

DAG File explained

A DAG file is where you define the tasks and dependencies using Python. This is also where you

Where is it stored?

In order for the airflow `Scheduler` to find your DAG and for it to be displayed in your `WebServer` UI, it needs to be stored in the DAG directory

When using `astro cli` project locally, you can put them in your `dags` folder.

When using AWS Managed Workflow for Apache Airflow, you can put it in a synced S3 bucket.

When using your own deployment, you have to check the specified DAGS folder in the airflow configuration.

Unfortunately, there's no easy way to just upload a DAG file using the Web UI.

Defining a DAG

```
from airflow import DAG
from datetime.datetime

dag = DAG(
    dag_id="hello_world_dag",
    start_date=datetime
    schedule_interval=None
)
```

Required parameters

1. `dag_id` is a required parameter that
2. `start_date` must be a python `datetime.datetime` object check here
3. `schedule_interval` defines how often your DAG would be run. This usually takes a cron syntax (crontab.guru) or a `datetime.timedelta` object specifying a period of time.

Defining a task

Let's look at the most basic dummy operator that does nothing.

```
from airflow import DAG
import datetime

dag = DAG(
    dag_id="hello_world_dag",
    start_date=datetime.datetime(2021, 1, 1)
    schedule_interval=None
)
```

```
)  
  
dummy = DummyOperator(task_id='task1', dag=dag)
```

Note how we specify which DAG a task belongs to with the parameter `dag=dag`

Other ways of defining DAG with a task

Aside from the syntax above, you may see DAG and tasks defined like below which makes it easier to not define `dag=dag` over and over

```
import datetime  
from airflow import DAG  
from airflow.operators.dummy import DummyOperator  
  
with DAG(  
    dag_id="hello_world_dag",  
    start_date=datetime.datetime(2021, 1, 1)  
    schedule_interval=None  
) as DAG:  
  
    dummy1 = DummyOperator(task_id='task1')  
    dummy2 = DummyOperator(task_id='task2')  
    dummy3 = DummyOperator(task_id='task3')
```

Notice how we don't have to specify the dag for each task anymore.

Specifying task dependencies

Most common syntax for defining dependency will be the following:

```
task1 >> task2
```

Where `task2` depends on `task1`.

To specify a task depending on multiple tasks you can do the following:

```
[task1, task2] >> task3
```


Waiting for DAG refresh

Airflow scheduler usually takes 30 seconds to check new DAGs or check if your DAG's syntax is valid. Check Web UI if your DAG is refreshed.


Checking DAG in Web UI

DAG list

This is the home page of Airflow UI. Check here if your new DAG is finally registered.

 Airflow

[DAGs](#) [Security](#) [Browse](#) [Admin](#) [Docs](#)


























































