Birdhouse Workshop Documentation

Release 0.1

Carsten Ehbrecht

Mar 28, 2018

Contents:

Welcome to the Birdhouse Workshop. This workshop is a hands-on session, which will guide you in creating a process for a Web Processing Service. During the workshop you will learn how Birdhouse supports this development cycle.

Warning: Under Construction . . .

CHAPTER 1

Let's do it quick

1.1 Ready

1. *Conda*.

1.2 Steady

2. *Getting Started*

1.3 Go

Basics:

3. *Writing a simple Plot Function*
4. *Testing the Plot Function*
5. *Adding a Command-Line Interface*

PyWPS:

6. *Introduction*
7. *Installation*
8. *Processes*
9. *Clients*

CHAPTER 2

Motivation

Let us see the Birds:

CHAPTER 3

Requirements

Warning: You need a good internet connection! We are using GitHub, Conda and external data.

Warning: Make sure you have enough *disk space* available! Especially when you are using a Virtual Machine. You should have **at least 5 GB of free disk space** available.

The main requirement of this workshop is *Conda*. It works on most common platforms like Linux, macOS and Windows. If you don't have Conda yet ... we will install it at the beginning of this workshop.

Note: You will need a text-editor to edit Python code. Choose your favorite one ... if you don't have one yet, you can try Atom or Kate.

Tip: On Windows you can also use VirtualBox with a Linux Installation, like Linux Mint

3.1 Conda

Conda is an Open Source package and environment manager that can help to manage project dependencies. Conda works on Linux, macOS and Windows. It was created for Python programs, but it can package and distribute software for any language. Therefore it allows us to use it for multi-language projects.

Conda allows you to build your own packages and share them via channels on Anaconda Cloud. There is a community effort to build and maintain packages needed by various projects, called Conda Forge.

You can create conda environments with a specified list of packages, similar to Python virtualenv. These environemnts can be documented by a `environment.yml` configuration file and shared with others.

Warning: In this workshop we will install *all* software packages using Conda.

3.1.1 Installation

Note: You don't need admin rights to install conda and conda packages.

Download and install the appropriate Miniconda installer from https://conda.io/miniconda.html

With Anaconda you can create environments that use any Python version (e.g. Python 2.7 or Python 3.6), so install the latest Python 3.x and if you find out later you need a Python 2.7 environment, you can create one.

Linux/macOS

You can *copy and paste* the following script to install Miniconda with default settings:

```
if [[ $(uname) == "Darwin" ]]; then
  url=https://repo.continuum.io/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
elif [[ $(uname) == "Linux" ]]; then
  url=https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
fi
curl $url -o miniconda.sh
bash miniconda.sh -b
export PATH=$HOME/miniconda3/bin:$PATH
```

We also recommend to add the following line to your `~/.bashrc` file to make Miniconda the Python found first than the system Python:

```
export PATH=$HOME/miniconda3/bin:$PATH
```

Windows

Run the installer, choose **Just Me** (not **All Users**), and choose a **Install Location** owned by you.

See also the Conda documentaion

3.1.2 Check your Python version

We are using Python 3.6:

```
$ which python
~/miniconda3/bin/python
$ python --version
Python 3.6.2 :: Continuum Analytics, Inc.
```

3.1.3 Links

- https://www.anaconda.com/blog/developer-blog/conda-data-science/
- https://docs.anaconda.com/docs_oss/conda/install/quick

- https://docs.anaconda.com/docs_oss/conda/test-drive
- https://conda.io/docs/user-guide/cheatsheet.html
- https://www.anaconda.com/blog/developer-blog/what-to-do-when-things-go-wrong-in-anaconda/

3.2 Getting Started

Clone the workshop repo from Github:

```
$ git clone https://github.com/bird-house/birdhouse-workshop.git
```

Note: In this workshop we assume that your workshop sources are in your home folder `~/birdhouse-workshop`. If the sources are located at a different place then you need to adapt the workshop root folder accordingly.

Create the *workshop* conda environment:

```
$ conda create -n workshop python=3
```

Activate the conda *workshop* environment (Linux and macOS):

```
$ source activate workshop
```

Warning: On *Windows* you use the following command:

```
$ activate workshop
```

3.2.1 I don't have git . . .

Don't worry . . . the quickest way to install git is using conda:

```
$ conda install git
```

3.2.2 If things go wrong . . .

Well, this can happen . . . you can easily get into troubles with resolving conda package dependencies. The easiest way to solve it is *tabula rasa* . . . remove the conda environment and install it from new.

Deactivate the current environment (Linux and MacOS):

```
$ source deactivate
```

Warning: On *Windows* you need to use the following command to deactivate the environment:

```
$ deactivate
```

Remove the *workshop* conda environment:

```
$ conda env remove -n workshop
```

Create a new *workshop* environment with *all* dependencies used in this workshop by using a conda `environment.yml` file in the top level folder:

```
$ conda env create -f environment.yml
```

CHAPTER 4

Basics

In the following sections we will write a Python function, which generates a plot from a netCDF file.

4.1 Writing a simple Plot Function

4.1.1 Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

4.1.2 Aim

We are going to write a simple plot function in Python using *matplotlib* and *cartopy*.

Objectives:

- You will learn how to install packages with Conda.
- You will learn the basic usage of the *netCDF*, *matplotlib* and *cartopy* Python libraries.

4.1.3 Run the plotter

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/01_plotter
```

Try the plotter Python module:

```
$ python plotter.py
Traceback (most recent call last):
File "plotter.py", line 1, in <module>
  import matplotlib.pylab as plt
ModuleNotFoundError: No module named 'matplotlib'
```

Oops, something is missing ... please install the `matplotlib` package via Conda:

```
# Let's see what is available
$ conda search -c conda-forge matplotlib

# ... and install it from the conda-forge channel
$ conda install -c conda-forge matplotlib
```

Conda will show you a list of packages, which are going to be installed. Have a look at this list and answer with `y` or just press `enter`.

