CWL-Airflow

Michael Kotliar, Andrey Kartashov, Artem Barski

Nov 18, 2021

CONTENTS:

Python package to extend Apache-Airflow 2.1.4 functionality with CWL v1.1 support.

CHAPTER

ONE

CITE AS

Michael Kotliar, Andrey V Kartashov, Artem Barski, CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language, GigaScience, Volume 8, Issue 7, July 2019, giz084, https://doi.org/10.1093/gigascience/giz084

CHAPTER

TWO

NOTE

Current documentaion is still in progress. If you one of those who has just noticed typo in the word **documentaion**, we need your Pull Requests

2.1 Quick start

We assume that you have already installed **python 3.8**, latest **pip**, latest **setuptools** and **docker** that has access to pull images from the DockerHub. If something is missing or should be updated refer to the How to install or What if is doesn't work sections.

1. Install CWL-airflow

```
$ pip3 install cwl-airflow \
--constraint "https://raw.githubusercontent.com/Barski-lab/cwl-airflow/master/
↪packaging/constraints/constraints-3.8.txt"
```

   When using optional `--constraint` parameter you can limit dependencies to those versions that were tested with your Python.

2. Configure CWL-airflow (for details refer to Initial configuration section)

```
$ cwl-airflow init
```

3. Get some workflows to run, for example from SciDAP

```
$ git clone https://github.com/datirium/workflows.git --recursive
```

4. To be able to use Airflow Webserver, create a new user following the example below

```
airflow users create \
--username admin \
--firstname firstname \
--lastname lastname \
--role Admin \
--email firstname@lastname.org
```

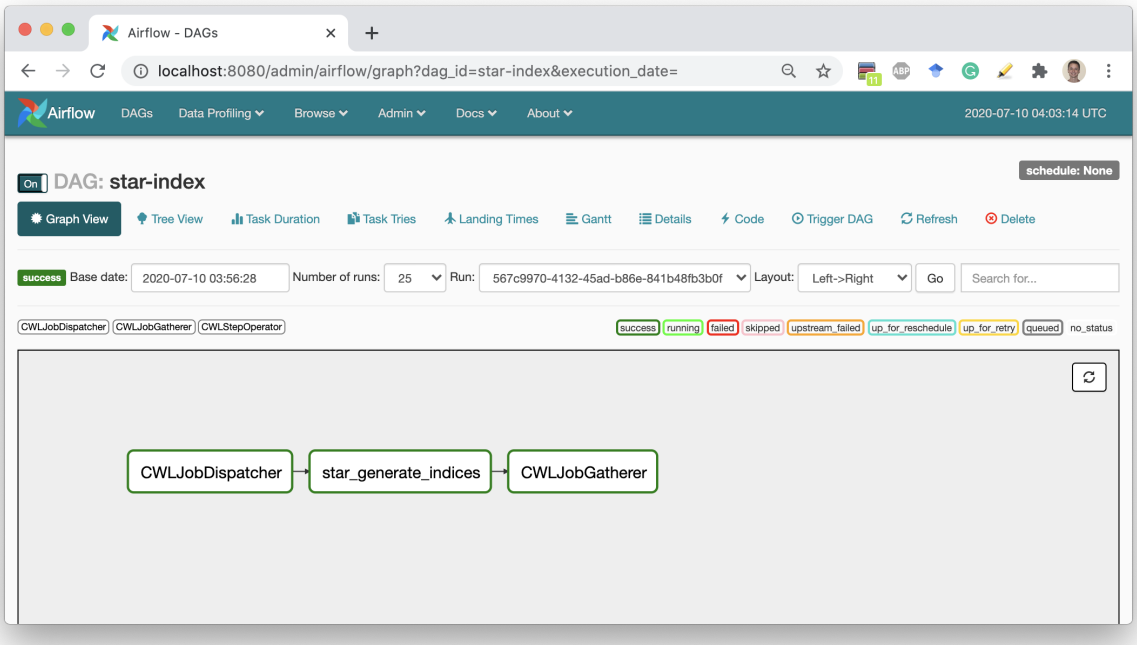
5. In a separate terminals start Airflow Webserver, Scheduler and our API

```
$ airflow scheduler
$ airflow webserver
$ cwl-airflow api
```

