copernicus-wps-demo Documentation

Release 0.3.0

Carsten Ehbrecht

Aug 16, 2018

Contents:

Demo with WPS processes for Copernicus.

- Free software: Apache Software License 2.0
- Documentation: https://copernicus-wps-demo.readthedocs.io.

CHAPTER 1

Coperniucs Climate Data Services

- Climate Data Store
- C3S MAGIC
- CP4CDS Quality Control

CHAPTER 2

Credits

This package was created with Cookiecutter and the bird-house/cookiecutter-birdhouse project template.

2.1 Installation

2.1.1 Install from Anaconda

2.1.2 Install from GitHub

Check out code from the copernicus-wps-demo GitHub repo and start the installation:

```
$ git clone https://github.com/cp4cds/copernicus-wps-demo.git
$ cd copernicus-wps-demo
$ conda env create -f environment.yml
$ source activate copernicus
$ python setup.py develop
```

. . . 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/cp4cds/copernicus-wps-demo.git
$ cd copernicus-wps-demo
$ make clean    # cleans up a previous Conda environment
$ make install  # installs Conda if necessary and runs the above installation steps
```

2.1.3 Start copernicus-wps-demo PyWPS service

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

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

OR

$ copernicus 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
```

... 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
```

2.1.4 Run copernicus-wps-demo as Docker container

You can also run copernicus-wps-demo as a Docker container, see the *Tutorial*.

2.1.5 Use Ansible to deploy copernicus-wps-demo on your System

Use the Ansible playbook for PyWPS to deploy copernicus-wps-demo on your system. Follow the example for copernicus-wps-demo given in the playbook.

2.1.6 Building the docs

First install dependencies for the documentation:

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

2.2 Configuration

2.2.1 Command-line options

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

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

Start service with different hostname and port:

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

2.2.2 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 copernicus-wps-demo
$ 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
$ copernicus start -c etc/custom.cfg
```

2.3 Tutorial

2.3.1 WPS Client Examples

The examples for the WPS clients are using a demo WPS serivce on the bovec test machine at DKRZ.

https://bovec.dkrz.de/ows/proxy/copernicus?Service=WPS&Request=GetCapabilities&Version=1.0.0

There is currently not much data attached (especially for observation data). Please use the default data selection parameters, otherwise no result might be returned.

CURL with HTTP Get requests

GetCapabilities

Run `GetCapabilities` request to see which processes are available:

```
$ curl -s -o caps.xml \
  "https://bovec.dkrz.de/ows/proxy/copernicus?Service=WPS&Request=GetCapabilities&
↪Version=1.0.0"
```

DescribeProcess

Run `DescribeProcess` request to see input/output parameters of the `rainfarm` process:

```
$ curl -s -o describe.xml \
  "https://bovec.dkrz.de/ows/proxy/copernicus?Service=WPS&Request=DescribeProcess&
↪Version=1.0.0&identifier=rainfarm"
```

Execute (sync mode)

Run `Exceute` in synchronous mode for `rainfarm` with default input parameters:

```
$ curl -s -o execute.xml \
  "https://bovec.dkrz.de/ows/proxy/copernicus?Service=WPS&Request=Execute&Version=1.0.
↪0&identifier=rainfarm&DataInputs=regridding=0;slope=0"
```

Warning: The execution request my have a time-out. Please use the *asynchronous* mode for real testing.

A status document is returned. Open the URL with the reference to the output plot:

```
<wps:Output>
  <ows:Identifier>output</ows:Identifier>
  <wps:Reference xlink:href="https://bovec.dkrz.de/download/wpsoutputs/copernicus/
↪40472cbe-8046-11e8-ad8f-109836a7cf3a/RainFARM_example_64x64_in8du35f.png" mimeType=
↪"image/png"/>
</wps:Output>
```

Execute (async mode)

Run `Exceute` in asynchronous mode for `rainfarm`:

```
$ curl -s -o execute.xml \
  "https://bovec.dkrz.de/ows/proxy/copernicus?Service=WPS&Request=Execute&Version=1.0.
↪0&identifier=rainfarm&DataInputs=regridding=0;slope=0&storeExecuteResponse=true&
↪status=true"
```

