Cromwell-tools Documentation

Release 2.4.1.dev4+gfb1753d

Mint Team

Feb 19, 2020

Overview

CHAPTER 1

Cromwell-tools

1.1 Overview

This repo contains a cromwell_tools Python package, accessory scripts and IPython notebooks.

The cromwell_tools Python package is designed to be a Python API and Command Line Tool for interacting with the Cromwell

- Python3 compatible. (Starting from release v2.0.0, cromwell-tools will no longer support Python 2.7)
- Consistency in authentication to work with Cromwell.
- Consistency between API and CLI interfaces.
- Sufficient test cases.
- Documentation on Read The Docs.

The accessory scripts and IPython notebooks are useful to:

- Monitor the resource usages of workflows running in Cromwell.
- Visualize the workflows benchmarking metrics.

1.2 Installation

1. (optional and highly recommended) Create a Python 3 virtual environment locally and activate it: e.g. `virtualenv -p python3 myenv && source myenv/bin/activate`

2. Install (or upgrade) Cromwell-tools from PyPI:

```
pip install -U cromwell-tools
```

3. You can verify the installation by:

```
cromwell-tools --version
```

1.3 Usage

1.3.1 Python API

In Python, you can import the package with:

```
import cromwell_tools.api as cwt
cwt.submit(*args)
```

assuming args is a list of arguments needed.

For more details, please check the tutorial on Read the Docs.

1.3.2 Commandline Interface

This package also installs a command line interface that mirrors the API and is used as follows:

A set of sub-commands to submit, query, abort, release on-hold workflows, wait for workflow completion and determining status of jobs are exposed by this CLI.

For more details, please check the tutorial on Read the Docs.

1.4 Testing

To run tests:

1.4.1 Run Tests with Docker

Running the tests within docker image is the recommended way, to do this, you need to have docker-daemon installed in your environment. From the root of the cromwell-tools repo:

1.4.2 Run Tests with local Python environment

- If you have to run the tests with your local Python environment, we highly recommend to create and activate a virtualenv with requirements before you run the tests:
- Finally, from the root of the cromwell-tools repo, run the tests with:

Note: Which version of Python is used to run the tests here depends on the virtualenv parameter. You can use `virtualenv -p` to choose which Python version you want to create the virtual environment.

1.5 Development

1.5.1 Code Style

The cromwell-tools code base is complying with the PEP-8 and using Black to format our code, in order to avoid "nitpicky" comments during the code review process so we spend more time discussing about the logic, not code styles.

In order to enable the auto-formatting in the development process, you have to spend a few seconds setting up the `pre-commit` the first time you clone the repo:

1. Install `pre-commit` by running: `pip install pre-commit` (or simply run `pip install -r requirements.txt`).
2. Run *pre-commit install* to install the git hook.

Once you successfully install the `pre-commit` hook to this repo, the Black linter/formatter will be automatically triggered and run on this repo. Please make sure you followed the above steps, otherwise your commits might fail at the linting test!

If you really want to manually trigger the linters and formatters on your code, make sure `Black` and `flake8` are installed in your Python environment and run `flake8 DIR1 DIR2` and `black DIR1 DIR2 --skip-string-normalization` respectively.

1.5.2 Dependencies

When upgrading the dependencies of cromwell-tools, please make sure `requirements.txt`, `requirements-test.txt` and `setup.py` are consistent!

1.5.3 Documentation

To edit the docmentation and rebuild it locally, make sure you have Sphinx installed. You might also want to install the dependencies for building the docs: `pip install requirements-docs.txt`. Finally from within the root directory, run:

and then you could preview the built documentation by opening `docs/_build/index.html` in your web browser.

1.5.4 Publish on PyPI

To publish a new version of Cromwell-tools on PyPI:

1. Make sure you have an empty `dist` folder locally.
2. Make sure you have `twine` installed: `pip install twine`.
3. Build the package: `python setup.py sdist bdist_wheel`
4. Upload and publish on PyPI: `twine upload dist/* --verbose`, note you will need the username and password of the development account to finish this step.

1.6 Contribute

Coming soon… For now, feel free to submit issues and open a PR, we will try our best to address them.