6. Schedule execution of a sample pipeline. Set the workflow number with `--range`

```
$ cwl-airflow test --suite workflows/tests/conformance_tests.yaml --range 1
```

7. Open Airflow Webserver (by default http://127.0.0.1:8080/admin/) and wait until Airflow Scheduler pick up a new DAG (by default every 5 min) and execute it. **On completion all results and temporary files will be removed**, so you can safely schedule other workflows by setting different values to `--range` parameter. Take a look at the How to use section for more details.



2.2 How it works

2.2.1 Keywords

1. **CWL descriptor file (aka pipeline or workflow)** - YAML or JSON file or its parsed content that complies with CWL v1.1 specification and describes inputs, outputs and sequence of steps to be executed.
2. **Job file (aka job or running configuration)** - YAML or JSON file or its parsed content that is used for initializing workflow inputs with values. Job can optionally include 2 additional fields:
   - **tmp_folder** - folder to keep temporary data that will be removed after successful workflow execution
   - **outputs_folder** - folder to move generated results after successful workflow execution

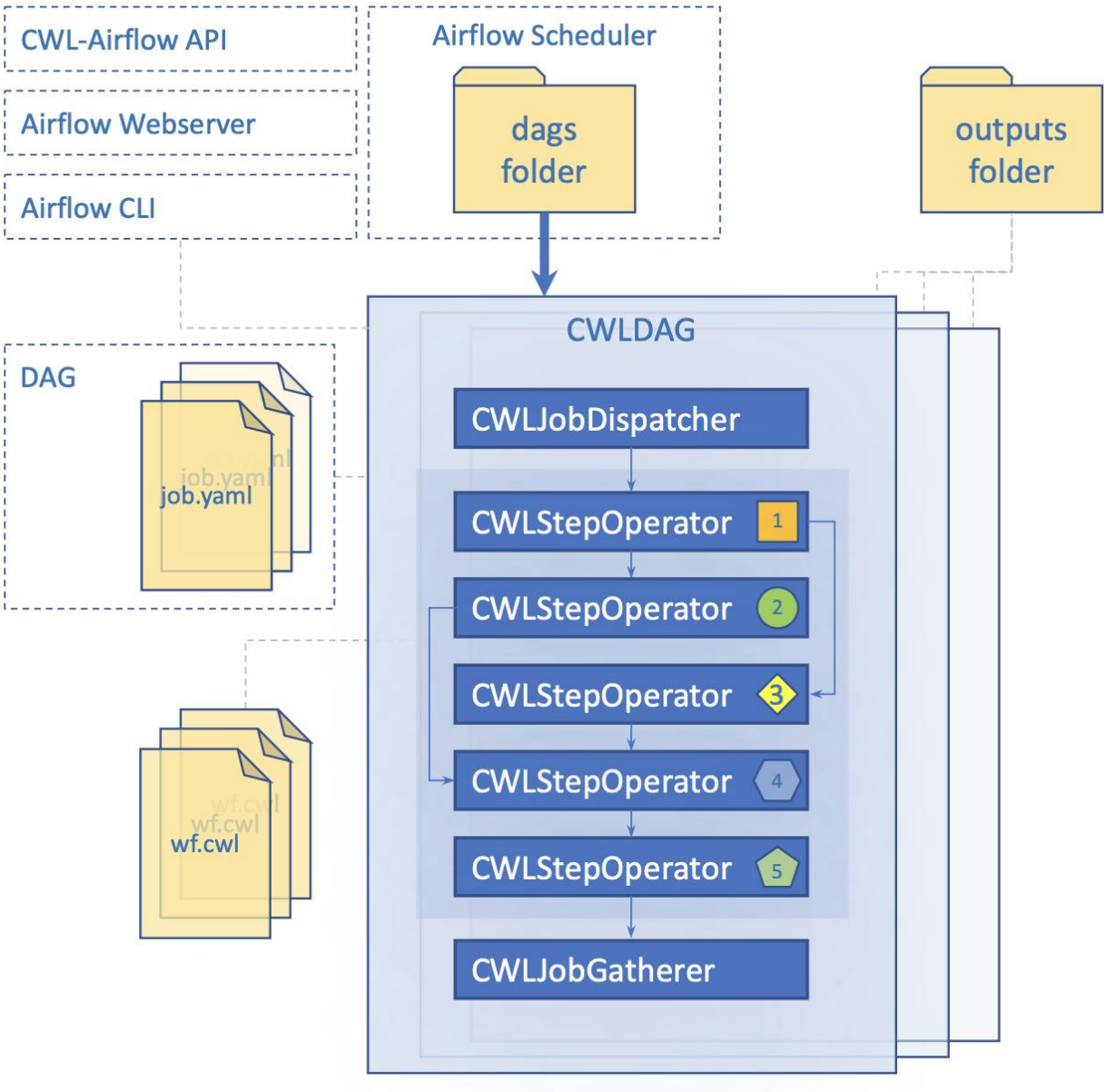
   If any of the abovementioned parameters was not set the default value will be derived from `[cwl]` section of **airflow.cfg**. For additional details refer to Updating airflow.cfg section.
3. **DAG** - directed acyclic graph that describes workflow structure.

