CWL-Airflow

Jun 29, 2019

Contents:

Python package to extend Apache-Airflow 1.9.0 functionality with CWL v1.0 support.

CHAPTER 1

Cite as

Kotliar M; Kartashov AV; Barski A (2019): Supporting data for "CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language" GigaScience Database. http://dx.doi.org/10.5524/100618

1.1 Run demo

1.1.1 Locally

We assume that you have already installed and properly configured **python**, 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 Installation or Troubleshooting sections.

1. Install *cwl-airflow*

```
$ pip install cwl-airflow==1.0.16 --find-links https://michael-kotliar.github.io/
↪cwl-airflow-wheels/ # --user
```

`--user` - explained in Installation section

2. Init configuration

```
$ cwl-airflow init
```

3. Run *demo*

```
$ cwl-airflow demo --auto
```

For every submitted workflow you will get the following information

```
CWL-Airflow demo mode
Process demo workflow 1/3
Load workflow
- workflow:    # path from where we load the workflow file
```

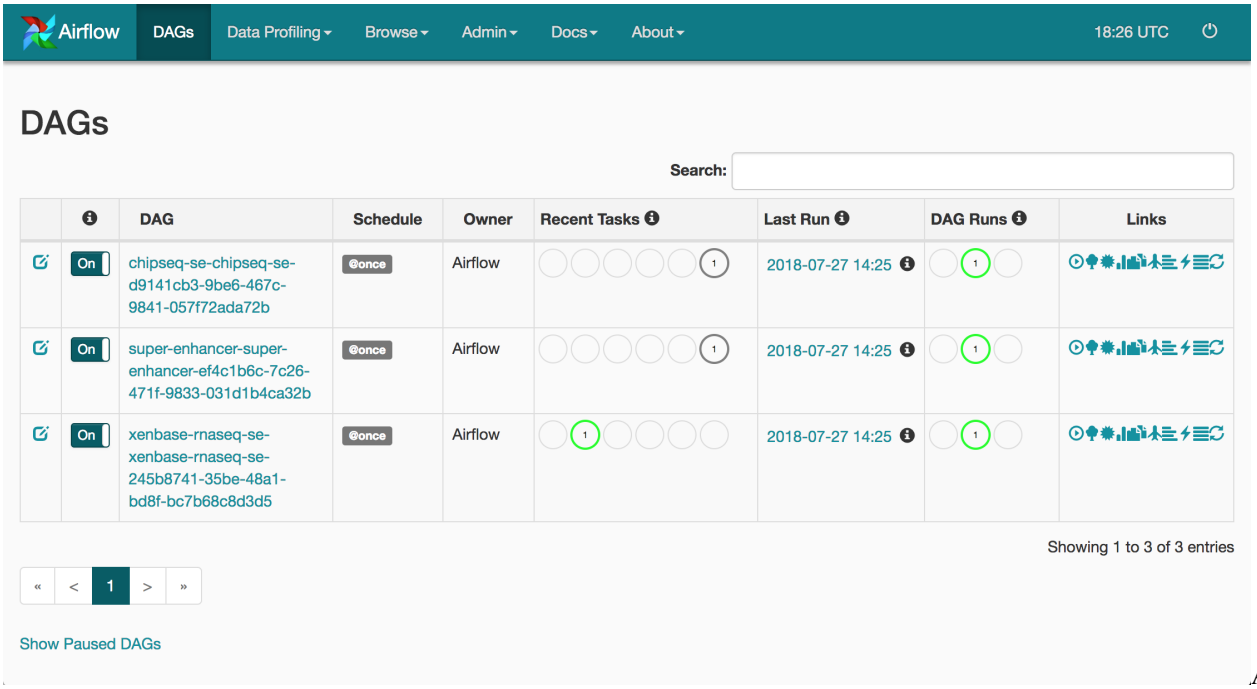
(continues on next page)

(continued from previous page)

```
- job:          # path from where we load the input parameters file
- uid:          # unique identifier for the submitted job
Save job file as
-               # path where we save submitted job for CWL-Airflow to run
```

`uid` - the unique identifier used for DAG ID and output folder name generation.