```
The following NEW packages will be INSTALLED:

matplotlib:       2.0.2-py36_2  conda-forge

Proceed ([y]/n)?
```

We should check now the `plotter.py` source code. Open the `plotter.py` in your favorite editor, some people like `vim`:

```
$ vim plotter.py
```

Besides `matplotlib` there is another `import` for `netCDF4`:

```
from netCDF4 import Dataset
```

Let us install `netcdf4`:

```
# same procedure as above ...
$ conda search -c conda-forge netcdf4
$ conda install -c conda-forge netcdf4
```

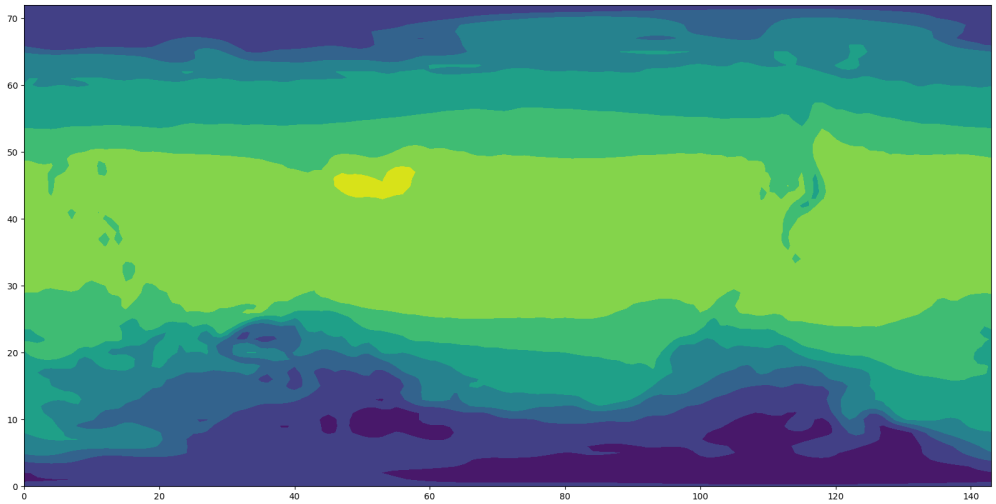
So, we should now be ready to run the plotter:

```
$ python plotter.py
Plotting ../data/air.mon.ltm.nc ...
Plot written to plot.png
```

A plot was generated. Open it in your favorite image viewer. On Ubuntu/LinuxMint you can try *Eye of Gnome*, on macOS just say `open`:

```
$ eog plot.png  # on Ubuntu use Eye of Gnome
or
$ open plot.png # on macOS
```

The image should look like the following:



4.1.4 Exercise

Open the `plotter.py` and implement the following features:

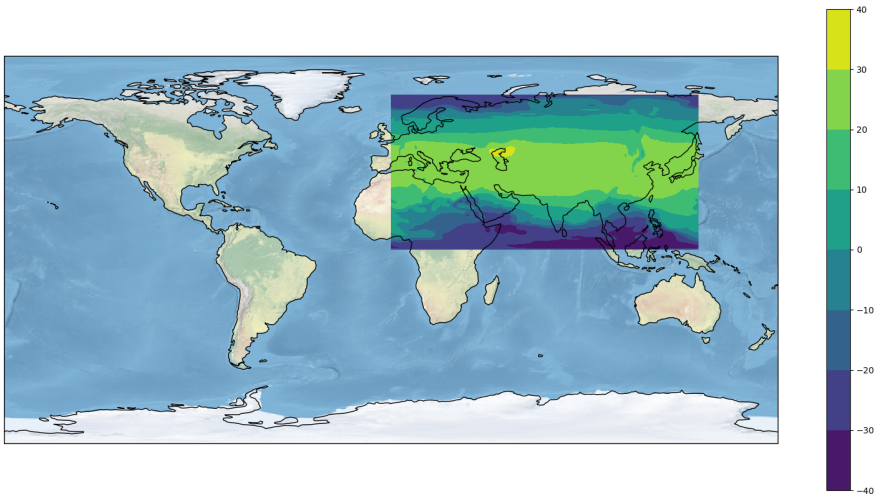
- add a colorbar
- add a background map with coastlines
- use a PlateCarree map projection

You will need an additional Python package, `cartopy`, which you can install with conda. This package is available on the conda-forge channel. You need to provide an option with the conda channel:

```
$ conda install -c conda-forge mypackage
```

Read the code and comments carefully to make this work.

The final result should look like this:



4.1.5 Links

- Matplotlib: http://matplotlib.org/
- Cartopy: http://scitools.org.uk/cartopy/
- Using cartopy with matplotlib
- GeoScience Notebook
- PyEarthScience: matplotlib examples

4.2 Testing the Plot Function

4.2.1 Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

4.2.2 Aim

We are going to write a unit test for our Python plot function.

Objectives:

- You will learn how to write a unit test with *pytest*.

4.2.3 Run pytest

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/02_testing_plotter
```

Run the `plotter.py` like in the previous tutorial and see if it works:

```
$ python plotter.py
```

Now, we want to implement a unit test for our plot function. We are using pytest as testing framework. Install it via conda:

```
$ conda install -c conda-forge pytest
```

Run now *pytest* on our plotting module:

```
$ pytest plotter.py
E       NotImplementedError: This test is not implemented yet. Help wanted!
```

Oops . . . the test is not working yet.

4.2.4 Exercise

Your task is to implement a meaningful test for our `simple_plot` function.

Start hacking `plotter.py` in your favorite editor and run `pytest` frequently.

> **Warning:** Read the comments carefully to make this work and **do not trust each line of code**.

4.2.5 Links

- pytest: https://docs.pytest.org/en/latest/

4.3 Adding a Command-Line Interface

4.3.1 Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

4.3.2 Aim

We are going to write a command line interface (CLI) for our Python plot function.

Objectives:

- You will learn how to write a CLI with the Python library argparse.

4.3.3 Run the plotter CLI

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/03_plotter_cli
```

See the command line options of our plotter:

```
$ python plotter.py -h
usage: plotter.py [-h] [-V [VARIABLE]] dataset
```

Plot our well-know image:

```
$ python plotter.py --variable air ../../data/air.mon.ltm.nc
```

4.3.4 Exercise 1

Play a little bit with the command-line options. Try some other options (`-V`), use invalid input (water) and skip some arguments.

4.3.5 Exercise 2

Use external data from a Thredds service, like NOAA:

https://www.esrl.noaa.gov/psd/thredds/catalog/Datasets/ncep.reanalysis.derived/surface/catalog.html

See access methods for a dataset, note *OpenDAP*:

https://www.esrl.noaa.gov/psd/thredds/catalog/Datasets/ncep.reanalysis.derived/surface/catalog.html?dataset=Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Access:

1. **OPENDAP:** /psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
2. **HTTPServer:** /psd/thredds/fileServer/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
3. **WCS:** /psd/thredds/wcs/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
4. **WMS:** /psd/thredds/wms/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Use *OpenDAP* URLs directly as dataset input:

http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Data URL: http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.