Note, for better understanding of CWL specification and its basic principles, please, refer to the official CWL User Guide.

2.2.2 Concepts

The CWL-airflow package extends Airflow's functionality with the ability to parse and execute workflows written with the CWL v1.1 specification. We defined 4 basic components — CWLJobDispatcher, CWLStepOperator, CWLJobCleanup, and CWLDAG. The latter is a class for combining the tasks into a DAG that reflects the CWL workflow structure. Every CWLStepOperator task corresponds to a workflow step and depends on others on the basis of the workflow step inputs and outputs. CWLJobDisptacher is used to provide the pipeline with the input data. CWLJobCleanup returns the calculated results to the output folder. Every new CWL workflow results in creation of a new CWLDAG. If the new job is run with the same pipeline, it will not create a new CWLDAG, but run the old one.

Previously, in order to execute CWLDAG a file describing workflow-specific input parameters in JSON or YAML format should have been placed in the special jobs folder. In the current version we removed the necessity for the jobs folder, as the new CWLDAGs can be easily triggered with the required input parameters through the REST API, Airflow UI or command line interface. In case someone needs to monitor a special folder for the new job files added, it can be easily implemented as a separate standard for Airflow DAG.



To add a new workflow, one should simply write a small python script (see example below) and place it into the DAGs folder. Only two parameters are required to initialize a new CWLDAG: path to the **workflow** file and **dag_id**.

```
#!/usr/bin/env python3
from cwl_airflow.extensions.cwldag import CWLDAG
dag = CWLDAG(workflow="my_awesome_workflow.cwl", dag_id="my_awesome_dag")
```

There are only three functions that our CWLDAG is responsible for. First – to parse CWL file. Second – to validate CWL syntax. Third – to create a DAG, that will have the same structure as our workflow.

2.3 How to install

2.3.1 Install requirements

Ubuntu 18.04.4 (Bionic Beaver)

- python3-dev

```
sudo apt-get install python3-dev
```

macOS 11.0.1 (Big Sur)

- Apple Command Line Tools

```
xcode-select --install
```

Both Ubuntu and macOS

- python 3.6 / 3.7 / 3.8
- docker (follow the installation guides)
- pip (follow the installation guides)
- setuptools

```
pip3 install -U setuptools
```

2.3.2 Install CWL-airflow

```
$ pip3 install cwl-airflow \
--constraint "https://raw.githubusercontent.com/Barski-lab/cwl-airflow/master/packaging/
↪constraints/constraints-3.7.txt"
```

When using optional `--constraint` parameter you can limit dependencies to those versions that were tested with your Python.