CHAPTER 2

Cromwell-tools Python API Quickstart

This notebook will help you get familiar with the Cromwell-tools' Python API client. After walking through this notebook you should be able to import cromwell-tools in your Python application and talk to Cromwell engine using various authentication methods.

2.1 Standard Authentication

2.1.1 1. [Recommended] Authenticate with Cromwell using the standardized method

cromwell-tools provides a standard method `cromwell_tools.cromwell_auth.CromwellAuth.harmonize_credentials` to authenticate with various Cromwell instances, which is the preferred auth method.

[2]:
```
from cromwell_tools.cromwell_auth import CromwellAuth

# Authenticate with Cromwell using HTTPBasicAuth (username and password)
auth = CromwellAuth.harmonize_credentials(username='username',
                                          password='password',
                                          url='https://cromwell.xxx.broadinstitute.org
↪')

# Authenticate with Cromwell using HTTPBasicAuth (secret JSON file)
auth_2 = CromwellAuth.harmonize_credentials(secrets_file='path/to/secrets_file.json')

# Authenticate with Cromwell using OAuth (service account JSON key file)
auth_3 = CromwellAuth.harmonize_credentials(service_account_key='path/to/service/
↪account/key.json',
                                            url='https://cromwell.caas-prod.
↪broadinstitute.org')
```

(continues on next page)

(continued from previous page)

```
# Authenticate with Cromwell with no Auth
auth_4 = CromwellAuth.harmonize_credentials(url='https://cromwell.caas-prod.
↪broadinstitute.org')
```

2.1.2 2. Authenticate with Cromwell using HTTPBasicAuth (username and password)

[37]:
```
from cromwell_tools.cromwell_auth import CromwellAuth

auth_httpbasic = CromwellAuth.from_user_password(username='username',
                                                 password='password',
                                                 url='https://cromwell.xxx.
↪broadinstitute.org')
```

2.1.3 3. Authenticate with Cromwell using HTTPBasicAuth (secret JSON file)

A secret JSON file should look like the following example with 3 keys:

```
{
    "url": "url",
    "username": "username",
    "password": "password"
}
```

[3]:
```
from cromwell_tools.cromwell_auth import CromwellAuth

auth_httpbasic_file = CromwellAuth.from_secrets_file(secrets_file='path/to/secrets_
↪file.json')
```

2.1.4 4. Authenticate with Cromwell using OAuth (service account JSON key file)

[25]:
```
from cromwell_tools.cromwell_auth import CromwellAuth

auth_oauth = CromwellAuth.from_service_account_key_file(service_account_key='path/to/
↪service/account/key.json',
                                                        url='https://cromwell.caas-
↪prod.broadinstitute.org')
```

2.1.5 5. Authenticate with Cromwell with no Auth

[5]:
```
from cromwell_tools.cromwell_auth import CromwellAuth

auth_no_auth = CromwellAuth.from_no_authentication(url='https://cromwell.caas-prod.
↪broadinstitute.org')
```

2.2 Submit jobs to Cromwell/Cromwell-as-a-Service

[3]:
```
from cromwell_tools import api
from cromwell_tools import utilities
```

[4]:
```
response = api.submit(auth=auth,
                      wdl_file='Examples/hello_world.wdl',
                      inputs_files=['Examples/inputs.json'],
                      dependencies=['Examples/helloworld.wdl'])
```

[5]:
```
response.json()
```

[5]:
```
{'id': '2df17053-57d1-44a4-a922-efea7b29beb0', 'status': 'Submitted'}
```

[7]:
```
api.wait(workflow_ids=[response.json()['id']], auth=auth, poll_interval_seconds=3)
```

```
All workflows succeeded!
```

2.3 Query jobs in Cromwell/Cromwell-as-a-Service

2.3.1 Get the workflows

[8]:
```
from cromwell_tools import api
from cromwell_tools import utilities
```

There are a lot of query keys can be used to filter workflows A complicated query dict could be:

```
custom_query_dict = {
    'label': {
        'cromwell-workflow-id': 'cromwell-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx',
        'project_shortname': 'Name of a project that trigerred this workflow'
    },
    'status': ['Running', 'Succeeded', 'Aborted', 'Submitted', 'On Hold', 'Failed'],
    'id': 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx',
    'additionalQueryResultFields': 'labels',
    'submission': '2018-01-01T00:01:01.410150Z',
    'start': '2018-01-01T01:01:01.410150Z',
    'end': '2018-01-01T02:01:01.410150Z',
    'name': ['SmartSeq2SingleCell', '10xCount'],
    'additionalQueryResultFields': ['labels'],
    'includeSubworkflows': True
}
```