```
$ python plotter.py --variable air \
      http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/
↪surface/air.mon.ltm.nc
```

4.3.6 Exercise 3

Extend the command line and with an optional parameter for the `timestep` to be plotted.

Open your editor on `plotter.py` ... and happy hacking.

Don't forget to test often:

```
$ pytest plotter.py
```

4.3.7 Exercise 4

The output name of the plot is always `plot.png`. Add an optional `output` parameter to set an output filename.

4.3.8 Links

- Python argparse
- NOAA Thredds Data Service

CHAPTER 5

PyWPS

In the following sections we will introduce PyWPS and write a WPS process for a simple plot function.

5.1 Introduction

In the following we describe WPS in general and PyWPS.

5.1.1 What is WPS?

Web Processing Service (WPS) is part of the OWS standards defined by OGC, while WFS, WMS, WCS, SOS are used for transfer of data (upload, download, transformation?), WPS is used for data processing on the server (All processing is done on server side).

WPS provides a standard interface for input, output, process discovery and execution. WPS is normally used for geospatial data to run spatial processes.

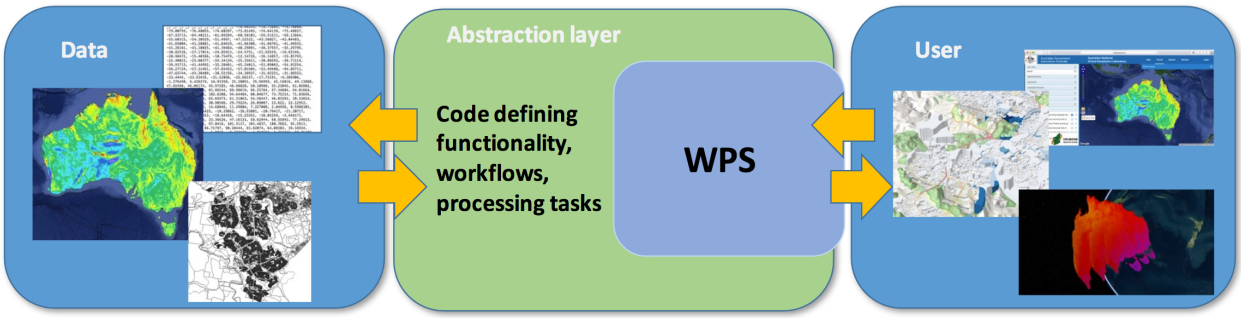


Fig. 5.1: Taken from: http://pointclouds.nci.org.au/talks/f4g_pointwps_adamsteer.pdf

5.1.2 What is PyWPS?

PyWPS is a WPS implementation written in the Python language. The current version is 4.0.0.

- Server (HTTP-WSGI)
- OGC WPS implementation
- Python 3 support
- Based on a webframework (werkzeug)
- Native WSGI itegration (Better server integration)
- MIT license (can be used in comercial projects)

PyWPS has been used in multiple projects concerning geospatial data processing:

- List of scientifical publications
- PyWPS gallery

5.1.3 A brief introdution to WPS

WPS is part of the OGC service suit (OWS) and some operations are common to other services (e.g. **GetCapabilities**), but others specific to WPS itself.

WPS requests:

- **GetCapabilities**
- **DescribeProcess**
- **Execute**

GetCapabilities this request provides a list of available services.

DescribeProcess describes a process indicating the inputs and outputs required by the process to execute and/or for metadata information.

Execute this request will accept inputs/outputs, processing conditions (async/sync) and will run the process on the server.

WPS async/sync

Some processes are time consuming, so it is better to start the process and later query the server for its status or output. This is refered as a async execute request

If you are confident that the process being executed is fast you can request a sync execution where the client waits for the immeditely reply from server withoutput (no need to pull the output later).

WPS input/output

WPS has 3 sorts of data I/O:

- **Literal**
- **ComplexData**
- **BoundingBox**

The **Literal** is any number (float, int) and string. **ComplexData** is geospatial data in multiple formats (mimetype, e.g: `application/gml+xml`) that can be integrated into the WPS request/response, when using vectorial data this one is transformed into XML and raster binary data coded into base64 (binary coding using ascii symbols).

5.1.4 WPS versions

WPS 1.0.0 was released in 2007, the new WPS 2.0.0 was released in 2015. So far major implementations have only used WPS 1.0.0.

WPS 1.0.0 can start processes, but there is no way to stop them before the process reaches its conclusion... it is like a car without brakes. But not all is bad. New WPS 2.0.0 allows for processes to be cancelled.

5.1.5 Links

- PyWPS
- PyWPS Workshop

5.2 Installation

5.2.1 Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

5.2.2 Aim

We are going to install *PyWPS* and run some example processes.

Objectives:

- You will learn how to install PyWPS, start a WPS service and execute a process.

5.2.3 Install PyWPS

You can install PyWPS via conda. Make sure you install PyWPS from the *birdhouse* conda channel. We also need the *conda-forge* channel, and the channels must be provided in the displayed order (channel priority):

```
$ conda install -c birdhouse -c conda-forge pywps gdal
```

Let's see if this has worked:

```
$ python -c "import pywps"
```

This bash command will load the pywps library and close the console. If the install was properly done *no error messages* will appear.

5.2.4 Start the demo WPS service

This workshop includes a demo service with some example processes. Let's try them.

Start the service by running the following command:

```
# change to workshop root folder
$ cd ~/birdhouse-workshop/
# start demo service
$ python demo/demo.py
```

If everything went well you should have a console output as follows:

```
Configuration file(s) ['demo/default.cfg'] loaded
starting WPS service on http://localhost:5000/wps
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Warning: If you need to start the service on a different port then 5000, you must edit the port in the PyWPS configuration `demo/default.cfg`:

```
[server]
url = http://localhost:5001/wps
outputurl = http://localhost:5001/outputs
```

5.2.5 Service check

To test the service, open your internet browser to this address: http://127.0.0.1:5000/wps.