Optionally, extra dependencies can be provided by adding `[mysql,celery,statsd]` at the end of the command above.

- **mysql** - enables MySQL server support
- **celery** - enables Celery cluster support

- **statsd** - enables StatsD metrics support

2.3.3 Download portable version of CWL-airflow

Alternatively to installation, the relocatable standalone **Python3 with pre-installed CWL-Airfow** can be downloaded from the Releases section on GitHub.

Note, these are **not** cross-platform packages, so the version of OS should be the same as mentioned in the name of the file. When extracted from archive, all executables can be found in the `python3/bin_portable` folder.

Similar packages for other versions of Ubuntu, Python and CWL-Airflow can be generated with the following commands:

```
# Ubuntu
# defaults: Ubuntu 18.04, Python 3.6, CWL-Airflow master branch

$ ./packaging/portable/ubuntu/pack.sh [UBUNTU_VERSION] [PYTHON_VERSION] [CWL_AIRFLOW_
↪VERSION]

# macOS
# package is always built for current macOS version
# defaults: Python 3.8, CWL-Airflow master branch

$ ./packaging/portable/macos/pack.sh [PYTHON_VERSION] [CWL_AIRFLOW_VERSION]
```

2.4 How to use

2.4.1 Initial configuration

Before using **CWL-airflow** it should be configured with `cwl-airflow init`

```
$ cwl-airflow init --help

usage: cwl-airflow init [-h] [--home HOME] [--config CONFIG] [--upgrade]

optional arguments:
  -h, --help       show this help message and exit
  --home HOME      Set path to Airflow home directory. Default: first try
                   AIRFLOW_HOME then '~/airflow'
  --config CONFIG  Set path to Airflow configuration file. Default: first try
                   AIRFLOW_CONFIG then '[airflow home]/airflow.cfg'
  --upgrade        Upgrade old CWLDAG files to the latest format. Default:
                   False
```

Init command will run the following steps for the specified `--home` and `--config` parameters:

- Call `airflow --help` to create a default `airflow.cfg`
- Update `airflow.cfg` to hide paused DAGs, skip loading example DAGs and connections and **do not** pause newly created DAGs. Also, we set our custom `logging_config_class` to split Airflow and CWL related logs into the separate files. In case of upgrading from the previous version of CWL-Airflow that used Airflow < 2.0.0 to the latest one, `airflow.cfg` will be backuped and upgraded to fit Airflow 2.1.4. You will have to manually make sure that all custom fields were properly copied to the new `airflow.cfg`

- Call `airflow db init` to init/upgrade Airflow metadata database.
- If run with `--upgrade`, upgrade old CWLDAGs to correspond to the latest format, save original CWLDAGs into `deprecated_dags` folder.
- Put **clean_dag_run.py** into the DAGs folder.

2.4.2 Updating airflow.cfg

For precise configuration the `[cwl]` section can be added to **airflow.cfg**. All of the parameters descibed below are **optional** and will take their default values if not provided.

If job already included absolute paths for **tmp_folder** and **outputs_folder** the corresponent parameters from **airflow.cfg** will be ignored.

In other situation, for example when running CWL-Airflow with `docker-compose`, one may need to set the exact locations for **tmp**, **outputs**, **inputs** and **pickle** folders to allow their proper mounting to Docker container.

