Emu Documentation

Release 0.8.0

Birdhouse

Jun 06, 2018

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 1

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`.

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. Follow the example for Emu given in the playbook.

1.6 Building the docs

First install dependencies for the documentation:

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

CHAPTER 2

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 3

Developer Guide

3.1 Running tests

Run tests using pytest.

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

```
$ cd emu
$ 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
```

3.2 Run tests the lazy way

Do the same as above using the `Makefile`.

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

3.3 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 --tags`

See the bumpversion documentation for details.

CHAPTER 4

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 5

Sphinx AutoAPI Index

This page is the top-level of your generated API documentation. Below is a list of all items that are documented here.

5.1 wsgi

5.1.1 Module Contents

wsgi.**create_app**(*cfgfiles=None*)

5.2 cli

5.2.1 Module Contents

cli.**write_user_config**(***kwargs*)

cli.**get_host**()

cli.**run_process_action**(*action=None*)
Run an action with psutil on current process and return a status message.

cli.**_run**(*application*, *bind_host=None*, *daemon=False*)

cli.**cli**()
Command line to start/stop a PyWPS service.

Do not use this service in a production environment. It's intended to be running in a test environment only! For more documentation, visit http://pywps.org/doc

cli.**status**()
Show status of PyWPS service

cli.**stop**()
Stop PyWPS service

cli.**start**(*config*, *bind_host*, *daemon*, *hostname*, *port*, *maxsingleinputsize*, *maxprocesses*, *parallelprocesses*, *log_level*, *log_file*, *database*)
Start PyWPS service. This service is by default available at http://localhost:5000/wps

5.3 processes

5.3.1 Submodules

processes.wps_bbox

Module Contents

class processes.wps_bbox.**Box**

_handler(*response*)

processes.wps_binaryoperator

Module Contents

class processes.wps_binaryoperator.**BinaryOperator**

_handler(*response*)

processes.wps_chomsky

Module Contents

class processes.wps_chomsky.**Chomsky**

Notes

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.

_handler(*response*)

processes.wps_dummy

DummyProcess to check the WPS structure

Author: Jorge de Jesus (jorge.jesus@gmail.com) as suggested by Kor de Jong

Module Contents

class processes.wps_dummy.**Dummy**

_handler(*response*)

processes.wps_error

Module Contents

class processes.wps_error.**ShowError**

_handler(*response*)

processes.wps_esgf

Module Contents

class processes.wps_esgf.**ESGFDemo**

_handler(*response*)

processes.wps_inout

Module Contents

class processes.wps_inout.**InOut**

This process defines several types of literal type of in- and outputs.

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

_handler(*response*)

processes.wps_multiple_outputs

Module Contents

class processes.wps_multiple_outputs.**MultipleOutputs**

_handler(*response*)

processes.wps_nap

Module Contents

class processes.wps_nap.**Nap**

_handler(*response*)

processes.wps_say_hello

Module Contents

class processes.wps_say_hello.**SayHello**

_handler(*response*)

processes.wps_sleep

Module Contents

class processes.wps_sleep.**Sleep**

_handler(*response*)

processes.wps_ultimate_question

Module Contents

class processes.wps_ultimate_question.**UltimateQuestion**

_handler(*response*)

processes.wps_wordcounter

Module Contents

class processes.wps_wordcounter.**WordCounter**

Notes

Counts ocurrences of all words in a document.

_handler(*response*)

Python Module Index

c

p

w

Index

Symbols

B

C

D

E

G

I

M

N

P

R

S

U

W