4. When all demo wokrflows are submitted the program will provide you with the link for Airflow web interface (by default it is accessible from your localhost:8080). It may take some time (usually less then half a minute) for Airflow web interface to load and display all the data.
5. On completion the workflow results will be saved in the current folder.



Airflow web interface

1.1.2 VirtualBox

In order to run CWL-Airflow virtual machine you have to install Vagrant and VirtualBox. The host machine should have access to the Internet, at least 8 CPUs and 16 GB of RAM.

1. Clone CWL-Airflow repository

```
$ git clone https://github.com/Barski-lab/cwl-airflow
```

2. Chose one of three possible configurations to run

   Single node

```
$ cd ./cwl-airflow/vagrant/local_executor
```

   Celery Cluster of 3 nodes (default queue)

```
$ cd ./cwl-airflow/vagrant/celery_executor/default_queue
```

Celery Cluster of 4 nodes (default + advanced queues)

```
$ cd ./cwl-airflow/vagrant/celery_executor/custom_queue
```

3. Start virtual machine

```
$ vagrant up
```

Vagrant will pull the latest virtual machine image (about 3.57 GB) from Vagrant Cloud. When started the following folders will be created on the host machine in the current directory.

```
├── .vagrant
└── airflow
    ├── dags
    │   └── cwl_dag.py      # creates DAGs from CWLs
    ├── demo
    │   ├── cwl
    │   │   ├── subworkflows
    │   │   ├── tools
    │   │   └── workflows   # demo workflows
    │   │       ├── chipseq-se.cwl
    │   │       ├── super-enhancer.cwl
    │   │       └── xenbase-rnaseq-se.cwl
    │   ├── data            # input data for demo workflows
    │   └── job             # sample job files for demo workflows
    │       ├── chipseq-se.json
    │       ├── super-enhancer.json
    │       └── xenbase-rnaseq-se.json
    ├── jobs                # folder for submitted job files
    ├── results             # folder for workflow outputs
    └── temp                # folder for temporary data
```

4. Connect to running virtual machine through `ssh`

```
$ vagrant ssh master
```

5. Submit all demo workflows for execution

```
$ cd /home/vagrant/airflow/results
$ cwl-airflow demo --manual
```

For every submitted workflow you will get the following information

```
CWL-Airflow demo mode
Process demo workflow 1/3
Load workflow
- workflow:    # path from where we load the workflow file
- job:         # path from where we load the input parameters file
- uid:         # unique identifier for the submitted job
Save job file as
-              # path where we save submitted job for CWL-Airflow to run
```

`uid` - the unique identifier used for DAG ID and output folder name generation.