Also, following the abovementioned scenario, all input files required for workflow execution might be placed into **inputs_folder**. At the same time, when using relative locations in the job file, all paths will be resolved based on the same **inputs_folder**. For additional details refer to *Running CWL-Airflow with docker-compose* section.

```
[cwl]

# Temp folder to keep intermediate workflow execution data.
# Ignored if job already has tmp_folder set as absolute path.
# If job has tmp_folder set as a relative path, it will be resolved based on this␣
↪location.
# Default: AIRFLOW_HOME/cwl_tmp_folder
tmp_folder =

# Output folder to save workflow execution results.
# Ignored if job already has outputs_folder set as absolute path.
# If job has outputs_folder set as a relative path, it will be resolved based on this␣
↪location.
# Default: AIRFLOW_HOME/cwl_outputs_folder
outputs_folder =

# Folder to keep input files.
# If job has relative paths for input files they will be resolved based on this location.
# Default: AIRFLOW_HOME/cwl_inputs_folder
inputs_folder =

# Folder to keep pickled workflows for fast workflow loading.
# Default: AIRFLOW_HOME/cwl_pickle_folder
pickle_folder =

# Boolean parameter to force using docker for workflow step execution.
# Default: True
use_container =

# Boolean parameter to disable passing the current user id to "docker run --user".
# Default: False
no_match_user =
```

2.4.3 Adding a pipeline

Set absolute path to the workflow file

The easiest way to add a new pipeline to CWL-airflow is to put the following python script into your DAGs folder. Here, `workflow` parameter is initialized with the absolute path to the CWL workflow file.

```
#!/usr/bin/env python3
from cwl_airflow.extensions.cwldag import CWLDAG
dag = CWLDAG(
    workflow="/absolute/path/to/workflow.cwl",
    dag_id="my_dag_name"
)
```

As `CWLDAG` class was inherited from Airflow's `DAG`, additional arguments, such as `default_args`, can be provided when calling class constructor.

`default_args` can also include `cwl` section similar to the one from **airflow.cfg** file described in *Updating airflow.cfg* section. However, parameters from **airflow.cfg** will always **have higher priority** compared to those that were passed in constructor.

Use zlib compressed workflow file content

Alternatively to file location, the value of `workflow` parameter can be initialized with **base64 encoded zlib compressed** file content. Below is an **example of script** generating compressed workflow content.

```
from cwl_airflow.utilities.helpers import get_compressed
with open("workflow.cwl", "r") as input_stream:
    print(get_compressed(input_stream))
```

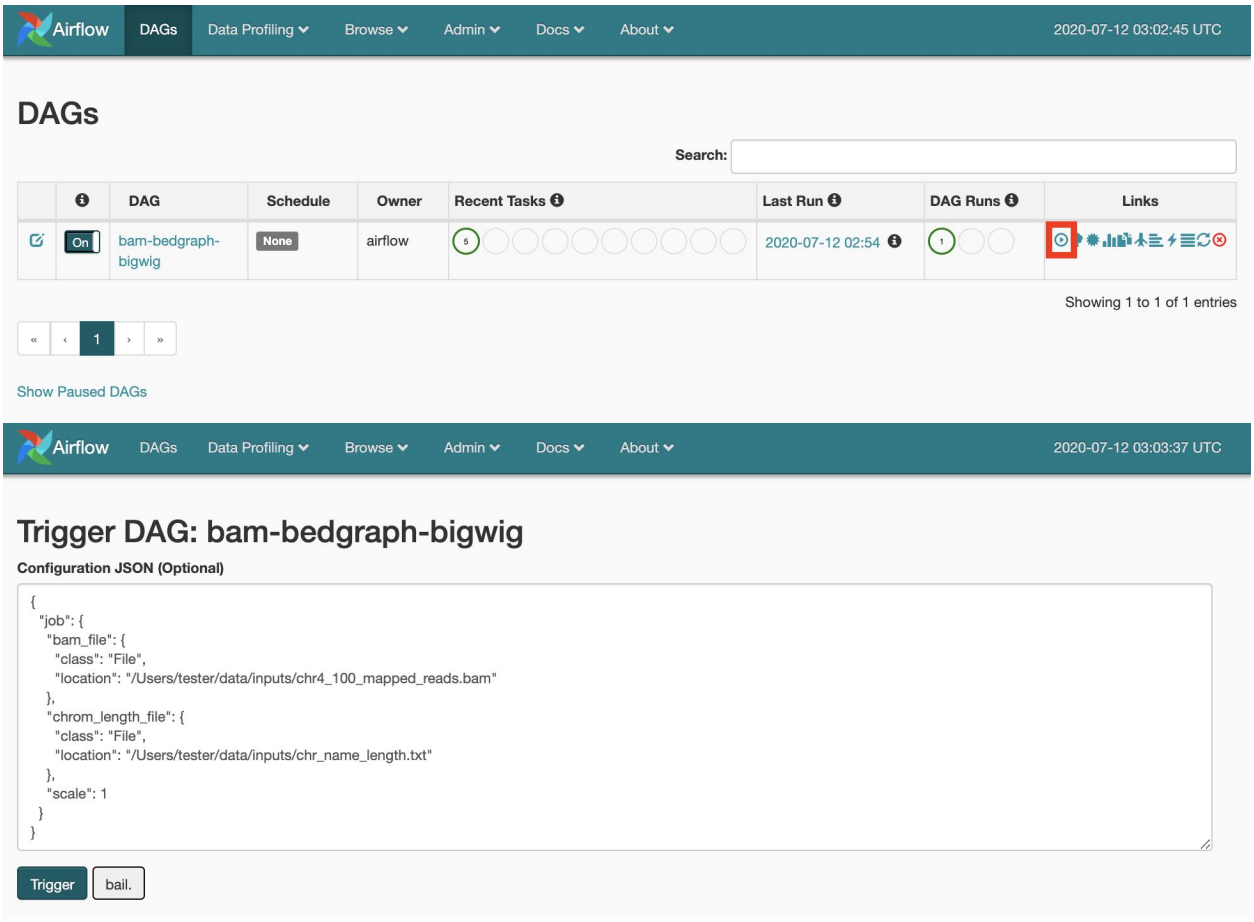
Note, to add a new pipeline one can also use **POST** to `/dags` API endpoing. For additional details refer to *Using an API* section.