We will just use a very simple example here to query for the workflow we just submitted:

[9]:
```
custom_query_dict = {
    'name': 'HelloWorld',
    'id': response.json()['id']
}

response = api.query(query_dict=custom_query_dict, auth=auth)
```

[10]:
```
response.json()['results']
```

[10]:
```
[{'name': 'HelloWorld',
  'id': '2df17053-57d1-44a4-a922-efea7b29beb0',
  'submission': '2018-11-19T20:58:34.362Z',
  'status': 'Succeeded',
  'end': '2018-11-19T21:00:04.674Z',
  'start': '2018-11-19T20:58:45.442Z'}]
```

2.4 Submit jobs with On Hold in Cromwell/Cromwell-as-a-Service

- Submit a job with "On Hold" status.
- Check the status.
- Release the job.

2.4.1 Submit the job

[11]:
```
from cromwell_tools import api
from cromwell_tools import utilities
```

[13]:
```
response = api.submit(auth=auth,
                      wdl_file='Examples/hello_world.wdl',
                      inputs_files=['Examples/inputs.json'],
                      dependencies=['Examples/helloworld.wdl'],
                      on_hold=True)
```

[14]:
```
response.json()
```

[14]:
```
{'id': 'a996309c-03ae-4c96-bced-63215448b0dd', 'status': 'On Hold'}
```

2.4.2 Check the status of the job

[16]:
```
response = api.status(auth=auth,
                      uuid=response.json()['id'])
response.json()
```

[16]:
```
{'status': 'On Hold', 'id': 'a996309c-03ae-4c96-bced-63215448b0dd'}
```

2.4.3 Release the job

[18]:
```
response = api.release_hold(auth=auth,
                            uuid=response.json()['id'])
response.json()
```

[18]:
```
{'id': 'a996309c-03ae-4c96-bced-63215448b0dd', 'status': 'Submitted'}
```

CHAPTER 3

Cromwell-tools Command Line Interface Quickstart

This notebook will help you get familiar with the Cromwell-tools' CLI (command line interface). After walking through this notebook, you should be able to work with Cromwell engine using the `cromwell-tools` command in your terminal.

3.1 Basic usage

3.1.1 Verify the installation

Once you successfully installed the `cromwell-tools`, you shall be bale to verify if the CLI is added to your available commands by:

[ ]:
```
%%bash

cromwell-tools --version
```

3.1.2 Check all of the available commands and help text

You could check all of the available commands by:

[7]:
```
%%bash

cromwell-tools --help
```

```
usage: cromwell-tools [-h] [-V]
                      {submit,wait,status,abort,release_hold,query,health} ...

positional arguments:
  {submit,wait,status,abort,release_hold,query,health}
                        sub-command help
    submit              submit help
```

(continues on next page)

(continued from previous page)

```
    wait                wait help
    status              status help
    abort               abort help
    release_hold        release_hold help
    query               query help
    health              health help

optional arguments:
  -h, --help            show this help message and exit
  -V, --version         show program's version number and exit
```

You could also check the detailed help message, for instance, to check the help message of `submit`, you could:

[8]:
```
%%bash

cromwell-tools submit --help
```

```
usage: cromwell-tools submit [-h] [--url URL] [--username USERNAME]
                             [--password PASSWORD]
                             [--secrets-file SECRETS_FILE]
                             [--service-account-key SERVICE_ACCOUNT_KEY] -w
                             WDL_FILE -i INPUTS_FILES [INPUTS_FILES ...]
                             [-d DEPENDENCIES [DEPENDENCIES ...]]
                             [-o OPTIONS_FILE] [-l LABEL_FILE]
                             [-c COLLECTION_NAME] [--on-hold ON_HOLD]
                             [--validate-labels VALIDATE_LABELS]

Submit a WDL workflow on Cromwell.

optional arguments:
  -h, --help            show this help message and exit
  --url URL             The URL to the Cromwell server. e.g.
                        "https://cromwell.server.org/"
  --username USERNAME   Cromwell username for HTTPBasicAuth.
  --password PASSWORD   Cromwell password for HTTPBasicAuth.
  --secrets-file SECRETS_FILE
                        Path to the JSON file containing username, password,
                        and url fields.
  --service-account-key SERVICE_ACCOUNT_KEY
                        Path to the JSON key file for authenticating with
                        CaaS.
  -w WDL_FILE, --wdl-file WDL_FILE
                        Path to the workflow source file to submit for
                        execution.
  -i INPUTS_FILES [INPUTS_FILES ...], --inputs-files INPUTS_FILES [INPUTS_FILES ...]
                        Path(s) to the input file(s) containing input data in
                        JSON format, separated by space.
  -d DEPENDENCIES [DEPENDENCIES ...], --deps-file DEPENDENCIES [DEPENDENCIES ...]
                        Path to the Zip file containing dependencies, or a
                        list of raw dependency files to be zipped together
                        separated by space.
  -o OPTIONS_FILE, --options-file OPTIONS_FILE
                        Path to the Cromwell configs JSON file.
  -l LABEL_FILE, --label-file LABEL_FILE
                        Path to the JSON file containing a collection of
                        key/value pairs for workflow labels.
  -c COLLECTION_NAME, --collection-name COLLECTION_NAME
```

(continues on next page)

(continued from previous page)

```
                        Collection in SAM that the workflow should belong to,
                        if use CaaS.
  --on-hold ON_HOLD     Whether to submit the workflow in "On Hold" status.
  --validate-labels VALIDATE_LABELS
                        Whether to validate cromwell labels.
```

3.2 Standard Authentication

Cromwell-tools supports 4 types of authentication when talking to Cromwell:

- Authenticate with Cromwell using HTTPBasicAuth (username and password)
- Authenticate with Cromwell using HTTPBasicAuth (secret JSON file)
- Authenticate with Cromwell using OAuth (service account JSON key file)
- Authenticate with Cromwell with no Auth

As you might have noticed, most of the commands that cromwell-tools provides share a same set of auth-related arguments:

- `--url`
- `--username`
- `--password`
- `--secrets-file`
- `--service-account-key`

You would like to choose the right auth combinations depending on the Cromwell engine you work with.

3.2.1 Authenticate with Cromwell using HTTPBasicAuth (username and password)

Passing only the `--username` and `--password` besides the `--url` indicates that you want to authenticate with a HTTPBasicAuth-protected Cromwell:

[9]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.xxx.broadinstitute.org" \
--username "xxx" \
--password "xxx" \
--wdl "Examples/hello_world.wdl" \
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl"
```

```
{"id":"d3dfa1a0-1134-46dc-9b29-335e645f3be9","status":"Submitted"}
```

3.2.2 Authenticate with Cromwell using HTTPBasicAuth (secret JSON file)

Passing the username and password every time you run the command sounds like a burden. You could save the efforts by storing the HTTPBasic Authentication credentials as well as the cromwell URL into a JSON file (e.g. `secrets.json`) following the format:

```
{
    "url": "url",
    "username": "username",
    "password": "password"
}
```

Now you can pass the secret file path to the cromwell-tools:

[10]:
```
%%bash

cromwell-tools submit \
--secrets-file "secrets.json" \
--wdl "Examples/hello_world.wdl" \
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl"
```

```
{"id":"25dc480d-6755-4b6b-b932-ded2ff634754","status":"Submitted"}
```

3.2.3 Authenticate with Cromwell using OAuth (Google Cloud service account JSON key file)

You could also use cromwell-tools to talk to a Crowmell that is using OAuth2. You will need to pass in the Service Account JSON key file to the `--service-account-key` argument. This file is usually generated by Google Cloud and downloaded from the Cloud console, which should have sufficient permission to talk to te OAuth-enbaled Cromwell.

Specifically, if you are working with Cromwell-as-a-Service (i.e. CaaS), you will need to specify `--collection-name` when submitting a workflow, which indicates a valid collection in SAM that the workflow should belong to.

Here is an example:

[11]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
--collection-name "your-collection" \
--wdl "Examples/hello_world.wdl" \
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl"
```

```
{"id":"a42e54ba-84e4-4305-be90-feeae33b7fc4","status":"Submitted"}
```

3.2.4 Authenticate with Cromwell with no Auth

Sometimes you have to work with a Cromwell that does not have any authentication layer in front of it, no worries, just skip the auth arguments and fill the `--url`!