A status document is returned.

```
<wps:ExecuteResponse
  statusLocation="https://bovec.dkrz.de/download/wpsoutputs/copernicus/8faddd84-8046-
↪11e8-814d-109836a7cf3a.xml">
  <wps:Process wps:processVersion="2.0.0">
    <ows:Identifier>rainfarm</ows:Identifier>
    <ows:Title>RainFARM stochastic downscaling</ows:Title>
  </wps:Process>
  <wps:Status creationTime="2018-07-05T13:28:38Z">
    <wps:ProcessAccepted>PyWPS Process rainfarm accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```

Check the status document given by the `statusLoction` URL until the job has finished:

```
$ curl -s -o status.xml \
  "https://bovec.dkrz.de/download/wpsoutputs/copernicus/8faddd84-8046-11e8-814d-
↪109836a7cf3a.xml"
```

The final status document should similar to this one:

```
<wps:ExecuteResponse
  statusLocation="https://bovec.dkrz.de/download/wpsoutputs/copernicus/8faddd84-8046-
↪11e8-814d-109836a7cf3a.xml">
  <wps:Process wps:processVersion="2.0.0">
    <ows:Identifier>rainfarm</ows:Identifier>
    <ows:Title>RainFARM stochastic downscaling</ows:Title>
  </wps:Process>
  <wps:Status creationTime="2018-07-05T13:28:38Z">
    <wps:ProcessSucceeded>PyWPS Process RainFARM stochastic downscaling finished</
↪wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
    <wps:Output>
      <ows:Identifier>output</ows:Identifier>
      <ows:Title>Output plot</ows:Title>
      <wps:Reference xlink:href="https://bovec.dkrz.de/download/wpsoutputs/copernicus/
↪8faddd84-8046-11e8-814d-109836a7cf3a/RainFARM_example_64x64_s8lwoktx.png" mimeType=
↪"image/png"/>
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Open the URL pointing to the plot output.

OWSLib Python module

OWSLib is a Python library to interact with OWS/OGC services like WPS, WMS, etc. It is using the Python requests library.

http://birdhouse-workshop.readthedocs.io/en/latest/advanced/owslib.html

Birdy Command line client

Birdy is a WPS command line client.

Install birdy (Linux, macOS):

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

Set WPS service:

```
$ export WPS_SERVICE=https://bovec.dkrz.de/ows/proxy/copernicus # demo service on␣
↪bovec
# OR
$ export WPS_SERVICE=http://localhost:5000/wps  # your local WPS service
```

See which processes are available:

```
$ birdy -h
```

Run *rainfarm*:

```
$ birdy rainfarm -h
$ birdy rainfarm --regridding 0 --slope 0
```

Check the process status. The processes should finish after 10 seconds with a response simliar to this one:

```
ProcessSucceeded  [####################################]  100%
Output:
output=https://bovec.dkrz.de/download/wpsoutputs/copernicus/ecac32a0-8047-11e8-ad8f-
↪109836a7cf3a/RainFARM_example_64x64_pe72ysqs.png
```

Open the ouptut URL in Browser to see the plot.

Phoenix Web Client

You can run the demo processes directly without log-in on Phoenix.

- GetCapabilites: https://bovec.dkrz.de/processes/list?wps=copernicus
- DescribeProcess: https://bovec.dkrz.de/processes/execute?wps=copernicus&process=rainfarm
- Execute: press `Submit` button.

Job status is monitored. When job has finished you can either show the output directly or show the output details: https://bovec.dkrz.de/monitor/details/3cd6b18e-81ce-431d-990a-6fca36cae052/outputs

2.3.2 WPS Client Examples with x509 Certificate

A WPS service can be secured with x509 certificates by using the Twitcher OWS security proxy. A WPS `Execute` request can only be run when the WPS client provides a valid x509 proxy certificate.

In the following examples we will use a CP4CDS WPS demo service which is protected by a Twitcher security proxy. It will only accept x509 proxy certificates from ESGF to execute a process. The `GetCapabilites` and `DescribeProcess` requests are public.

CURL with HTTP Get requests

The following examples are using `curl`. You may also like to use the Firefox RestClient plugin.