A new pipeline can be run after Airflow Scheduler loads new DAG (by default if happens **every 5 minutes**).

2.4.4 Executing a pipeline

Using Airflow UI

The most convenient way to **manually execute** DAG is to trigger it from **Airflow UI**. Input parameters can be set in the **job** section of the DAG run configuration as in the example below.



Using Airflow CLI

Alternatively, DAGs can be triggered through the **Airflow CLI** with the JSON input paramerers file.

```
$ airflow trigger_dag --conf "{\"job\":$(cat ./bam-bedgraph-bigwig.json)}" bam-bedgraph-
↪bigwig
```

Note, to trigger workflow execution one can also use **POST** to `/dag_runs` API endpoing. For additional details refer to *Using an API* section.

2.4.5 Posting pipeline execution progress, statistics and results

To make CWL-Airflow **post workflow executions progress, statistics and results** `process_report` connection should be added. Parameters can be adjusted based on the current needs following the example below.

```
$ airflow connections add process_report --conn-type http --conn-host localhost --conn-
↪port 3070
```

In case CWL-Airflow failed to POST progress updates or workflow execution results, the corresponded records with the prefixes `post_progress__` and `post_results__` will be added to the Airflow Variables. Later, when CWL-Airlfow API run with `--replay N` argument, it will attemt to resend not delivered messages every N seconds. Workflow execution statistics is sent as part of the progress report at the end of the pipeline execution regardless of whether it finished with success or failure. If progress report is sent from the task, the statistics will be set to "".