Alternatively, you can also try `curl`:

```
$ curl "http://127.0.0.1:5000/wps"
```

You will get an XML exception report by the PyWPS service:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- PyWPS 4.0.0 -->
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/ows/1.1␣
↪http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd" version="1.0.0">
  <ows:Exception exceptionCode="MissingParameterValue" locator="service" >
    <ows:ExceptionText>service</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

The good thing . . . the service is running and talking to you :)

5.2.6 Test PyWPS

Test the WPS service itself using a **GetCapabilities** request; insert this address in your browser:

http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

In the **GetCapabilities** XML document notice the following:

- Abstract describing service
- Service provider
- Process Offerings (Title, Abstract, Metadata)

5.2.7 Say hello

We can run now our first process. The **GetCapabilities** XML document tells us that this WPS serivce has a process with identifier `say_hello`. Please find this description in the document. It should look like this:

```
<wps:Process wps:processVersion="1.3.2">
  <ows:Identifier>say_hello</ows:Identifier>
  <ows:Title>Process Say Hello</ows:Title>
</wps:Process>
```

Now, we need some more details about this process. Therefore we do a **DescribeProcess** request; insert this address in your browser:

http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess&version=1.0.0&identifier=say_hello

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess&version=1.0.0&
↪identifier=say_hello"
```

The resulting XML document tells us something about the *input* and *output* parameters, for example there is an input parameter `name`:

```
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>name</ows:Identifier>
    <ows:Title>Input name</ows:Title>
    <LiteralData>
      <ows:DataType ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</
↪ows:DataType>
      <ows:AnyValue/>
    </LiteralData>
  </Input>
```

Let us now execute the `say_hello` process with an input parameter `name` *Birdy*:

http://127.0.0.1:5000/wps?service=WPS&request=Execute&version=1.0.0&identifier=say_hello&DataInputs=name=Birdy

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=Execute&version=1.0.0&
↪identifier=say_hello&DataInputs=name=Birdy"
```

If all went well, you get an output parameter with the value *Hello Birdy*:

```
<wps:ProcessOutputs>
  <wps:Output>
    <ows:Identifier>response</ows:Identifier>
    <ows:Title>Output response</ows:Title>
    <wps:Data>
      <wps:LiteralData dataType="urn:ogc:def:dataType:OGC:1.1:string" uom=
↪"urn:ogc:def:uom:OGC:1.0:unity">Hello Birdy</wps:LiteralData>
    </wps:Data>
  </wps:Output>
</wps:ProcessOutputs>
```

5.2.8 Exercise 1

Try the `say_hello` again with some other input values.

5.2.9 Exercise 2

Before you fall into *sleep* … let's do another exercise. Our service has another process. Which one is it?

Please find it and run an execute request … you need to know the input parameters.

5.2.10 Links

- PyWPS Workshop
- PyWPS Flask Demo
- Geoprocessing Info

5.3 Processes

5.3.1 Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

5.3.2 Aim

We are going to write a PyWPS process.

Objectives:

- You will learn how to write a PyWPS process.

5.3.3 What is a WPS Process?

In PyWPS a process is a Python class that has the following structure:

- The parent `Process` class.
- Four input/ouput classes: `ComplexInput`, `LiteralInput`, `ComplexOutput` and `LiteralOutput`
- The `_handler(request, response)` method
- The `request.inputs` and the `response.output` properties.

Go through the PyWPS documentation on Processes.

5.3.4 Create your first process

Let's create a new process that generates a nice and simple plot from a NetCDF file. We have written a `simple_plot` function, which we can use here. We need to do the following:

1. write the PyWPS process definition,
2. call our `simple_plot` method,
3. activate our process in PyWPS.

5.3.5 Check the plotter function

Change into the tutorial `processes` folder:

```
$ cd ~/birdhouse-workshop/tutorials/10_pywps_process/processes
```

You can find here the `plotter.py` module from our previous exercise:

```
$ ls
plotter.py
```

Let's see if it still works:

```
$ python plotter.py -h
```

Generate a plot:

```
$ python plotter.py ../../../data/air.mon.ltm.nc -V air
dataset=['../../../data/air.mon.ltm.nc'], variable=air
Plotting ../../../data/air.mon.ltm.nc ...
Using map projection <cartopy.crs.PlateCarree object at 0x7fae109538e0>
Plot written to plot.png
Output: plot.png
```

5.3.6 Write the process definition

In the `processes/` folder there is another file:

```
$ ls
wps_simple_plot.py
```

This file contains the process definition. Notice the input and output parameters.

5.3.7 Start the service

Change into the tutorials folder:

```
$ cd ~/birdhouse-workshop/tutorials/10_pywps_process
```

Start the WPS service:

```
$ python ../../demo/demo.py
```

Check if the service is running:

http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

Notice that the `simple_plot` service is not activated. Well, time to exercise ...

5.3.8 Exercise 1

Activate the `SimplePlot` process from the `wps_simple_plot` module. See if it shows up in the **GetCapabilites** request.

Tip: You need to edit `processes/__init__.py` and restart the demo service.

5.3.9 Exercise 2

When the `SimplePlot` process is activated then run a **DescribeProcess** request.

Tip: Find the process `identifier` of `SimplePlot` in the **GetCapabilities** document and adapt the **DescribeProcess** URL from our previous exercise.

5.3.10 Exercise 3

Run an **Execute** request with a remote netCDF file from a Thredds data server.

Use the following request URL.

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL
```

Or as a one-liner:

http://127.0.0.1:5000/wps?Service=WPS&Request=Execute&Version=1.0.0&Identifier=PLOT_IDENTIFIER&DataInputs=variable=air;dataset=@xlink:href=NC_URL

You need to replace **PLOT_IDENTIFIER** with the correct processes identifier. Replace **NC_URL** with a remote netCDF data file (HTTP, not OpenDAP), for example:

https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Notice that the output will be returned as reference, for example:

```
<wps:ProcessOutputs>
  <wps:Output>
    <ows:Identifier>output</ows:Identifier>
    <ows:Title>Simple Plot</ows:Title>
    <wps:Reference xlink:href="http://localhost:5000/outputs/4d075e9a-acf4-11e7-9396-
↪acde48001122/plot_ex33_nbf.png" mimeType="image/png"/>
  </wps:Output>
</wps:ProcessOutputs>
```

5.3.11 Exercise 4

You can also run the process in asynchronous mode by adding the parameters `storeExecuteResponse=true` and `status=true`.

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL&
    storeExecuteResponse=true&
    status=true
```

Warning: Asynchronous requests do not work on Windows.

