Emu Documentation

Release 0.10.0

Carsten Ehbrecht

Apr 17, 2019

CONTENTS

Emu (the bird) *Emus are curious birds who are known to follow and watch other animals and humans. Emus do not sleep continuously at night but in several short stints sitting down. [..].* (Wikipedia).

Emu is a Python package with some test proccess for Web Processing Services (WPS). Currently it is using the PyWPS 4.x server.

Full documentation is available on ReadTheDocs or in the docs directory.

CHAPTER

ONE

INSTALLATION

1.1 Install from Conda

Install the `emu` Conda package:

```
$ conda install -c birdhouse -c conda-forge emu
$ emu --help
```

1.2 Install from GitHub

Check out code from the Emu GitHub repo and start the installation:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ conda env create -f environment.yml
$ source activate emu
$ python setup.py develop
```

1.2.1 . . . or do it the lazy way

The previous installation instructions assume you have Anaconda installed. We provide also a `Makefile` to run this installation without additional steps:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean    # cleans up a previous Conda environment
$ make install  # installs Conda if necessary and runs the above installation steps
```

1.3 Start Emu PyWPS service

After successful installation you can start the service using the `emu` command-line.

```
$ emu start --help # show help
$ emu start       # start service with default configuration

OR

$ emu start --daemon # start service as daemon
loading configuration
forked process id: 42
```

The deployed WPS service is by default available on:

http://localhost:5000/wps?service=WPS&version=1.0.0&request=GetCapabilities.

Note: Remember the process ID (PID) so you can stop the service with `kill PID`.

You can find which process uses a given port using the following command (here for port 5000):

```
$ netstat -nlp | grep :5000
```

Check the log files for errors:

```
$ tail -f  pywps.log
```

1.3.1 . . . or do it the lazy way

You can also use the `Makefile` to start and stop the service:

```
$ make start
$ make status
$ tail -f pywps.log
$ make stop
```

1.4 Run Emu as Docker container

You can also run Emu as a Docker container, see the *Tutorial*.

1.5 Use Ansible to deploy Emu on your System

Use the Ansible playbook for PyWPS to deploy Emu on your system.

CHAPTER

TWO

CONFIGURATION

2.1 Command-line options

You can overwrite the default PyWPS configuration by using command-line options. See the Emu help which options are available:

```
$ emu start --help
--hostname HOSTNAME        hostname in PyWPS configuration.
--port PORT                port in PyWPS configuration.
```

Start service with different hostname and port:

```
$ emu start --hostname localhost --port 5001
```

2.2 Use a custom configuration file

You can overwrite the default PyWPS configuration by providing your own PyWPS configuration file (just modifiy the options you want to change). Use one of the existing `sample-*.cfg` files as example and copy them to `etc/custom.cfg`.

For example change the hostname (*demo.org*) and logging level:

```
$ cd emu
$ vim etc/custom.cfg
$ cat etc/custom.cfg
[server]
url = http://demo.org:5000/wps
outputurl = http://demo.org:5000/outputs

[logging]
level = DEBUG
```

Start the service with your custom configuration:

```
# start the service with this configuration
$ emu start -c etc/custom.cfg
```

CHAPTER

THREE

DEVELOPER GUIDE

Warning: To create new processes look at examples in Emu.

3.1 Building the docs

First install dependencies for the documentation:

```
$ make bootstrap_dev
$ make docs
```

3.2 Running tests

Run tests using pytest.

First activate the `emu` Conda environment and install `pytest`.

```
$ source activate emu
$ conda install pytest flake8  # if not already installed
```

Run quick tests (skip slow and online):

```
$ pytest -m 'not slow and not online'"
```

Run all tests:

```
$ pytest
```

Check pep8:

```
$ flake8
```

3.3 Run tests the lazy way

Do the same as above using the `Makefile`.

```
$ make test
$ make testall
$ make pep8
```

3.4 Prepare a release

Update the Conda specification file to build identical environments on a specific OS.

Note: You should run this on your target OS, in our case Linux.

```
$ make clean
$ make install
$ make spec
```

3.5 Bump a new version

Make a new version of Emu in the following steps:

- Make sure everything is commit to GitHub.
- Update `CHANGES.rst` with the next version.
- Dry Run: `bumpversion --dry-run --verbose --new-version 0.8.1 patch`
- Do it: `bumpversion --new-version 0.8.1 patch`
- ... or: `bumpversion --new-version 0.9.0 minor`
- Push it: `git push`
- Push tag: `git push --tags`

See the bumpversion documentation for details.

CHAPTER

FOUR

TUTORIALS

4.1 Tutorial: using Docker

Emu WPS is available as docker image. You can download the docker image from DockerHub or build it from the provided Dockerfile.