On the example below, the workflow execution statistics includes `total` section with the `start_date` in isoformat. This timestamp will be used as a reference point for all other `start_date` and `end_date` fields which are represented in `seconds.milliseconds` format. All `tmp_folder_size` and `outputs_folder_size` are in kBytes.


```
{
  'state': 'success',
  'dag_id': 'star-index',
  'run_id': 'ba46dd51-9c7d-4f92-adc5-503a812ddb6d',
  'progress': 100,
  'statistics':
  {
    'version': '1.0',
    'total':
    {
      'tmp_folder_size': 3080904,
      'outputs_folder_size': 1538044,
      'start_date': '2021-01-28T20:55:03.258202+00:00',
      'end_date': 60.715
    },
    'steps':
    {
      'CWLJobDispatcher':
      {
        'tmp_folder_size': 4,
        'start_date': 2.69,
        'end_date': 6.96
      },
      'CWLJobGatherer':
      {
        'tmp_folder_size': 0,
        'start_date': 56.534,
        'end_date': 58.718
      },
      'star_generate_indices':
      {
        'tmp_folder_size': 3080900,
        'start_date': 10.657,
        'end_date': 52.23
      }
    }
  },
  'error': '',
  'logs': ''
}
```

2.4.6 Using an API

Besides built-in experimental API from the Airflow Webserver, CWL-airflow provides **extended API** that supports WES and can be run with `cwl-airflow api`

```
$ cwl-airflow api --help

usage: cwl-airflow api [-h] [--port PORT] [--host HOST]

optional arguments:
  -h, --help   show this help message and exit
  --port PORT  Set port to run API server. Default: 8081
  --host HOST  Set host to run API server. Default: 127.0.0.1
  --simulation SIMULATION
               Set path to the test suite file to simulate reports.
               Pipelines won't get triggered in this mode.
  --replay     REPLAY
               Retries to post undelivered progress and results reports to
               the process_report connection every N seconds. If connection
               is not set this parameter is ignored.
               Default: do not resend not delivered reports.
```

Although, **detailed API specification** available on SwaggerHub, here we provide the **most commonly used endpoints**.

1. Get list of dags

`GET /dags`

Parameters:

Response example:

```
{
  "dags": [
    {
      "dag_id": "string",
      "tasks": [
        "string"
      ]
    }
  ]
}
```

2. Create new dag

`POST /dags`

Parameters:

Response example:

```
{
  "dag_id": "string",
  "dag_path": "string",
```

(continues on next page)

(continued from previous page)

```
  "cwl_path": "string"
}
```

3. Get list of dag_runs

`GET /dag_runs`

Parameters:

Enumerated values:

Response example:

```
{
  "dag_runs": [
    {
      "dag_id": "string",
      "run_id": "string",
      "execution_date": "2019-08-24T14:15:22Z",
      "start_date": "2019-08-24T14:15:22Z",
      "state": "running",
      "tasks": [
        {
          "id": "string",
          "state": "scheduled"
        }
      ],
      "progress": 0
    }
  ]
}
```

4. Trigger dag

`POST /dag_runs`

Parameters:

Response example:

```
{
  "dag_id": "string",
  "run_id": "string",
  "execution_date": "2019-08-24T14:15:22Z",
  "start_date": "2019-08-24T14:15:22Z",
  "state": "running"
}
```

2.4.7 Running CWL-Airflow with docker-compose

To start CWL-Airflow with LocalExecutor using docker-compose, run the following commands

```
cd ./packaging/docker_compose/local_executor
docker-compose up --build
```

Default values for mount volumes, mapped ports and other configurations can be found `.env` file in the same folder.

2.5 What if is doesn't work

2.5.1 CWL-airflow is not found

Perhaps, you have installed it with `--user` option and your **PATH** variable doesn't include your user based Python3 **bin** folder. Update **PATH** with the following command

