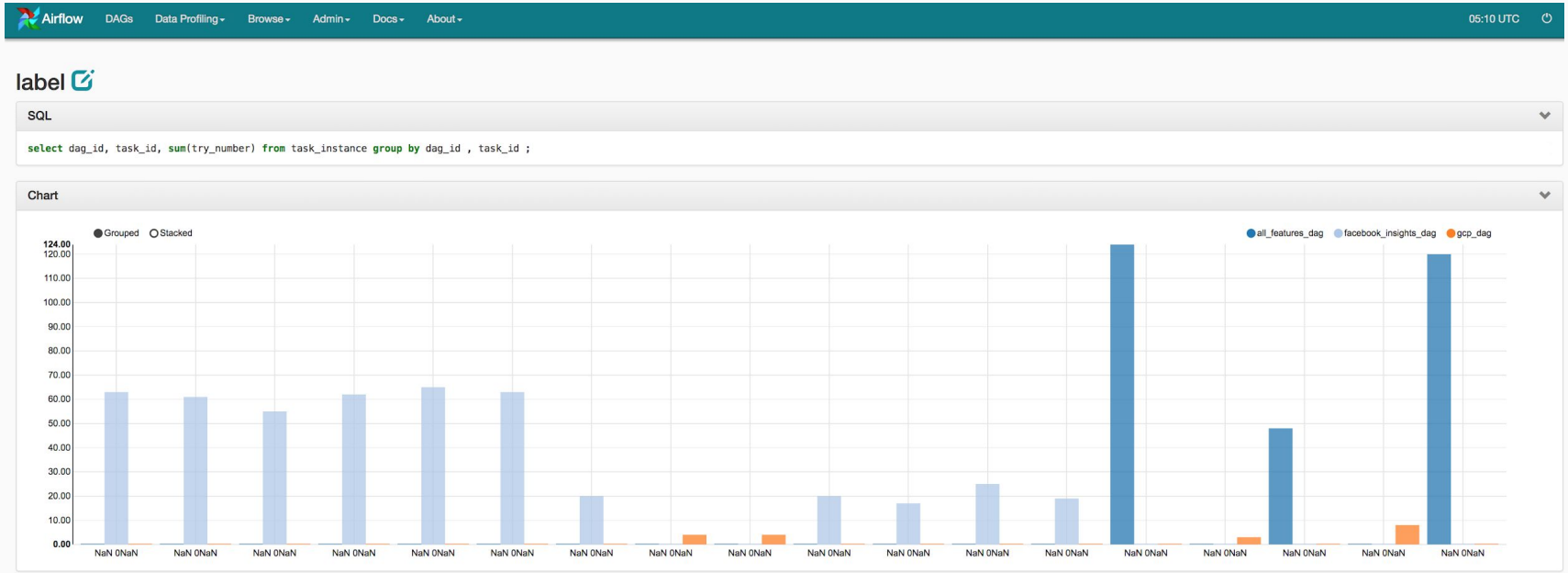
```
1 select dag_id, task_id, sum(try_number) from task_instance group by dag_id , task_id ;
```

Default Parameters
{}
A dictionary of {"key": "values",} that define what the templated fields (parameters) values should be by default. To be valid, it needs to "eval" as a Python dict. The key values will show up in the url's querystring and can be altered there.

Save | Save and Add Another | Save and Continue Editing | Cancel

User Interface

Data Profiling : Charts



CLI

CLI

```
airflow variables [-h] [-s KEY VAL] [-g KEY] [-j] [-d VAL] [-i FILEPATH] [-e FILEPATH] [-x KEY]

airflow connections [-h] [-l] [-a] [-d] [--conn_id CONN_ID]
                    [--conn_uri CONN_URI] [--conn_extra CONN_EXTRA]
                    [--conn_type CONN_TYPE] [--conn_host CONN_HOST]
                    [--conn_login CONN_LOGIN] [--conn_password CONN_PASSWORD]
                    [--conn_schema CONN_SCHEMA] [--conn_port CONN_PORT]


airflow pause [-h] [-sd SUBDIR] dag_id


airflow test [-h] [-sd SUBDIR] [-dr] [-tp TASK_PARAMS] dag_id task_id execution_date

airflow backfill dag_id task_id -s START_DATE -e END_DATE

airflow clear DAG_ID

airflow resetdb [-h] [-y]
```

https://airflow.apache.org/cli.html

...

Security

Security

By default : all access are open

Support ;

- Web authentication with :
  - Password
  - LDAP
  - Custom auth
  - Kerberos
  - OAuth
    - Github Entreprise Authentication
    - Google Authentication

- Impersonation (run as other $USER)
- Secure access via SSL

Demo

Demo

1. **Facebook Ads insights** data pipeline.
2. Run a pyspark script on a ephemeral **dataproc** cluster only when s3 data input is available
3. Useless workflow : Hook + Connection + Operators + Sensors + XCom +(SLA):
    - List s3 files (hooks)
    - Share state with the next task (xcom)
    - Write content to s3 (hooks)
    - Resume the workflow when an S3 DONE.FLAG file is ready (sensor)

Resources

https://airflow.apache.org

http://www.clairvoyantsoft.com/assets/whitepapers/GuideToApacheAirflow.pdf

https://speakerdeck.com/artwr/apache-airflow-at-airbnb-introduction-and-lessons-learned

https://www.slideshare.net/sumitmaheshwari007/apache-airflow

Thanks