[9]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.xxx.broadinstitute.org" \
--wdl "Examples/hello_world.wdl" \
```

(continues on next page)

(continued from previous page)

```
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl"
```

```
{"id":"d3dfa1a0-1134-46dc-9b29-335e645f3be9","status":"Submitted"}
```

3.3 Explain the arguments and the `submit` command

3.3.1 Using short optional arguments

You may have noticed that some of the arguments of `submit` have shorter versions, and yes, they are identical to their long counterparts: the following 2 examples are identical!

[11]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
--collection-name "your-collection" \
--wdl "Examples/hello_world.wdl" \
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl"
```

```
{"id":"a42e54ba-84e4-4305-be90-feeae33b7fc4","status":"Submitted"}
```

[11]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
-c "your-collection" \
-w "Examples/hello_world.wdl" \
-i "Examples/inputs.json" \
-d "Examples/helloworld.wdl"
```

```
{"id":"a42e54ba-84e4-4305-be90-feeae33b7fc4","status":"Submitted"}
```

3.3.2 Workflow labels and options

Workflow labels and options files are optional, you can pass the JSON files for your workflow when submitting:

[11]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
--collection-name "your-collection" \
--wdl "Examples/hello_world.wdl" \
--inputs-files "Examples/inputs.json" \
--deps-file "Examples/helloworld.wdl" \
--label-file "Examples/options.json" \
--options-file "Examples/labels.json"
```

```
{"id":"a42e54ba-84e4-4305-be90-feeae33b7fc4","status":"Submitted"}
```

3.3.3 Using URLs as the file paths

So far we have been passing the local (absolute or relative) paths as arguments, but we could also use HTTP/HTTPS paths, crowmell-tools will download and compose them for you:

[12]:
```
%%bash

cromwell-tools submit \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
--collection-name "your-collection" \
--wdl "https://raw.githubusercontent.com/broadinstitute/cromwell-tools/v2.0.0/
↪notebooks/Quickstart/Examples/hello_world.wdl" \
--inputs-files "https://raw.githubusercontent.com/broadinstitute/cromwell-tools/v2.0.
↪0/notebooks/Quickstart/Examples/inputs.json" \
--deps-file "https://raw.githubusercontent.com/broadinstitute/cromwell-tools/v2.0.0/
↪notebooks/Quickstart/Examples/helloworld.wdl"
```

```
{"id":"13f7577a-c496-4770-8f3e-0f085f4edac0","status":"Submitted"}
```

3.4 The `wait` command

The `wait` command is a special command that helps polling and monitoring the workflow(s) in the Cromwell engine. You can use it to keep track of a list of workflows(they are NOT necessarily to be submitted by cromwell-tools!). By default `wait` will print verbose information while polling, but you can configure its behavior using the following arguemnts and flag(s):

- `--timeout-minutes`
- `--poll-interval-seconds`
- `--silent`

Note: failure of one of the workflows will cause the whole polling process to be terminated, so please only poll one workflow if you are not sure if the workflow can succeed.

[14]:
```
%%bash

cromwell-tools wait \
--url "https://cromwell.caas-prod.broadinstitute.org" \
--service-account-key "/path/to/your/service-account-json-key.json" \
--poll-interval-seconds 10 \
a42e54ba-84e4-4305-be90-feeae33b7fc4 \
13f7577a-c496-4770-8f3e-0f085f4edac0 \
d9d8dc18-d462-46f6-a39d-b68b20dfb5ab
```

```
--- polling from cromwell ---
Workflow a42e54ba-84e4-4305-be90-feeae33b7fc4 returned status Succeeded
Workflow 13f7577a-c496-4770-8f3e-0f085f4edac0 returned status Succeeded
All workflows succeeded!
```

3.5 Other commands

You can check the available arugments of other commands by using `cromwell-tools COMMAND -h`, other commands are just simple mappings of the corresponding Cromwell endpoints. Note that the `query` command is not yet

implemented in the CLI, please use the Python API client instead!