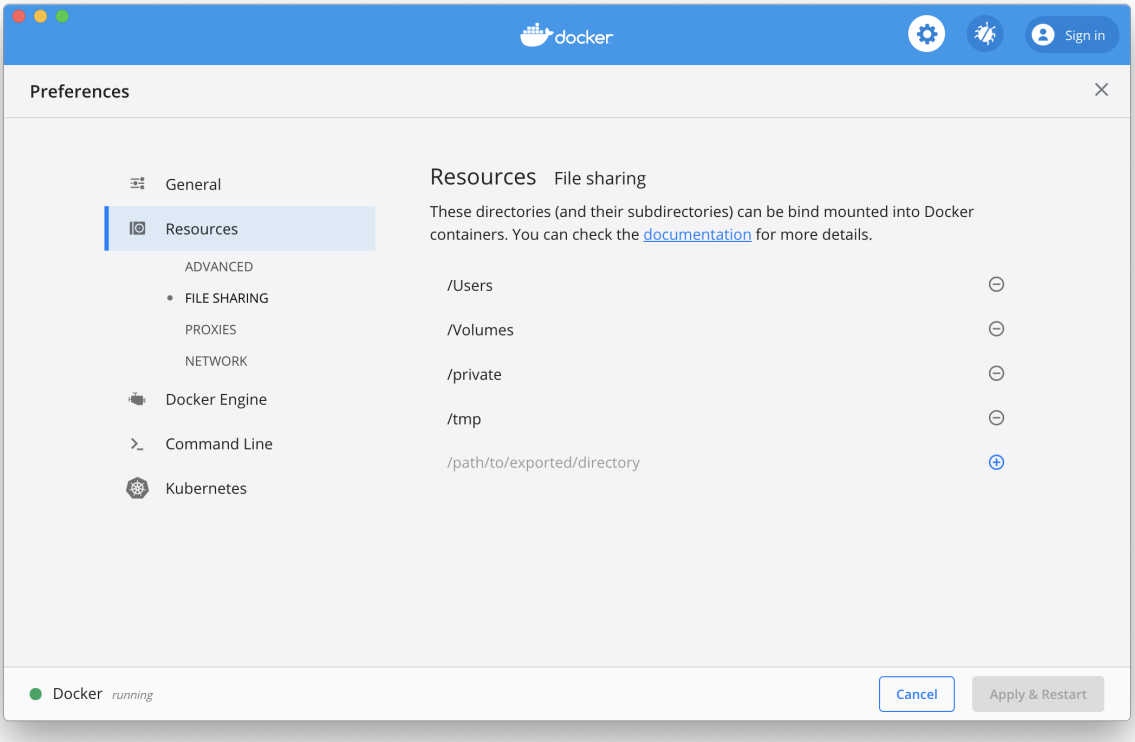
```
$ export PATH="$PATH:`python3 -m site --user-base`/bin"
```

2.5.2 Docker is unable to pull images from the Internet

If you are using proxy, your Docker should be configured properly too. Refer to the official documentation.

2.5.3 Docker is unable to mount input files

When running Docker App on **macOS** there is a default list of directories that Docker has permission to mount. If your input files are located in the directories that are not included in this list, you should add them in **Preferences / Resources / File Sharing**.



2.5.4 Missing DAGs in Airflow UI

If after adding a new DAG you don't see it in Airflow UI, first check if Airflow Scheduler is running, then make sure that **dag_dir_list_interval** parameter in **airflow.cfg** is not too big. By default, Airflow Scheduler will check for new DAGs every 5 minutes.

2.5.5 Workflow execution failed

Make sure that your CWL descriptor file is correct and DAG was triggered with correct input parameters. You can always check it with **cwltool** of the same version that is included in CWL-airflow package.

```
cwltool --debug WORKFLOW JOB
```

2.5.6 Fails to compile ruamel.yaml

Perhaps, you should update your **setuptools** and try to reinstall **ruamel.yaml**

2.5.7 mysql_config not found

When running on Ubuntu with MySQL backend, it might be necessary to install **libmysqlclient-dev**

```
sudo apt-get install libmysqlclient-dev
```

HTTP ROUTING TABLE

/dag_runs

GET /dag_runs, ??
POST /dag_runs, ??

/dags

GET /dags, ??
POST /dags, ??
POST /dags/dag_runs, ??
POST /dags/{dag_id}/dag_runs, ??

/wes

GET /wes/runs, ??
GET /wes/runs/{run_id}, ??
GET /wes/runs/{run_id}/status, ??
GET /wes/service-info, ??
POST /wes/runs, ??
POST /wes/runs/{run_id}/cancel, ??