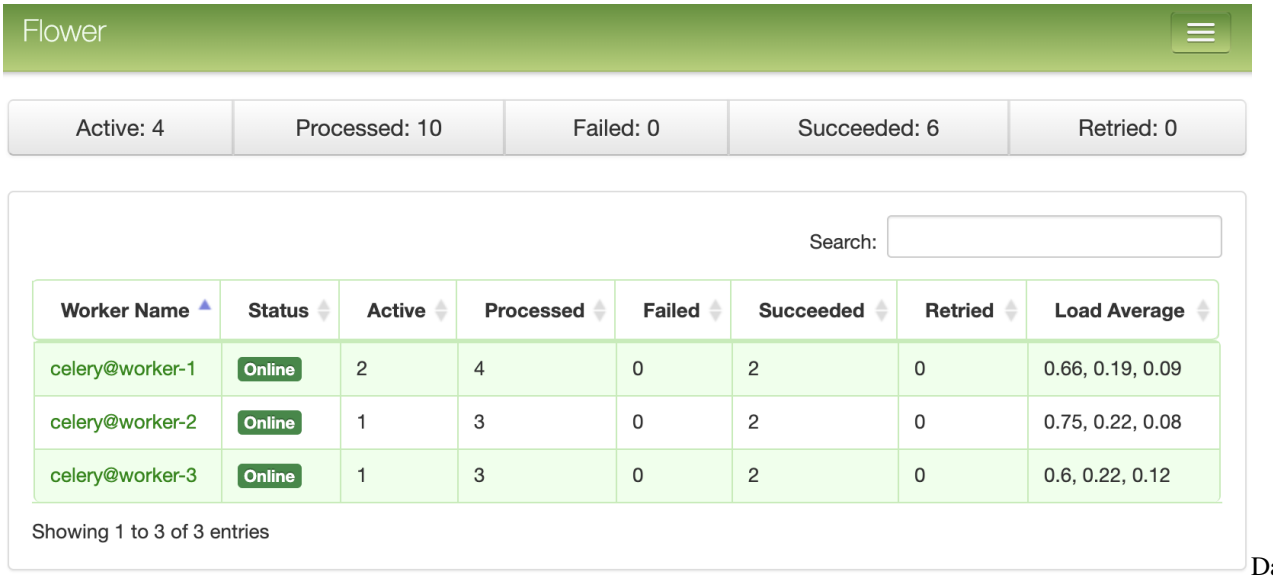
6. Open Airflow web interface (localhost:8080) and, if multi-node configuration is run, Celery Flower Monitoring Tool (localhost:5555). It might take up to 20 seconds for Airflow web interface to display all newly added workflows.

7. On completion, you can view workflow execution results in the `/home/vagrant/airflow/results` folder of the Virtual Machine or in `./airflow/results` folder on your host machine.
8. Stop `ssh` connection to the virtual machine by pressing `ctlr+D` and then run one of the following commands

```
$ vagrant halt              # stop virtual machines
```

or

```
$ vagrant destroy           # remove virtual machines
$ rm -rf ./airflow .vagrant  # remove created folders
```



Dashboard of the Celery monitoring tool Flower

1.2 How It Works

1.2.1 Key concepts

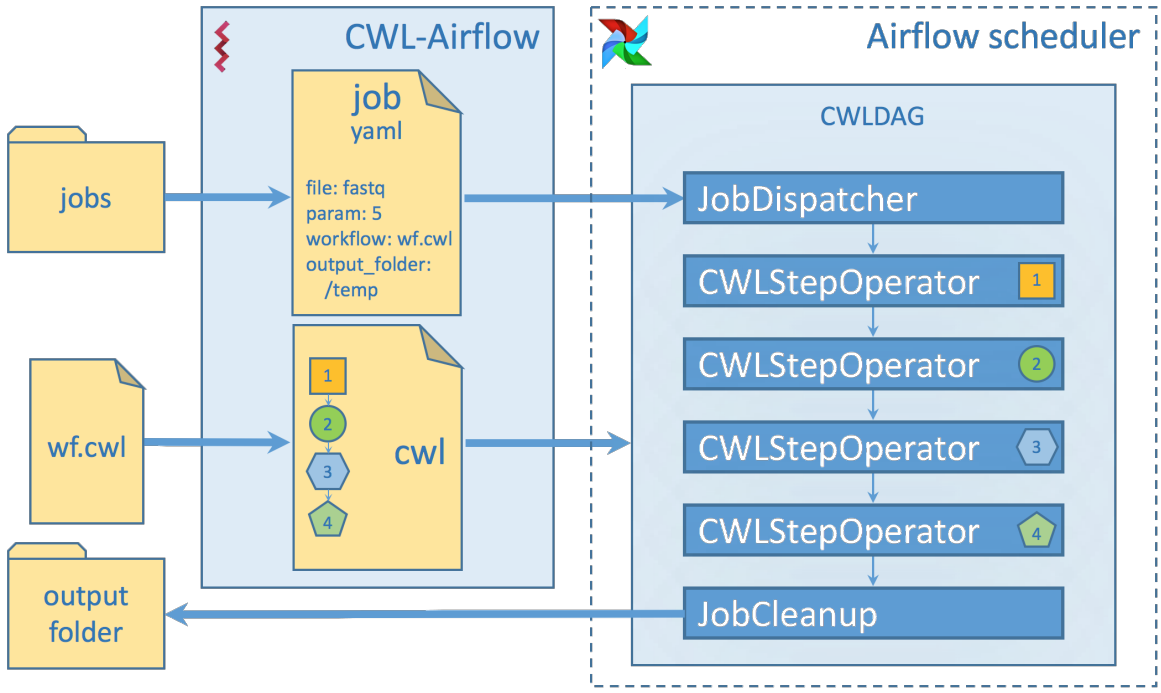
1. **CWL descriptor file** - *YAML* or *JSON* file to describe the workflow inputs, outputs and steps. File should be compatible with CWL v1.0 specification
2. **Job file** - *YAML* or *JSON* file to set the values for the wokrflow inputs. For *cwl-airflow* to function properly the Job file should include 3 mandatory and one optional fields:
   - *workflow* - mandatory field to specify the absolute path to the CWL descriptor file
   - *output_folder* - mandatory field to specify the absolute path to the folder where all the output files should be moved after successful workflow execution
   - *tmp_folder* - optional field to specify the absolute path to the folder for storing intermediate results. After workflow execution this folder will be deleted.
   - *uid* - mandatory field that is used for generating DAG's unique identifier.
3. **DAG** - directed acyclic graph that describes the workflow structure.
4. **Jobs folder** - folder to keep all Job files scheduled for execution or the ones that have already been processed. The folder's location is set as *jobs* parameter of *cwl* section in Airflow configuration file.