Start the container with the following command:

```
$ docker run -d -p 5000:5000 --name=emu birdhouse/emu
```

It is using the port 5000 for the PyWPS service and also to access the WPS outputs.

You can map the container port also to another port on your machine, for example: `-p 8094:5000` (your machine port=8094, container port=5000).

Check the docker logs:

```
$ docker logs emu
```

Show running docker containers:

```
$ docker ps
```

Run a GetCapabilites WPS request:

> http://localhost:5000/wps?service=WPS&version=1.0.0&request=getcapabilities

Run DescribeProcess WPS request for *Hello*:

> http://localhost:5000/wps?service=WPS&version=1.0.0&request=describeprocess&identifier=hello

Execute *Hello* process with you user name:

> http://localhost:5000/wps?service=WPS&version=1.0.0&request=execute&identifier=hello&DataInputs=name=Pingu

Install *Birdy* WPS command line tool from Conda:

```
$ conda install -c birdhouse birdhouse-birdy
```

Use Birdy to access Emu WPS service:

```
$ export WPS_SERVICE=http://localhost:5000/wps
$ birdy -h
$ birdy hello -h
$ birdy hello --name Pingu
```

Stop and remove docker container:

```
$ docker stop emu
```

4.1.1 Using docker-compose

Use docker-compose (you need a recent version > 1.7) to start the container:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ docker-compose up -d
$ docker-compose logs emu
```

Execute `tail` command in the running container to see the logs:

```
$ docker ps   # get the container name
NAMES
emu_emu_1
$ docker exec -it emu_emu_1 tail -f /opt/wps/pywps.log
```

You can customize the `docker-compose.yml` file. See the docker-compose documentation.

Stop the container with:

```
$ docker-compose down
```

4.1.2 Build image using docker-compose

You can build locally a new docker image from the Dockerfile by running docker-compose:

```
$ docker-compose build
```

4.2 Tutorial: using postgres database

You can use a postgres database for PyWPS, the default is sqlite. PyWPS is using SQLAlchemy, see the PYWPS documentation for details.

First run the Emu default installation:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean install
```

The default installation is using sqlite. We now need a postgres database. If you don't have one yet you can use a postgres docker container.

```
$ docker pull postgres
$ docker run --name postgres -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres
```

The postgres database is now available on default port 5432.

SQLAlchemy needs the psycopg2 postgres adapter. You can install it via Conda into the `emu` environment.

```
$ conda install -n emu psycopg2
```

The SQLAlchemy connection string for this database is:

```
# postgresql+psycopg2://user:password@host:port/dbname
postgresql+psycopg2://postgres:postgres@localhost:5432/postgres
```

Configure this connection string in `etc/custom.cfg`, `logging` section, `database` option:

```
$ vim etc/custom.cfg
[logging]
level = INFO
database = postgresql+psycopg2://postgres:postgres@localhost:5432/postgres
```

Start the emu service:

```
$ emu start -c etc/custom.cfg
```

Your Emu WPS service should be available at the following URL:

```
$ firefox http://localhost:5000/wps?request=GetCapabilities&service=WPS
```

CHAPTER

FIVE

PROCESSES

5.1 Say Hello

class emu.processes.wps_say_hello.**SayHello**

hello Say Hello (v1.5)

Just says a friendly Hello. Returns a literal string output with Hello plus the inputed name.

Parameters **name** (*string*) – Please enter your name.

Returns **output** – A friendly Hello from us.

Return type string

References

- User Guide
- PyWPS Demo

5.2 Sleep

class emu.processes.wps_sleep.**Sleep**

sleep Sleep Process (v1.0)

Testing a long running process, in the sleep. This process will sleep for a given delay or 10 seconds if not a valid value.

Parameters **delay** (*float*) – Delay between every update

Returns **sleep_output** – Sleep Output

Return type string

References

- User Guide
- PyWPS Demo

5.3 Wordcounter

class emu.processes.wps_wordcounter.**WordCounter**

wordcounter Word Counter (v1.0)

Counts words in a given text.

Parameters **text** (*text/plain*) – URL pointing to a text document, for example "Alice in Wonderland": http://www.gutenberg.org/cache/epub/19033/pg19033.txt

Returns **output** – Word counter result

Return type *application/json*

References

- User Guide

Counts occurrences of all words in a document.

5.4 Chomsky

class emu.processes.wps_chomsky.**Chomsky**

chomsky Chomsky text generator (v1.0)

Generates a random chomsky text

Parameters **times** (*integer*) – Generates a random chomsky text.

Returns **output** – Chomsky text

Return type *text/plain*

Generates a random chomsky text: http://code.activestate.com/recipes/440546-chomsky-random-text-generator/

CHOMSKY is an aid to writing linguistic papers in the style of the great master. It is based on selected phrases taken from actual books and articles written by Noam Chomsky. Upon request, it assembles the phrases in the elegant stylistic patterns that Chomsky is noted for. To generate n sentences of linguistic wisdom, type:

- (CHOMSKY n) – for example

- (CHOMSKY 5) generates half a screen of linguistic truth.