CHAPTER 4

API Documentation

Comprehensive reference material for most of the public API exposed by Cromwell-tools:

4.1 cromwell_tools.cromwell_api

TODO: add some module docs TODO: once switched to support only Py3.7+, replace all 'cls' type annotations with the actual Types, rather than using the strings. This in Py3.6(-) is limited by the lack of Postponed Evaluation of Annotations, see: https://www.python.org/dev/peps/pep-0563/

class cromwell_tools.cromwell_api.**CromwellAPI**

Contains a set of classmethods that implement interfaces to cromwell REST API endpoints.

classmethod abort(*uuid: str*, *auth: cromwell_tools.cromwell_auth.CromwellAuth*, *raise_for_status: bool = False*) → requests.models.Response

Request Cromwell to abort a running workflow by UUID.

Parameters

- **uuid** – A Cromwell workflow UUID, which is the workflow identifier.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises requests.exceptions.HTTPError – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod health(*auth: cromwell_tools.cromwell_auth.CromwellAuth*, *raise_for_status: bool = False*) → requests.models.Response

Return the current health status of any monitored subsystems of the Cromwell Server.

Parameters

- **auth** – authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod metadata(*uuid: str, auth: cromwell_tools.cromwell_auth.CromwellAuth, includeKey: Union[List[str], str] = None, excludeKey: Union[List[str], str] = None, expandSubWorkflows: bool = False, raise_for_status: bool = False*) → requests.models.Response

Retrieve the workflow and call-level metadata for a specified workflow by UUID.

Parameters

- **uuid** – A Cromwell workflow UUID, which is the workflow identifier.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **includeKey** – When specified key(s) to include from the metadata. Matches any key starting with the value. May not be used with excludeKey.
- **excludeKey** – When specified key(s) to exclude from the metadata. Matches any key starting with the value. May not be used with includeKey.
- **expandSubWorkflows** – When true, metadata for sub workflows will be fetched and inserted automatically in the metadata response.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod patch_labels(*uuid: str, labels: Dict[str, str], auth: cromwell_tools.cromwell_auth.CromwellAuth, raise_for_status: bool = False*) → requests.models.Response

Add new labels or patch existing labels for an existing workflow.

Parameters

- **uuid** – A Cromwell workflow UUID, which is the workflow identifier.
- **labels** – A dictionary representing the label key-value pairs.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod query(*query_dict: Dict[str, Union[str, List[str], Dict[str, str], bool]], auth: cromwell_tools.cromwell_auth.CromwellAuth, raise_for_status: bool = False*) → requests.models.Response

Query for workflows.

TODO: Given that Cromwell-as-a-Service blocks a set of features that are available in Cromwell, e.g. 'labelor', for security concerns, the first iteration of this API doesn't come up with the advanced query keys of the Cromwell except a set of necessary ones. However, we need to implement this for completeness and keep an eye on the compatibility between CaaS and Cromwell.

All of the query keys will be used in an OR manner, except the keys within *labels*, which are defined in an AND relation. For instance, [{'status': 'Succeeded'}, {'status': 'Failed'}] will give you all of the workflows that in either *Succeeded* or *Failed* statuses.

Parameters

- **query_dict** – A dictionary representing the query key-value paris. The keys should be accepted by the Cromwell or they will get ignored. The values could be str, list or dict.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod release_hold(*uuid: str*, *auth: cromwell_tools.cromwell_auth.CromwellAuth*, *raise_for_status: bool = False*) → requests.models.Response

Request Cromwell to release the hold on a workflow.

It will switch the status of a workflow from 'On Hold' to 'Submitted' so it can be picked for running. For a workflow that was not submitted with *workflowOnHold = true*, Cromwell will throw an error.

Parameters

- **uuid** – A Cromwell workflow UUID, which is the workflow identifier. The workflow is expected to have *On Hold* status.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod status(*uuid: str*, *auth: cromwell_tools.cromwell_auth.CromwellAuth*, *raise_for_status: bool = False*) → requests.models.Response

Retrieves the current state for a workflow by UUID.