In this case you will a response, which tells you that the process has been accepted, and you need to poll the status document given by the **statusLocation** URL:

```
<wps:ExecuteResponse
  service="WPS" version="1.0.0" xml:lang="en-US"
  serviceInstance="http://localhost:5000/wps?service=WPS&amp;request=GetCapabilities"
  statusLocation="http://localhost:5000/outputs/c894c1b4-acf7-11e7-b989-acde48001122.
↪xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>simple_plot</ows:Identifier>
    <ows:Title>Simple Plot</ows:Title>
    <ows:Abstract>Returns a nice and simple plot.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2017-10-09T15:43:10Z">
    <wps:ProcessAccepted>PyWPS Process simple_plot accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```

5.3.12 Exercise 5

You can also return the output directly. For this modify the above request and add the `RawDataOutput` parameter:

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL&
    RawDataOutput=output
```

Warning: Due to a bug in PyWPS it works currently only with Python 2.7.

5.3.13 Links

- PyWPS workshop
- Geoprocessing Info
- NOAA Thredds Catalog
- Notebook with WPS requests

5.4 Testing

5.4.1 Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

5.4.2 Aim

As you develop more complex process and use more structured datasets, using simply a web browser to test becomes impractical. In this chapter you get acquainted with alternative tools to interact with a PyWPS instance.

Objectives:

- You will learn how to test a PyWPS process.

5.4.3 wget

Start by trying the *GetCapabilities* request:

```
$ wget -q -O caps.xml \
  "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

Important question: Why `-q`, `-O` and `"` in the comnand:

`-q` quit verbose information about requests.

`-O` Output to file. You can use `-`, and the content will be dumped into the prompt.

" Otherwise wget would not consider & as part of the URL and would cut it.

5.4.4 curl

Similar to *wget* you can also use *curl* to retrieve the *GetCapabilities* XML document:

```
$ curl -s -o caps.xml \
  "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

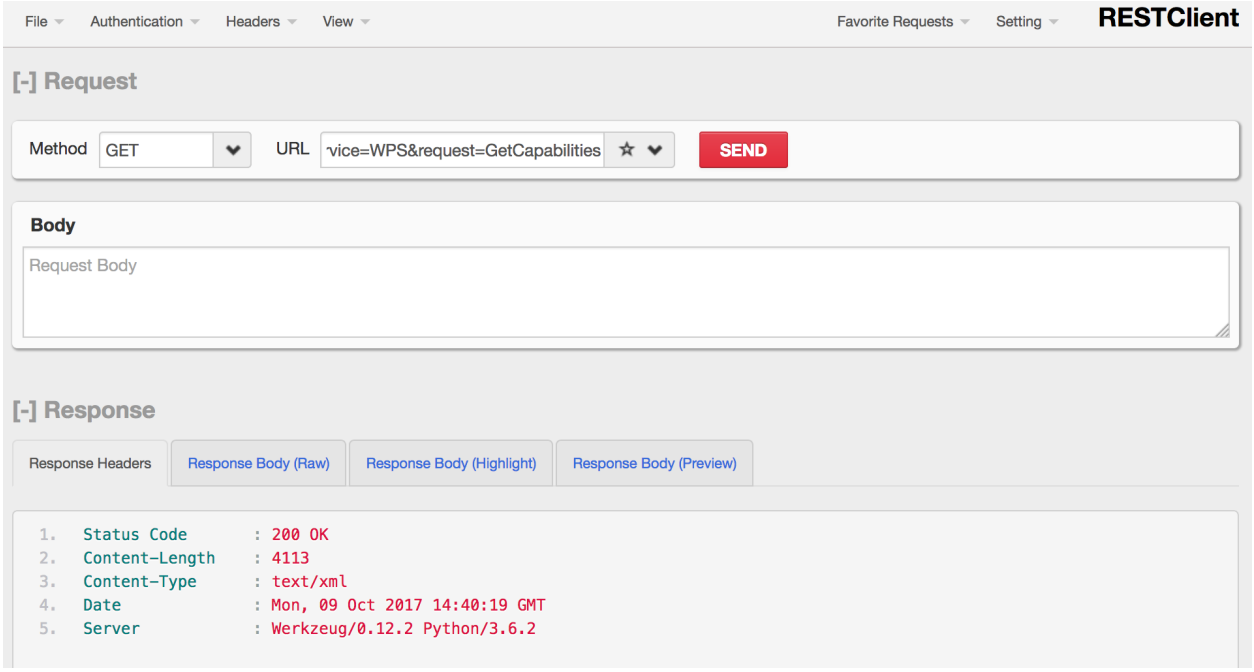
-s silent mode . . . no progress bar.

-o Output to file. You can use –, and the content will be dumped into the prompt.

5.4.5 RESTClient (Firefox only)

You can use the RestClient Firefox plugin to run requests.