1.2.2 What's inside

To build a workflow *cwl-airflow* uses three basic classes:

- *CWLStepOperator* - executes a separate workflow step
- *JobDispatcher* - serializes the Job file and provides the worflow with input data
- *JobCleanup* - returns the calculated results to the output folder

A set of *CWLStepOperator*s, *JobDispatcher* and *JobCleanup* are combined in *CWLDAG* that defines a graph to reflect the workflow steps, their relationships and dependencies. Automatically generated *cwl_dag.py* script is placed in the DAGs folder. When Airflow Scheduler loads DAGs from the DAGs folder, the *cwl_dag.py* script parses all the Job files from the Jobs folder and creates DAGs for each of them. Each DAG has a unique DAG ID that is formed accodring to the following scheme: `CWL descriptor file basename-Job file basename-uid field from the Job file`



CWL-Airflow diagram

1.3 Installation

1.3.1 Requirements

Ubuntu 16.04.4 (Xenial Xerus)

- python 2.7 or 3.5 (tested on the system Python 2.7.12 and 3.5.2)
- docker (follow the link to install Docker on Ubuntu)

  Don't forget to add your user to the *docker* group and then to log out and log back in so that your group membership is re-evaluated.
- python-dev (or python3-dev if using Python 3.5)

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

python-dev is required in case your system needs to compile some python packages during the installation. We have built python *wheels* for most of such packages and provided them through *–find-links* argument while installing *cwl-airflow*. Nevertheless in case of installation problems you might still be required to install this dependency.

macOS 10.13.5 (High Sierra)

- python 2.7 or 3.6 (tested on the system Python 2.7.10 and brewed Python 2.7.15 / 3.6.5; **3.7.0 is not supported**)
- docker (follow the link to install Docker on Mac)
- Apple Command Line Tools

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

Click *Install* on the pop up window when it appears, follow the instructions. *Apple Command Line Tools* are required in case your system needs to compile some python packages during the installation. We have built python wheels for most of such packages and provided them through *–find-links* argument while installing *cwl-airflow*. Nevertheless in case of installation problems you might still be required to install this dependency.