21:11 UTC 

DAGs

All 26 Active 10 Paused 16

Filter DAGs by tag

Search DAGs

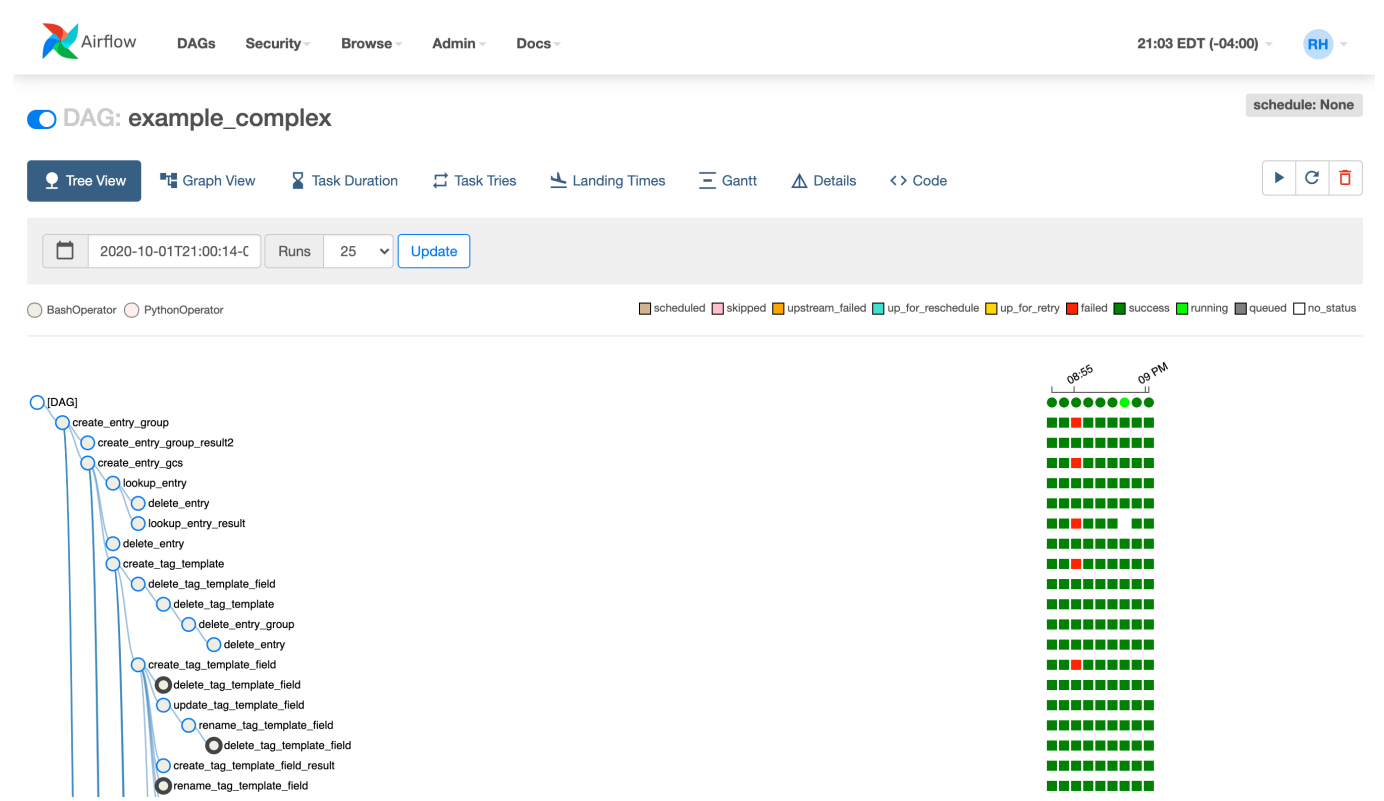
DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
 example_bash_operator example example2	airflow	 2	0 0 * * *	2020-10-26, 21:08:11	 8	  	...
 example_branch_dop_operator_v3 example	airflow		* / 1 * * * *			  	...
 example_branch_operator example example2	airflow	 1	@daily	2020-10-23, 14:09:17	 11	  	...
 example_complex example example2 example3	airflow	 1	None	2020-10-26, 21:08:04	 37	  	...
 example_external_task_marker_child	airflow	 1	None	2020-10-26, 21:07:33	 2	  	...
 example_external_task_marker_parent	airflow	 1	None	2020-10-26, 21:08:34	 1	  	...
 example_kubernetes_executor example example2	airflow		None			  	...
 example_kubernetes_executor_config example3	airflow	 1	None	2020-10-26, 21:07:40	 5	  	...
 example_nested_branch_dag example	airflow	 1	@daily	2020-10-26, 21:07:37	 9	  	...
 example_passing_params_via_test_command example	airflow		* / 1 * * * *			  	...

Check DAG in detail

After clicking a DAG in the DAG list, you can check the following:

Check Tree View

This shows a tree view as well as the status of your DAG runs across time.



Each vertical column in the graph on the right represents an execution of your DAG at a point in time.



The colors of the individual tasks determine their state (running, success etc.)

Check Graph View

Graph view is how you visualize your dag dependencies.

Airflow DAGs Security Browse Admin Docs 21:06 EDT (-04:00) RH

DAG: example_bash_operator schedule: 0 0 ***

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

2020-09-30T20:00:01-04:00 Runs 25 Run scheduled__2020-10-01T00:00:00+00:00 Layout Left > Right Update Find Task...

BashOperator DummyOperator

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

```

graph LR
    runme_0 --> also_run_this
    runme_1 --> also_run_this
    runme_2 --> run_after_loop
    also_run_this --> run_this_last
    run_after_loop --> run_this_last
  
```

Check code

Go to the DAG code tab to check if your DAG code has been updated.

Airflow DAGs Security Browse Admin Docs 21:30 EDT (-04:00) RH

DAG: example_bash_operator schedule: 0 0 ***

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

Toggle Wrap

```

1 #
2 # Licensed to the Apache Software Foundation (ASF) under one
3 # or more contributor license agreements. See the NOTICE file
4 # distributed with this work for additional information
5 # regarding copyright ownership. The ASF licenses this file
6 # to you under the Apache License, Version 2.0 (the
7 # "License"); you may not use this file except in compliance
8 # with the License. You may obtain a copy of the License at
9 #
10 # http://www.apache.org/licenses/LICENSE-2.0
11 #
12 # Unless required by applicable law or agreed to in writing,
13 # software distributed under the License is distributed on an
14 # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
15 # KIND, either express or implied. See the License for the
16 # specific language governing permissions and limitations
17 # under the License.
18
19 """Example DAG demonstrating the usage of the BashOperator."""
20
21 from datetime import timedelta
22
23 from airflow import DAG
24 from airflow.operators.bash import BashOperator
25 from airflow.operators.dummy_operator import DummyOperator
26 from airflow.utils.dates import days_ago
27
28 args = {
29     'owner': 'airflow',
30 }
31
32 dag = DAG(
33     dag_id='example_bash_operator',
34     default_args=args,
35     schedule_interval='0 0 * * *'.

```

Manually triggering DAG

Exercise:

Create your own DAG from scratch that 5 tasks or more.

- Create tasks with one or more dependencies.
- Create tasks that can run in parallel with other tasks.
- Name your dag "dag-exercise-1", both `dag_id` and filename.