GetCapabilities

Run `GetCapabilities` request to see which processes are available:

```
$ curl -s -o caps.xml \
  "https://bovec.dkrz.de:5000/ows/proxy/copernicus?Service=WPS&
↪Request=GetCapabilities&Version=1.0.0"
```

DescribeProcess

Run `DescribeProcess` request to see input/output parameters of the `rainfarm` process:

```
$ curl -s -o describe.xml \
  "https://bovec.dkrz.de:5000/ows/proxy/copernicus?Service=WPS&
↪Request=DescribeProcess&Version=1.0.0&identifier=rainfarm"
```

Execute (sync mode)

Run `Exceute` in synchronous mode for `rainfarm` with default input parameters:

```
$ curl -s -o execute.xml \
  "https://bovec.dkrz.de:5000/ows/proxy/copernicus?Service=WPS&Request=Execute&
↪Version=1.0.0&identifier=rainfarm&DataInputs=regridding=0;slope=0"
```

You should get an exception report asking you to provide a x509 certificate:

```
<ExceptionReport>
  <Exception exceptionCode="NoApplicableCode" locator="AccessForbidden">
    <ExceptionText>A valid X.509 client certificate is needed.</ExceptionText>
  </Exception>
</ExceptionReport>
```

Get a valid x509 certifcate from ESGF, for example using the esgf-pyclient. See the logon example. Let's say your proxy certificate is in the file `cert.pem`. Run the curl example above with this certificate:

```
$ curl -s -o execute.xml --cert cert.pem --key cert.pem \
  "https://bovec.dkrz.de:5000/ows/proxy/copernicus?Service=WPS&Request=Execute&
↪Version=1.0.0&identifier=rainfarm&DataInputs=regridding=0;slope=0"
```

If your certificate is valid then your process will be executed (sync mode) and you will get an XML result document providing you with URL references to a generated plot:

```
<wps:Output>
  <ows:Identifier>output</ows:Identifier>
  <wps:Reference xlink:href="http://bovec.dkrz.de:8000/wpsoutputs/copernicus/aaaa6eb2-
↪8056-11e8-9f87-dea873cae3fc/RainFARM_example_64x64_rx7m0ycd.png" mimeType="image/png
↪"/>
</wps:Output>
```

Try more examples as shown in the examples above using a x509 certificate.

Using Python requests library

In this example we show how you can use the Python requests library to run WPS requests.

```
import requests

# GetCapabilites
url = "https://bovec.dkrz.de:5000/ows/proxy/copernicus?request=GetCapabilities&
↪service=WPS"
requests.get(url, verify=True).text
# DescribeProcess
url = "https://bovec.dkrz.de:5000/ows/proxy/copernicus?request=DescribeProcess&
↪service=WPS&version=1.0.0&identifier=sleep"
requests.get(url, verify=True).text
# Execute with client certifcate cert.pem
url = "https://bovec.dkrz.de:5000/ows/proxy/copernicus?request=Execute&service=WPS&
↪version=1.0.0&identifier=sleep&DataInputs=delay=1"
requests.get(url, cert="cert.pem" verify=True).text
```

See the requests documentation for details.

Using OWSLib Python library

An example with OWSLib.

Make sure you have the latest version from the conda birdhouse channel:

```
$ conda install -c birdhouse -c conda-forge owslib
```

Run the *hello* process with a client certificate:

```
from owslib.wps import WebProcessingService
wps = WebProcessingService(url="https://bovec.dkrz.de:5000/ows/proxy/copernicus",
                           verify=True, cert="cert.pem")

exc = wps.execute(identifier='sleep', inputs=[('delay', '1')])
exc.checkStatus()
exc.getStatus()
exc.isSucceded()
exc.processOutputs[0].data
```

Using Birdy

An example with Birdy

Install latest birdy (Linux, macOS) from conda birdhouse channel:

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

```
$ export WPS_SERVICE=https://bovec.dkrz.de:5000/ows/proxy/copernicus
$ birdy -h
$ birdy sleep -h
$ birdy --cert cert.pem sleep --delay 1
```

2.3.3 Using Docker

Get docker images using docker-compose:

```
$ docker-compose pull
```

Start the demo with docker-compose:

```
$ docker-compose up -d  # runs with -d in the background
$ docker-compose logs -f  # check the logs if running in background
```

By default the WPS service should be available on port 5000:

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

Run docker exec to watch logs:

```
$ docker ps     # find container name
copernicus-wps-demo_copernicus_1
$ docker exec copernicus-wps-demo_copernicus_1 tail -f /opt/wps/pywps.log
```

Use docker-compose to stop the containers:

```
$ docker-compose down
```

2.3.4 Testdata

For the demo processes you can fetch CMIP5 test-data from the ESGF archive. You need a valid ESGF credentials which you can fetch for example with esgf-pyclient.