Both Ubuntu and macOS

- pip (follow the link to install the latest stable Pip)

  Consider using `--user` if you encounter permission problems

- setuptools (tested on setuptools 40.0.0)

```
pip install -U setuptools # --user
```

`--user` - optional parameter to install all the packages into your *HOME* directory instead of the system Python directories. It will be helpful if you don't have enough permissions to install new Python packages. You might also need to update your *PATH* variable in order to have access to the installed packages (an easy way to do it is described in Troubleshooting section). If installing on macOS brewed Python `--user` **should not** be used (explained here)

1.3.2 Install cwl-airflow

```
$ pip install cwl-airflow==1.0.16 --find-links https://michael-kotliar.github.io/cwl-
↪airflow-wheels/ # --user
```

`--find-links` - using pre-compiled wheels from Cwl-Airflow-Wheels repository allows to avoid installing *Xcode* for macOS users and *python[3]-dev* for Ubuntu users

1.4 Using cwl-airflow

1.4.1 Configuration

Before using *cwl-airflow* it should be initialized with the default configuration by running the command

```
$ cwl-airflow init
```

Optional parameters:

Consider using `-r 5 -w 4` to make Airflow Webserver react faster on all newly created DAGs

If you update Airflow configuration file manually (default location is *~/airflow/airflow.cfg*), make sure to run *cwl-airflow init* command to apply all the changes, especially if *core/dags_folder* or *cwl/jobs* parameters from the configuration file are changed.

1.4.2 Submitting new job

To submit new CWL descriptor and Job files to *cwl-airflow* run the following command

```
cwl-airflow submit WORKFLOW JOB
```

Optional parameters:

Arguments `-o`, `-t` and `-u` doesn't overwrite the values from the Job file set in the fields *output_folder*, *tmp_folder* and *uid* correspondingly. The meaning of these fields is explaned in How It Works section.

The *submit* command will resolve all relative paths from Job file adding mandatory fields *workflow*, *output_folder* and *uid* (if not provided) and will copy Job file to the Jobs folder. The CWL descriptor file and all input files referenced in the Job file should not be moved or deleted while workflow is running. The *submit* command will **not** execute submitted workflow unless *-r* argument is provided. Otherwise, make sure that *Airflow Scheduler* (and optionally *Airflow Webserver*) is running. Note, that *-r* argument was added only to comply with the interface through which CWL community runs it's conformance tests. So it's more preferable to execute submitted workflow with *Airflow Scheduler*, especially if you are planning to use `LocalExecutor` instead of default `SequentialExecutor`.

Depending on your Airflow configuration it may require some time for Airflow Scheduler and Webserver to pick up new DAGs. Consider using `cwl-airflow init -r 5 -w 4` to make Airflow Webserver react faster on all newly created DAGs.

To start Airflow Scheduler (**don't** run it if *cwl-airflow submit* is used with *-r* argument)

```
airflow scheduler
```

To start Airflow Webserver (by default it is accessible from your localhost:8080)

```
airflow webserver
```

Please note that both Airflow Scheduler and Webserver can be adjusted through the configuration file (default location is *~/airflow/airflow.cfg*). Refer to the official documentation

1.4.3 Demo mode

- To get the list of the available demo workflows

  ```
  $ cwl-airflow demo --list
  ```

- To submit the specific demo workflow from the list (workflow will not be run until Airflow Scheduler is started separately)

  ```
  $ cwl-airflow demo super-enhancer.cwl
  ```

Depending on your Airflow configuration it may require some time for Airflow Scheduler and Webserver to pick up new DAGs. Consider using `cwl-airflow init -r 5 -w 4` to make Airflow Webserver react faster on all newly created DAGs.

- To submit all demo workflows from the list (workflows will not be run until Airflow Scheduler is started separately)

```
$ cwl-airflow demo --manual
```

Before submitting demo workflows the Jobs folder will be automatically cleaned.