5.5 NCMeta

class emu.processes.wps_ncmeta.**NCMeta**

ncmeta Return NetCDF Metadata (v4)

Return metadata from a netCDF dataset, either on file or an OpenDAP service.

Parameters

- **dataset** (*application/x-netcdf*, optional) – http://test.opendap.org:80/opendap/netcdf/examples/sresa1b_ncar_ccsm3_0_run1_200001.nc.nc4
- **dataset_opendap** (*application/x-ogc-dods*, optional) – http://test.opendap.org:80/opendap/netcdf/examples/sresa1b_ncar_ccsm3_0_run1_200001.nc

Returns **output** – Metadata

Return type *text/plain*

References

- User Guide

Returns metadata of a NetCDF file or OpenDAP resource.

5.6 ShowError

class emu.processes.wps_error.**ShowError**

show_error Show a WPS Error (v1.0)

This process will fail intentionally with a friendly WPS error message.

Parameters

- **message** (*string*) – Enter an error message that will be returned.
- **nice** (*boolean*) – Be nice and show a friendly error message. Default: true

References

- PyWPS
- Birdhouse
- User Guide

An example request:

```
http://localhost:5000/wps?
    version=1.0.0&
    service=wps&
    request=Execute&
    identifier=show_error&
    DataInputs=message=bad-day;nice=true
```

5.7 SimpleDryRun

class emu.processes.wps_dry_run.**SimpleDryRun**
simple_dry_run Simple Dry Run (v1.0)

A dummy download as simple dry-run example.

Parameters

- **dry_run** (*boolean*) – Dry run mode. Default false
- **count** (*{'None'}*) – How many files do you want to download? The limit is 10

Returns **output** – Output response

Return type string

References

- User Guide

5.8 MultipleOutputs

class emu.processes.wps_multiple_outputs.**MultipleOutputs**
multiple_outputs Multiple Outputs (v1.1)

Produces multiple files and returns a document with references to these files.

Parameters **count** (*{'None'}*) – The number of generated output files.

Returns

- **output** (*application/metalink+xml; version=3.0*) – Testing metalink v3 output
- **output_meta4** (*application/metalink+xml; version=4.0*) – Testing metalink v4 output

References

- User Guide

5.9 InOut

class emu.processes.wps_inout.**InOut**
inout In and Out (v1.0)

Testing all WPS input and output parameters.

Parameters

- **string** (*string*) – Enter a simple string.
- **int** (*{'1', '2', '3', '5', '7', '11'}*) – Choose an integer number from allowed values.
- **float** (*float*) – Enter a float number.

- **boolean** (*boolean*) – Make your choice :)
- **angle** (*angle*) – Enter an angle [0, 360] :)
- **time** (*time*) – Enter a time like 12:00:00
- **date** (*date*) – Enter a date like 2012-05-01
- **datetime** (*dateTime*) – Enter a datetime like 2016-09-02T12:00:00Z
- **string_choice** (*{'rock', 'paper', 'scissor'}*) – Choose one item form list.
- **string_multiple_choice** (*{'sitting duck', 'flying goose', 'happy pinguin', 'gentle albatros'}, optional*) – Choose one or two items from list.
- **int_range** (*{'None', 'None'}*) – Choose number from range: 1-10 (step 1), 100-200 (step 10) (PyWPS Docs, AllowedValue Example)
- **any_value** (*{'None'}*) – Enter any value. (PyWPS Docs)
- **ref_value** (*{'None'}*) – Choose a referenced value (PyWPS Docs)
- **text** (*text/plain*, optional) – Enter a URL pointing to a text document (optional) (Info)
- **dataset** (*application/x-netcdf*, optional) – Enter a URL pointing to a NetCDF file (optional) (NetCDF Format)

Returns

- **string** (*string*) – String
- **int** (*integer*) – Integer
- **float** (*float*) – Float
- **boolean** (*boolean*) – Boolean
- **angle** (*angle*) – Angle
- **time** (*time*) – Time
- **date** (*date*) – Date
- **datetime** (*dateTime*) – DateTime
- **string_choice** (*string*) – String Choice
- **string_multiple_choice** (*string*) – String Multiple Choice
- **int_range** (*integer*) – Integer Range
- **any_value** (*string*) – Any Value
- **ref_value** (*string*) – Referenced Value
- **text** (*text/plain*) – Copy of input text file.
- **dataset** (*application/x-netcdf*, *text/plain*) – Copy of input netcdf file.
- **bbox** (*[epsg:4326]*) – Bounding Box

References

- Birdhouse
- User Guide

TODO: add literal input with value range[(0,100)] ... see pywps doc

CHAPTER

SIX

CHANGES

6.1 0.10.0 (2019-04-17)

This is the San Francisco release.