Parameters

- **uuid** – A Cromwell workflow UUID, which is the workflow identifier.
- **auth** – The authentication class holding headers or auth information to a Cromwell server.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod submit(*auth: cromwell_tools.cromwell_auth.CromwellAuth, wdl_file: Union[str, _io.BytesIO], inputs_files: Union[List[Union[str, _io.BytesIO]], str, _io.BytesIO] = None, options_file: Union[str, _io.BytesIO] = None, dependencies: Union[str, List[str], _io.BytesIO] = None, label_file: Union[str, _io.BytesIO] = None, collection_name: str = None, on_hold: bool = False, validate_labels: bool = False, raise_for_status: bool = False*) → requests.models.Response

Submits a workflow to Cromwell.

Parameters

- **auth** – authentication class holding auth information to a Cromwell server.
- **wdl_file** – The workflow source file to submit for execution. Could be either the path to the file (str) or the file content in io.BytesIO.
- **inputs_files** – The input data in JSON format. Could be either the path to the file (str) or the file content in io.BytesIO. This could also be a list of unlimited input file paths/contents, each of them should have a type of Union[str, io.BytesIO].
- **options_file** – The Cromwell options file for workflows. Could be either the path to the file (str) or the file content in io.BytesIO.
- **dependencies** – Workflow dependency files. Could be the path to the zipped file (str) containing dependencies, a list of paths(List[str]) to all dependency files to be zipped or a zipped file in io.BytesIO.
- **label_file** – A collection of key/value pairs for workflow labels in JSON format, could be either the path to the JSON file (str) or the file content in io.BytesIO.
- **collection_name** – Collection in SAM that the workflow should belong to, if use CaaS.
- **on_hold** – Whether to submit the workflow in "On Hold" status.
- **validate_labels** – If True, validate cromwell labels.
- **raise_for_status** – Whether to check and raise for status based on the response.

Raises `requests.exceptions.HTTPError` – This will be raised when raise_for_status is True and Cromwell returns a response that satisfies 400 <= response.status_code < 600.

Returns HTTP response from Cromwell.

classmethod wait(*workflow_ids: List[str], auth: cromwell_tools.cromwell_auth.CromwellAuth, timeout_minutes: int = 120, poll_interval_seconds: int = 30, verbose: bool = True*) → None

Wait until cromwell returns successfully for each provided workflow

Given a list of workflow ids, wait until cromwell returns successfully for each status, or one of the workflows fails or is aborted.

Parameters

- **workflow_ids** – A list of workflow ids to wait for terminal status.
- **timeout_minutes** – Maximum number of minutes to wait.
- **auth** – Authentication class holding headers or auth information to a Cromwell server.
- **poll_interval_seconds** – Number of seconds between checks for workflow completion.
- **verbose** – If True, report to stdout when all workflows succeed.

4.2 cromwell_tools.cromwell_auth

TODO: add some module docs TODO: once switched to support only Py3.7+, replace all 'cls' type annotations with the actual Types, rather than using the strings. This in Py3.6(-) is limited by the lack of Postponed Evaluation of Annotations, see: https://www.python.org/dev/peps/pep-0563/

4.3 cromwell_tools.utilities

cromwell_tools.utilities.**compose_oauth_options_for_jes_backend_cromwell**(*auth: cromwell_tools.cromwell_a cromwell_options_file: _io.BytesIO = None, execution_bucket: str = None*) → _io.BytesIO

Append special options that are required by JES(Google Job Execution Service) backend Cromwell.

This helper function will append special options that are required by JES(Google Job Execution Service) backend Cromwell/Cromwell-as-a-Service to the default workflow options. Note: These options only work with Cromwell instances that use the Google Cloud Backend and allow user-service-account authentication.

Parameters

- **auth** – authentication class holding auth information to a Cromwell server.
- **cromwell_options_file** – Optional, contents of the options for a workflow in BytesIO format. if not specified, this function will create an empty option stream and add the necessary keys to it.
- **execution_bucket** – Optional, the Google CLoud Bucket that Cromwell will use to output execution results and store temporary scripts. If not specified, it will use 'gs://{google_project}-cromwell-execution/caas-cromwell-executions' by default.

Returns BytesIO object of the updated workflow options with the required auth fields.

Return type options_stream

cromwell_tools.utilities.**download**(*url: str*) → Union[str, bytes]

Reads the contents located at the url into memory and returns them.

Urls starting with http are fetched with an http request. All others are assumed to be local file paths and read from the local file system.

Parameters **url** – The url to the content to be downloaded, or the path to the local file.