For the examples you need CMIP5 data with the following facets:

- project=CMIP5
- experiment=historical
- ensemble=r1i1p1
- variable=ta, tas, or pr
- model=MPI-ESM-LR
- time_frequency=mon

You can use wget to download ESGF NetCDF files (`-x` option to create directories):

```
$ wget --certificate cert.pem --private-key cert.pem --ca-certificate cert.pem -N -x -
↪P /path/to/esgf/cmip5/archive
```

2.4 Processes

- *Sleep*
- *MyDiag*
- *Perfmetrics*
- *PyDemo*

- *RainFarm*
- *RMSE*

2.4.1 Sleep

class copernicus.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

- PyWPS Demo

2.4.2 MyDiag

class copernicus.processes.wps_mydiag.**MyDiag**

mydiag Simple plot (v2.0.0)

Generates a plot for temperature using ESMValTool. It is a diagnostic used in the ESMValTool tutoriaal doc/toy-diagnostic-tutorial.pdf. The default run uses the following CMIP5 data: project=CMIP5, experiment=historical, ensemble=r1i1p1, variable=ta, model=MPI-ESM-LR, time_frequency=mon

Parameters

- **model** (*{'MPI-ESM-LR', 'MPI-ESM-MR'}*) – Choose a model like MPI-ESM-LR.
- **experiment** (*{'historical', 'rcp26', 'rcp45', 'rcp85'}*) – Choose an experiment like historical.
- **ensemble** (*{'r1i1p1', 'r2i1p1', 'r3i1p1'}*) – Choose an ensemble like r1i1p1.
- **start_year** (*integer*) – Start year of model data.
- **end_year** (*integer*) – End year of model data.

Returns

- **namelist** (*text/plain*) – ESMValTool namelist used for processing.
- **log** (*text/plain*) – Log File of ESMValTool processing.
- **output** (*application/pdf*) – Generated output plot of ESMValTool processing.

References

- ESMValTool
- Documentation

- Media
- ESGF Testdata

2.4.3 Perfmetrics

class copernicus.processes.wps_perfmetrics.**Perfmetrics**

perfmetrics Performance metrics (v2.0.0)

Creates a performance metrics report comparing models using ESMValTool. The goal is to create a standard namelist for the calculation of performance metrics to quantify the ability of the models to reproduce the climatological mean annual cycle for selected Essential Climate Variables (ECVs) plus some additional corresponding diagnostics and plots to better understand and interpret the results. The namelist can be used to calculate performance metrics at different vertical levels (e.g., 5, 30, 200, 850 hPa as in Gleckler et al., 2008) and in four regions (global, tropics 20N-20S, northern extratropics 20-90N, southern extratropics 20-90S). As an additional reference, we consider the Righi et al. (2015) paper.

Parameters

- **model** (*{'MPI-ESM-LR', 'MPI-ESM-MR'}*) – Choose a model like MPI-ESM-LR.
- **experiment** (*{'historical', 'rcp26', 'rcp45', 'rcp85'}*) – Choose an experiment like historical.
- **ensemble** (*{'r1i1p1', 'r2i1p1', 'r3i1p1'}*) – Choose an ensemble like r1i1p1.
- **start_year** (*integer*) – Start year of model data.
- **end_year** (*integer*) – End year of model data.

Returns

- **namelist** (*text/plain*) – ESMValTool namelist used for processing.
- **log** (*text/plain*) – Log File of ESMValTool processing.
- **output** (*application/pdf*) – Generated output plot of ESMValTool processing.