- To execute all available demo workflows (automatically starts Airflow Scheduler and Airflow Webserver)

```
$ cwl-airflow demo --auto
```

Optional parameters:

1.4.4 Running sample ChIP-Seq-SE workflow

This ChIP-Seq-SE workflow is a CWL version of a Python pipeline from BioWardrobe. It starts by extracting an input FASTQ file (if it was compressed). Next step runs BowTie to perform alignment to a reference genome, resulting in an unsorted SAM file. The SAM file is then sorted and indexed with Samtools to obtain a BAM file and a BAI index. Next MACS2 is used to call peaks and to estimate fragment size. In the last few steps, the coverage by estimated fragments is calculated from the BAM file and is reported in bigWig format. The pipeline also reports statistics, such as read quality, peak number and base frequency, as long as other troubleshooting information using tools such as Fastx-toolkit and Bamtools.

To get sample workflow with input data

```
$ git clone --recursive https://github.com/Barski-lab/ga4gh_challenge.git --branch v0.
↪0.5
$ ./ga4gh_challenge/data/prepare_inputs.sh
```

Please, be patient it may take some time to clone submodule with input data. Runing the script *prepare_inputs.sh* will uncompress input FASTQ file.

To submit worflow for execution

```
cwl-airflow submit ga4gh_challenge/biowardrobe_chipseq_se.cwl ga4gh_challenge/
↪biowardrobe_chipseq_se.yaml
```

To start Airflow Scheduler (**don't** run it if *cwl-airflow submit* is used with *-r* argument)

```
airflow scheduler
```

To start Airflow web interface (by default it is accessible from your localhost:8080)

```
airflow webserver
```

Pipeline was tested with

- macOS 10.13.6 (High Sierra)
- Docker
  - Engine: 18.06.0-ce
  - Machine: 0.15.0
  - Preferences

    * CPUs: 4
    * Memory: 2.0 GiB
    * Swap: 1.0 GiB
- Elapsed time: 23 min (may vary depending on you Internet connection bandwidth, especially when pipeline is run for the first time and all Docker images are being fetched from DockerHub)

1.5 Troubleshooting

Most of the problems are already handled by *cwl-airflow* itself. User is provided with the full explanation and ways to correct them through the console output. Additional information regarding the failed workflow steps, can be found in the task execution logs that are accessible through Airflow Webserver UI.

Common errors and ways to fix them

- ***`cwl-airflow` is not found***

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

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

- ***Fails to install on the latest Python 3.7.0***

  Unfortunatelly *Apache-Airflow 1.9.0* cannot be properly installed on the latest *Python 3.7.0*. Consider using *Python 3.6* or *2.7* instead.

  macOS users can install Python 3.6.5 (instead of the latest Python 3.7.0) with the following command (explained here)

```
  brew install https://raw.githubusercontent.com/Homebrew/homebrew-core/
↪f2a764ef944b1080be64bd88dca9a1d80130c558/Formula/python.rb
```

- ***Fails to compile ruamel.yaml***

  Perhaps, you should update your *setuptools*. Consider using *–user* if necessary. If installing on macOS brewed Python *–user* **should not** be used (explained here)

```
pip install -U setuptools # --user
```

  `--user` - explained in Installation section

- ***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

- ***Docker is unable to mount directory***

  For macOS docker has a list of directories that it's allowed to mount by default. If your input files are located in the directories that are not included in this list, you are better of either changing the location of input files and updating your Job file or adding this directories into Docker configuration *Preferences / File Sharing*.

- ***Airflow Webserver displays missing DAGs***

  If some of the Job files have been manually deleted, they will be still present in Airflow database, hence they will be displayed in Webserver's UI. Sometimes you may still see missing DAGs because of the inertness of Airflow Webserver UI.

- ***Airflow Webserver randomly fails to display some of the pages***

  When new DAG is added Airflow Webserver and Scheduler require some time to update their states. Consider using `cwl-airflow init -r 5 -w 4` to make Airflow Webserver react faster for all newly created DAGs. Or manualy update Airflow configuration file (default location is *~/airflow/airflow.cfg*) and restart both Webserver and Scheduler. Refer to the official documentation here

- ***Workflow execution fails***

  Make sure that CWL descriptor and Job files are correct. You can always check them with *cwltool* (trusted version 1.0.20180622214234)

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