Returns Downloaded content in str or bytes format.

Raises TypeError – If the url is not a str type.

cromwell_tools.utilities.**download_http**(*url: str*) → Union[str, bytes]

Makes an http request for the contents at the given url and returns the response body.

Parameters **url** – The url to the content to be downloaded.

Returns Content returned from the server. Will be *str* in Python2 and bytes in *Python3*.

Return type response_str

cromwell_tools.utilities.**download_to_map**(*urls: List[str]*) → Dict[str, Any]

Reads contents from each url into memory and returns a map of urls to their contents.

Parameters **urls** – A list of urls to the contents to be downloaded.

Returns A dict representing the mapping from url to the downloaded contents in-memory.

Return type url_to_contents

cromwell_tools.utilities.**make_zip_in_memory**(*url_to_contents: Dict[str, Any]*) → _io.BytesIO

Given a map of urls and their contents, returns an in-memory zip file containing each file.

For each url, the part after the last slash is used as the file name when writing to the zip archive.

Parameters **url_to_contents** – A dict representing the mapping from url to the downloaded contents in-memory.

Returns Zipped files content in bytes.

Return type bytes_buf

cromwell_tools.utilities.**prepare_workflow_manifest**(*wdl_file: Union[str, _io.BytesIO], inputs_files: Union[List[Union[str, _io.BytesIO]], str, _io.BytesIO] = None, options_file: Union[str, _io.BytesIO] = None, dependencies: Union[str, List[str], _io.BytesIO] = None, label_file: Union[str, _io.BytesIO] = None, collection_name: str = None, on_hold: bool = False*) → Dict[str, Union[_io.BytesIO, str]]

Prepare the submission manifest for a workflow submission.

Parameters

- **wdl_file** – The workflow source file to submit for execution. Could be either the path to the file (str) or the file content in io.BytesIO.
- **inputs_files** – The input data in JSON format. Could be either the path to the file (str) or the file content in io.BytesIO. This could also be a list of unlimited input file paths/contents, each of them should have a type of Union[str, io.BytesIO].
- **options_file** – The Cromwell options file for workflows. Could be either the path to the file (str) or the file content in io.BytesIO.
- **dependencies** – Workflow dependency files. Could be the path to the zipped file (str) containing dependencies, a list of paths(List[str]) to all dependency files to be zipped or a zipped file in io.BytesIO.
- **label_file** – A collection of key/value pairs for workflow labels in JSON format, could be either the path to the JSON file (str) or the file content in io.BytesIO.
- **collection_name** – Collection in SAM that the workflow should belong to, if use CaaS.
- **on_hold** – Whether to submit the workflow in "On Hold" status.

Returns A dictionary representing the workflow manifest ready for workflow submission.

Return type workflow_manifest

Raises ValueError – If a str ing of path to the dependencies is given but not endswith ".zip".

cromwell_tools.utilities.**read_local_file**(*path: str*) → Union[str, bytes]

Reads the file contents and returns them.

Parameters **path** – Path to the local file to be loaded.

Returns The loaded content. bytes in Python3 and str in Python2.

Return type contents

cromwell_tools.utilities.**validate_cromwell_label**(*label_object: Union[str, _io.BytesIO, bytes, Dict[str, str]]*) → None

Check if the label object is valid for Cromwell.

Note: this function as well as the global variables _CROMWELL_LABEL_LENGTH, _CROMWELL_LABEL_KEY_REGEX and _CROMWELL_LABEL_VALUE_REGEX are implemented based on the Cromwell's documentation: https://cromwell.readthedocs.io/en/develop/Labels/ and the Cromwell's code base: https://github.com/broadinstitute/cromwell/blob/master/core/src/main/scala/cromwell/core/labels/Label.scala#L16 Both the docs and the code base of Cromwell could possibly change in the future, please update this checker on demand.

Parameters **label_object** – A dictionary or a key-value object string defines a Cromwell label.

Raises ValueError – This validator will raise an exception if the label_object is invalid as a Cromwell label.

4.4 cromwell_tools.cli

class cromwell_tools.cli.**DefaultHelpParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

add_argument(*dest*, ..., *name=value*, ...)

add_argument(option_string, option_string, . . . , name=value, . . .)

error(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

Python Module Index

C

Index