Changes:

- Added example for Metalink as process output response (#84).
- Updated *inout* process with examples for AllowedValue, AnyValue and ValuesReference (#88, #85, #82).
- Using pywps *ProcessError* exception (#86)
- Added example process for *dry-run* usage (#83).
- Updated to latest cookiecutter template (#87).

6.2 0.9.1 (2018-12-04)

This is the Washington release.

Changes:

- Using *emu.__version__.py* in *setup.py* (#67 and #68).
- Added Angle data type (#65).
- Added test for wps_multiple_outputs (#60).

6.3 0.9.0 (2018-09-06)

This is the release for FOSS4G in Dar Es Salaam.

Changes:

- Enabled Conda support on ReadTheDocs (#40).
- Added `ncmeta` process with PyWPS OpenDAP support (#54).
- Added `output_formats` process to test NetCDF and JSON output formats (#42).
- Numerous fixes.

6.4 0.8.0 (2018-06-06)

This is the first release without Buildout. Is has a command-line interface `emu` to start/stop the PyWPS service using Werkzeug.

Changes:

- Removed Buildout configuration and relying only on Conda and Werkzeug.
- Support for Python 2.7/3.x (#6).
- Added templates for issues, PRs and contribution guide (#15).
- Use bumpversion (#36).
- Makefile with clean, install, start, stop and status targets (#35).
- Use staticmethod for PyWPS handler (#33).
- Using Click CLI to start/stop PyWPS service (#31).
- Using jinja template for pywps configuration (#29)

6.5 0.7.0 (2018-05-17)

This is the last release using Buildout for deployment. This release will be maintained on the 0.7.x branch.

Issues solved:

- Fix async mode in demo service (#26)
- Fix WSGI app initialisation (#17)
- Use six for Python 2/3 compatibility (#20)
- Reference Readme in Sphinx docs (#22)
- Move `tests/` folder to top-level directory (#21)
- Updated gunicorn 19.x (#19)

6.6 0.6.3 (2018-04-04)

Issues solved:

- Clean up directory structure and files (#13)
- clean up of buildout and docker (#14)

Others:

- Updated buildout conda recipe 0.4.0.

6.7 0.6.2 (2018-02-07)

- using pywps autodoc extension for Sphinx.
- added badges for chat, docs and license.

- fixed pywps output format.

6.8 0.6.1 (2018-01-10)

- hello process: using keywords in metadata for description.
- updated dependencies.
- updated demo service.

6.9 0.6.0 (2017-08-16)

- added esgf_demo process.
- added psycopg2 conda package for postgres
- added dill and drmaa package for scheduler.
- updated pywps recipe 0.9.2.
- added demo module.

6.10 0.5.3 (2017-05-18)

- updated pywps recipe 0.9.0.
- added wsgi application.

6.11 0.5.2 (2017-05-08)

- updated pywps recipe 0.8.8.
- updated supervisor recipe 0.3.6.
- updated zc.buildout 2.7.1
- update Makefile.
- enabled bbox parameter.
- using Metadata role attribute.
- updated say_hello process.
- added multiple_outputs process.
- updated conda recipe 0.3.6.

6.12 0.5.1 (2017-01-04)

- added processes: nap, binaryoperator, show_error.
- updated pywps recipe 0.8.2.

- updated pywps 4.0.0.
- fixed wps_caps test.
- using __version__ constant.
- fixed install on ubuntu 16.04: updated conda env (lxml, icu).

6.13 0.5.0 (2016-12-07)

- using pywps-4.
- updated all processes to pywps-4.
- updated Dockerfile.
- using docker-compose with environment from .env.

6.14 0.4.1 (2016-10-20)

- fixed docs and comments.
- updated recipes, using conda-offline.

6.15 0.4.0 (2016-07-11)

- using new buildout recipes.
- using conda environment.yml

6.16 0.3.2 (2016-07-11)

- using pytest.

6.17 0.3.1 (2016-03-23)

- added bbox process.

6.18 0.3.0 (2016-01-21)

- removed malleefowl dependency.

6.19 0.2.2 (2016-01-07)

- using pywps WPSProcess class.
- zonal-mean process added.
- docker-compose added.
- updated Dockerfile.
- updated pywps, supervisor and docker recipe.
- log to stderr/supervisor.

6.20 0.2.1 (2015-02-25)

- updated docs and makefile.

6.21 0.2.0 (2015-02-24)

- Now possible to use shared anaconda for installation.

6.22 0.1.2 (2014-11-24)

- Using buildout 2.x.

6.23 0.1.1 (2014-11-11)

- Using Makefile from birdhousebuilder.bootstrap to install and start application.

6.24 0.1.0 (2014-09-04)

Initial Paris Release

INDEX