References

- ESMValTool
- Documentation
- Media
- Diagnostic Description
- Diagnostic Metadata

2.4.4 PyDemo

class copernicus.processes.wps_pydemo.**PyDemo**

py_demo Python Demo (v2.0.0)

Generates a plot for temperature using ESMValTool. The default run uses the following CMIP5 data: project=CMIP5, experiment=historical, ensemble=r1i1p1, variable=ta, model=MPI-ESM-LR, time_frequency=mon

Parameters

- **model** (*{'MPI-ESM-LR', 'MPI-ESM-MR'}*) – Choose a model like MPI-ESM-LR.
- **experiment** (*{'historical', 'rcp26', 'rcp45', 'rcp85'}*) – Choose an experiment like historical.
- **ensemble** (*{'r1i1p1', 'r2i1p1', 'r3i1p1'}*) – Choose an ensemble like r1i1p1.
- **start_year** (*integer*) – Start year of model data.
- **end_year** (*integer*) – End year of model data.

Returns

- **namelist** (*text/plain*) – ESMValTool namelist used for processing.
- **log** (*text/plain*) – Log File of ESMValTool processing.
- **output** (*image/png*, *application/pdf*) – Generated output plot of ESMValTool processing.

References

- ESMValTool
- Documentation
- Media
- ESGF Testdata

2.4.5 RainFarm

class copernicus.processes.wps_rainfarm.**RainFarm**

rainfarm RainFARM stochastic downscaling (v2.0.0)

Tool to perform stochastic precipitation downscaling, generating an ensemble of fine-scale precipitation fields from information simulated by climate models at regional scale.

Parameters

- **model** (*{'ACCESS1-0'}*) – Choose a model like MPI-ESM-LR.
- **experiment** (*{'historical'}*) – Choose an experiment like historical.
- **start_year** (*integer*) – Start year of model data.
- **end_year** (*integer*) – End year of model data.
- **subset** (*string*) – Choose a geographical subset with a Bounding Box: 4,13,44,53
- **regridding** (*boolean*) – Flag for regridding.
- **slope** (*boolean*) – Flag for slope.
- **num_ens_members** (*integer*) – Choose a number of ensemble members.
- **num_subdivs** (*integer*) – Choose a number of subdivisions.

Returns **output** – Generated output plot of ESMValTool processing.

Return type *image/png*

References

- ESMValTool
- Documentation
- Media
- Diagnostic Description
- Diagnostic Metadata

2.4.6 RMSE

class copernicus.processes.wps_rmse.**RMSE**

rmse Modes of variability (v2.0.0)

Tool to compute the RMSE between the observed and modelled patterns of variability obtained through classification and their relative relative bias (percentage) in the frequency of occurrence and the persistence of each mode.

Parameters

- **region** (*{'Arctic'}*) – Choose a region like Arctic.
- **model** (*{'NASA'}*) – Choose a model like NASA.
- **variable** (*{'sic'}*) – Choose a variable like sic.
- **ncenters** (*integer*) – Choose a number of centers.
- **cluster_method** (*{'kmeans'}*) – Choose a cluster method.
- **eofs** (*boolean*) – Choose EOFS.
- **detrend** (*integer*) – Choose a detrend.
- **experiment** (*{'historical', 'rcp26', 'rcp85'}*) – Choose an experiment.

Returns **output** – Generated output plot of ESMValTool processing.

Return type *image/png*

References

- ESMValTool
- Documentation
- Media
- Diagnostic Description
- Diagnostic Metadata

2.5 Changes

2.5.1 0.3.0 (2018-06-22)

- Generated from cookiecutter template (#13).

- Skipped buildout just relying on conda and werkzeug.
- Use sphinx documentation (#12).
- Added "fake" processes rainfarm and rmse from MAGIC demo (#16, #22).
- Using static diagnostics description from MAGIC demo (#20).

2.5.2 0.2.2 (2018-06-18)

- Added WPS client examples (#7).
- Added WPS client examples with x509 certificate (#10).

2.5.3 0.2.1 (2018-02-08)

- fixed ncl installation (#3).
- added demo service using werkzeug.

2.5.4 0.2.0 (2018-02-06)

- using ESMValTool 2.x
- diag list: mydiag and py_demo

2.5.5 0.1.1 (2018-01-29)

- cleaned up Dockerfile.
- added pscopg2 for postgres.
- updated pywps recipe 0.9.3.
- added workaround for broken ncl conda package.
- updated Readme with birdy examples.

2.5.6 0.1.0 (2017-06-06)

- added tutorial diagnostics from esmvaltool (MyDiag, Overview, timeseriesplot).
- timeseriesplot both added with a generic wps profile (opendap, netcdf) and with esgf search facets.
- added NCL coutour plot with opendap and netcdf input.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index