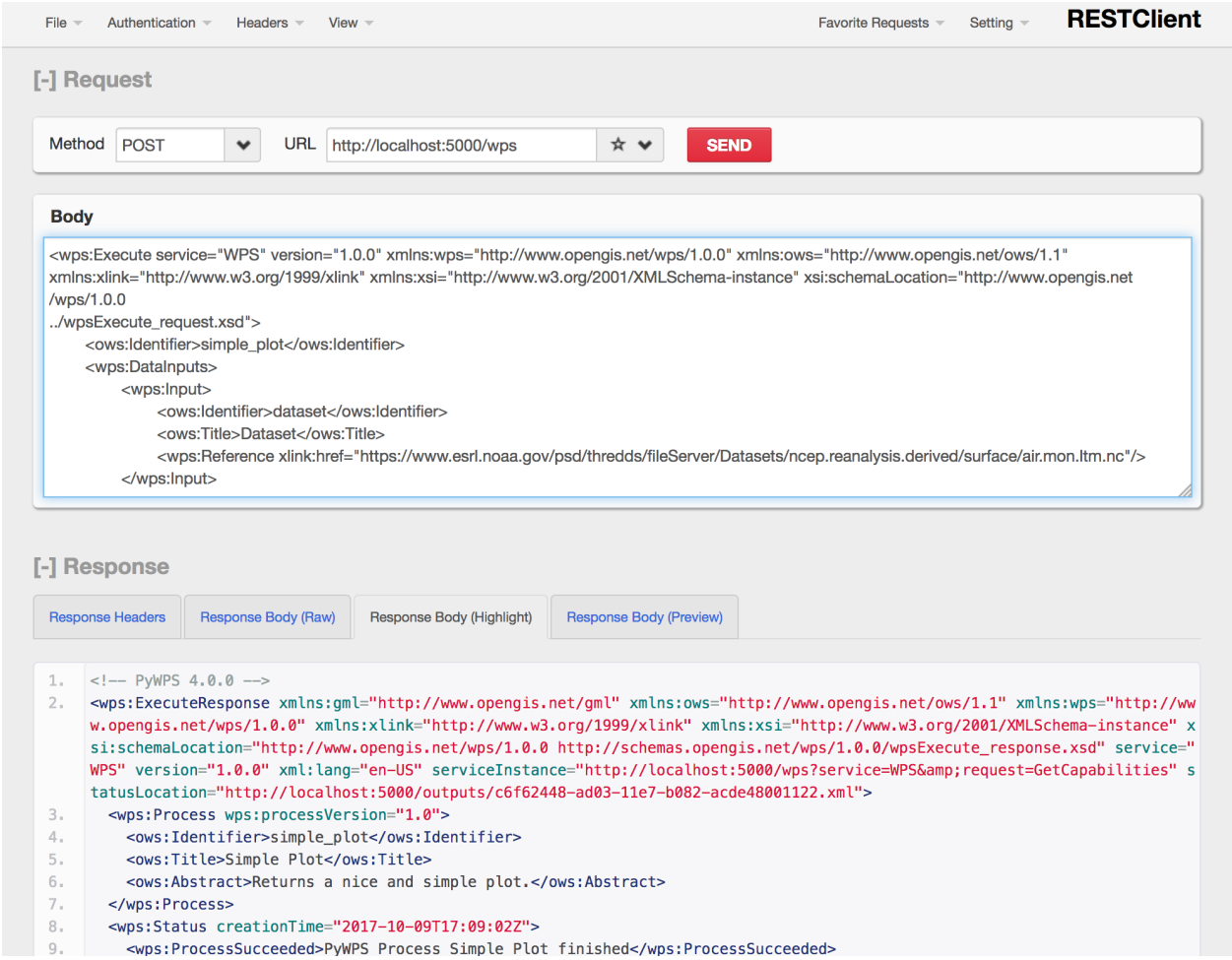
Here is an example with a **GetCapabilities** request using HTTP method **GET**:



5.4.6 XML HTTP Post Request

As requests and data become more structure and lengthy, concatenating all parameters into a URL for a GET type request becomes difficult or impossible. For this reason the WPS standard allows the definition of requests as XML documents sent to the server using the POST method of the HTTP protocol.

Here is an example with an **Execute** request using HTTP method **POST**:



It is using the XML description of the **Execute** request.

It is also possible to use curl (or wget) for POST requests:

```
$ curl -H "Content-Type: text/xml" -X POST \
  -d@execute_req.xml http://localhost:5000/wps
```

-d@ pass data from the given filename (XML payload)

-X HTTP method, GET or POST

-H Header variable, in our case we set the Content-Type.

5.4.7 Exceptions

ExceptionReport is an important feature of WPS. In WPS 1.0.0 we have the following exceptions:

MissingParameterValue The request does not include a parameter value or a default cannot be found.

InvalidParameterValue The request contains an invalid parameter value.

NoApplicableCode Generic exception, no other code could be applied.

NotEnoughStorage The server does not have enough space available.

Try the following request:

http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess"
```

The exception is *MissingParameterValue*:

```
<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/ows/1.1␣
↪http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd" version="1.0.0">
<ows:Exception exceptionCode="MissingParameterValue" locator="version" >
  <ows:ExceptionText>Missing version</ows:ExceptionText>
</ows:Exception>
</ows:ExceptionReport>
```

The *version* parameter is missing.

In case of Python errors in the called process, PyWPS will dump the Python stack into the *ExceptionReport*.

5.4.8 Exercise 1

Try `wget` or `curl` with some of the previous *DescribeProcess* and *Execute* requests.

5.4.9 Exercise 2

Run the **POST** request using the prepared XML payload.

Change into the tutorial `processes` folder:

```
$ cd ~/birdhouse-workshop/tutorials/11_pywps_testing
```

Make sure no WPS service is running ... stop it with `CTRL-c`.

Start the demo service:

```
$ python ../../demo/demo.py
```

Use the above `curl` command with the payload `execute_req.xml`, which you can find in this folder. Modify the input parameters of the payload.

Note: There is another POST request example in the point-clouds talk by NCI.

5.4.10 Links

- RestClient
- Poster on Chrome
- PyWPS workshop
- Geoprocessing Info
- WPS Tutorial

5.5 Logging

5.5.1 Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

5.5.2 Aim

Take a look at the Logging section in the configuration file. PyWPS currently logs events to two different locations:

- A log file where messages are stored. The kind of messages is set in the configuration file.
- A database where each request to the service is registered.

PyWPS uses SQLAlchemy to connect and work with multiple database management systems. SQLite and PostgreSQL tend to be the most used options.

Objectives:

- You will learn how to configure and check the logs.

5.5.3 Check the logs

Our demo WPS service is configured to log to the `pywps.log` file. Using the `tail`, `less` or `cat` commands search for error messages in the `pywps.log` file.

Tip: These messages are preceded by the string "[ERROR]", it is possible to grep the error messages:

```
cat pywps.log | grep "\[ERROR\]"
```

5.5.4 Continuous monitoring

Use the tail command to continuously monitor the activity of the service:

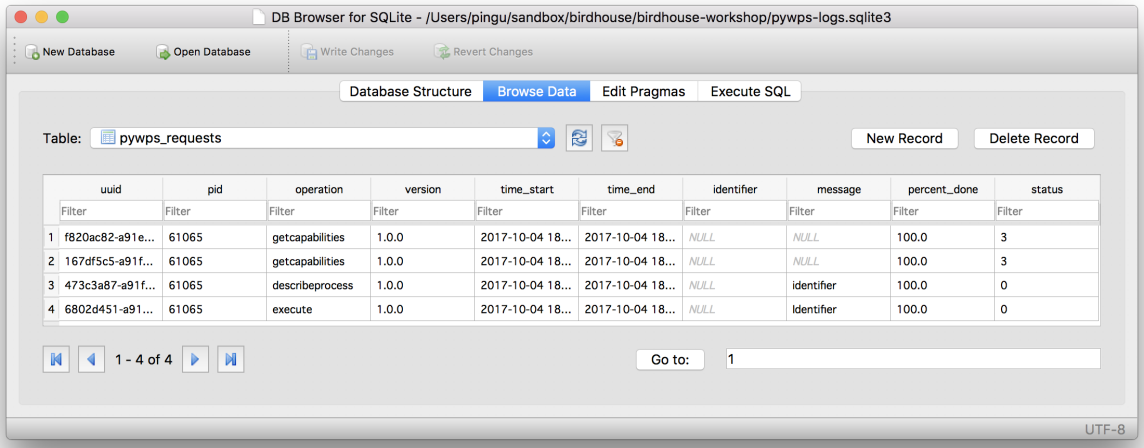
```
$ tail -f pywps.log
```

5.5.5 Database browser

If you have no database browsing programme installed, Install DB Browser for SQLite on your system. On Debian based systems it can be installed from the command line:

```
$ sudo apt install sqlitebrowser
```

See a screenshot with an open SQLite database file `pywps-logs.sqlite3`:



You can also use SQLite on the command-line:

```
$ sqlite3 pywps-logs.sqlite3
sqlite> select * from pywps_requests;
sqlite> .quit
```

5.5.6 Configure logging

Change into the tutorials folder `pywps_logging`:

```
$ cd ~/birdhouse-workshop/tutorials/12_pywps_logging
```

It contains a `pywps.cfg` file with a logging section. You can overwrite the default PyWPS configration by starting the PyWPS service with another config:

```
$ python ../../demo/demo.py -c pywps.cfg
loading configuration
Configuration file(s) ['../../demo/default.cfg', 'pywps.cfg'] loaded
```

5.5.7 Exercise 1

Edit the `pywps.cfg` ... use `DEBUG` logging level. Start the demo WPS service and monitor the log file.

Run a few processes.

5.5.8 Exercse 2

Start the demo WPS service with processes form the previous tutorial. Add some `logging` statements and monitor the service.

5.5.9 Links

- PyWPS workshop

5.6 Clients

5.6.1 Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

5.6.2 Aim

We are going to use a WPS client.

Objectives:

- You will learn how to use the Birdy WPS clients.
- You can try an online demo with the Phoenix Web UI.

5.6.3 Birdy

Birdy is a command-line client for Web Processing Services. It is using the Python library *OWSLib* to interact with WPS services.

Install it via conda:

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

Start the demo WPS service:

```
# go to the workshop root folder
$ cd ~/birdhouse-workshop/
# start wps service
$ python demo/demo.py
```

Warning: The WPS service is running in *foreground*. You need to open a new terminal and activate the conda *workshop* environment for the *birdy* WPS client.

Tip: You can also start the WPS service in *background*:

```
$ python demo/demo.py -d
```

Remember the process id to kill the service:

```
forked process id:  16483
```

Let birdy know the WPS service URL:

```
$ export WPS_SERVICE=http://localhost:5000/wps
```

Note: On Windows you can use:

```
$ set WPS_SERVICE=http://localhost:5000/wps
```

See which processes are available:

```
$ birdy -h
usage: birdy [<options>] <command> [<args>]
```

Show the description of `say_hello`:

```
$ birdy say_hello -h
usage: birdy say_hello [-h] --name [NAME]
                    [--output [{response} [{response} ...]]]
```

Run `say_hello`:

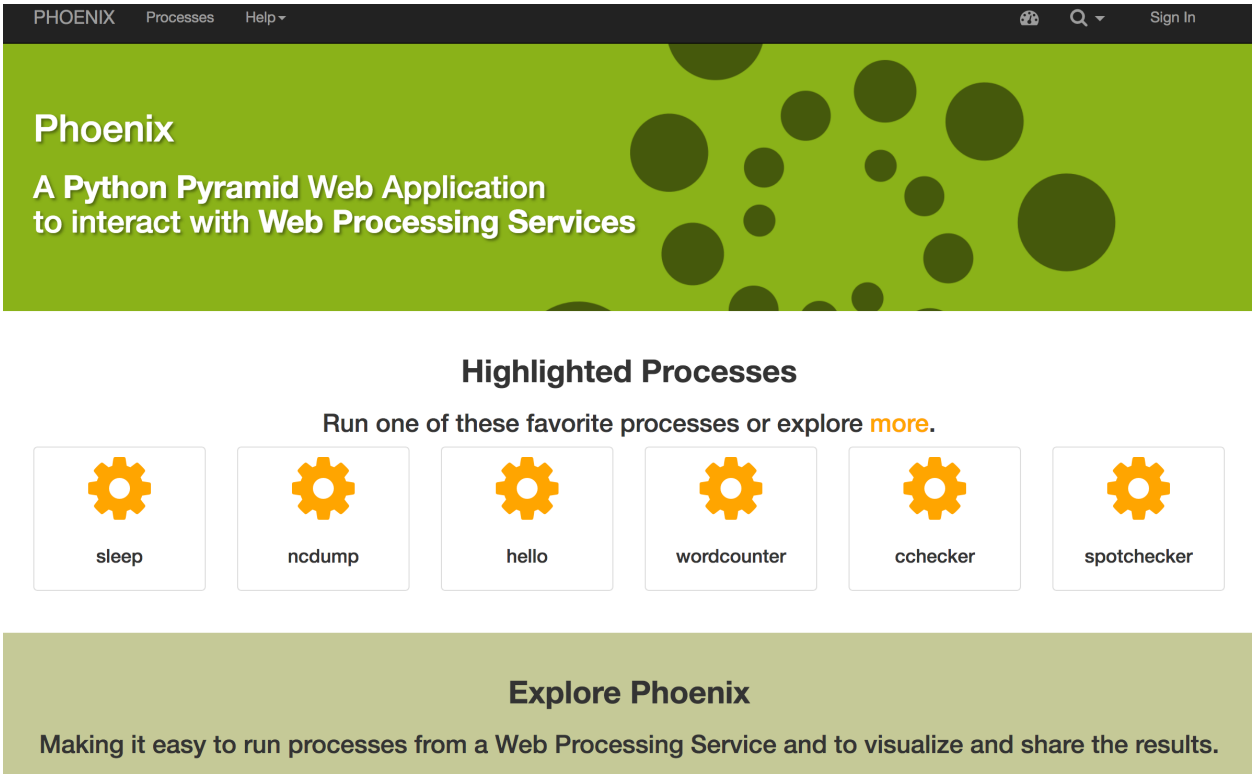
```
$ birdy say_hello --name Birdy
[ProcessAccepted 0/100] PyWPS Process say_hello accepted
[ProcessSucceeded 0/100] PyWPS Process Process Say Hello finished
Output:
response=Hello Birdy
```

Warning: On Windows the asynchronous call does not work, which birdy uses by default. Use the `--sync` option instead:

```
$ birdy --sync say_hello --name Birdy
```

5.6.4 Phoenix

Phoenix is Pyramid web-application to interact with WPS services.



You can try the online demo.

5.6.5 Exercise 1

Play with birdy . . . run the `sleep` process.

5.6.6 Exercise 2

Start the demo service with the processes from *Testing* and call the `simple_plot` process with **birdy**.

5.6.7 Exercise 3

Try some of the processes on the Phoenix demo. For example the **wordcounter** and the **spotchecker**.

5.6.8 Links

- Birdy
- Phoenix
- Pyramid
- OWSLib

CHAPTER 6

The Birds

In the following sections we will introduce the Birds.

6.1 Introduction

Birdhouse is a collection of Web Processing Service (WPS) related Python components to support climate data analysis. Birdhouse uses OGC/WPS software from the GeoPython project, like PyWPS and OWSLib.

The aim of Birdhouse is to support (climate science) projects to setup a Web Processing Service infrastructure.

Birdhouse is the *Home* of several *Birds*, the components of the Birdhouse ecosystem. There are birds for the Web Processing client side, to make the WPS service access more convenient and also as an example for project own Web UIs. There are fully configured WPS services with example processes, which run *out-of-the-box* and can be forked and used as template. There is also a middleware component to control access to WPS services.

The Birdhouse documentation gives an overview of the architecture.

The Birdhouse components can be installed with a simple `make install`. See the installation documentation for details.

All Birdhouse components are Open Source and released under the Apache License. The source code is available on GitHub.

See the documentation of the Birdhouse components and try the demo.

6.2 Live Demo

6.2.1 Phoenix

Showing Phoenix with CDO, Spotchecker and Subsetting process with ESGF, OpenDAP and uploaded data.

6.2.2 Birdy

Showing Birdy with ncdump on ESGF data (using access token).

6.2.3 curl

Using `curl` to run the `wordcounter` on an external service.

CHAPTER 7

Advanced

In the following sections we will go into advanced topics.

7.1 OWSLib

OWSLib is a Python library for client programming with Open Geospatial Consortium (OGC) web service (hence OWS), like WMS, CSW and WPS.

We are using an Jupyter notebook to look at some example code.

Todo: Add ssh tunnel or jupyter console example.

We need to install Jupyter via conda:

```
$ conda install -c conda-forge jupyter
```

Go to the tutorials folder `pywps_clients`:

```
$ cd ~/birdhouse-workshop/tutorials/31_owslib
```

You will find there an Jupyter notebook:

```
$ ls
owslib-wps.ipynb
```

Open the Jupyter notebook:

```
$ jupyter notebook
```

And point your browser to the following URL:

http://localhost:8888/notebooks/owslib-wps.ipynb

Or see it on GitHub.

7.1.1 Links

- https://try.jupyter.org/
- https://nbviewer.jupyter.org/

7.2 ESGF

7.2.1 Using ESGF pyclient to access ESGF data

Example Notebook: https://github.com/cehbrecht/demo-notebooks

7.2.2 Use birdy command-line with ESGF data

Example: http://birdy.readthedocs.io/en/latest/tutorial.html

7.2.3 Use Phoenix Wizard with ESGF data

Example: http://pyramid-phoenix.readthedocs.io/en/latest/tutorial/visualisation.html

7.3 Docker

The Birdhouse WPS services are available as a Docker image on Docker Hub.

7.3.1 What is Docker?

https://www.docker.com/what-docker

7.3.2 Run Emu as Docker container

Example: http://emu.readthedocs.io/en/latest/tutorial/using_docker.html

7.3.3 Links

- Docker Training: http://slides.com/dataduke/docker-001#/

7.4 Travis CI

Todo: add travis example

Continuous Integration with Travis . . . triggered by commit on GitHub and via cron job.

See Emu example:

https://travis-ci.org/bird-house/emu

Travis config:

https://github.com/bird-house/emu/blob/master/.travis.yml

WPS tests:

https://github.com/bird-house/emu/blob/master/emu/tests/test_wps_hello.py

CHAPTER 8

Appendix

8.1 Why using WPS?

- Web based services could help researchers collaborate
  - The fact that individual researchers are increasingly specialized raises the "cost" of interacting with other disciplines.
  - Due to these costs, multidisciplinary projects are often run in parallel, with no real dependencies and synergies between teams.
  - Open source code has helped tremendously, but there is a large resource gap between installing software and having a working application, especially in earth system modeling.
- Why would individual scientists publish services?
  - Increased visibility and funding opportunities.
  - Improved research quality.
  - Participate to intercomparison projects.
- Access to external resources
  - operations would be calculated on the server, while the system resources could be exposed to clients.
  - large climate data stores
  - compute resources
  - complex software systems
- Cross institutional, cross-community
  - depends *only* on an open standard interface.
  - several implementations for a Processing Service can be used.
  - clients (web portals) can rely on a stable processing service interface.

8.2 Who is using WPS?

8.2.1 Copernicus, EU Project

- Copernicus Climate Change Service: http://climate.copernicus.eu/
- WPS Demo: https://github.com/cp4cds/copernicus-wps-demo

8.2.2 CEDA/STFC, UK

- CEDA: http://www.ceda.ac.uk/
- STFC: http://www.stfc.ac.uk/
- COWS WPS: http://wps-web1.ceda.ac.uk/ui/home

8.2.3 IPSL/LSCE, France

- IPSL: https://www.ipsl.fr/en/
- LSCE: http://www.lsce.ipsl.fr/
- Talk at Euro Cordex 2016
- Paper about Flyingpigeon (in Review)

8.2.4 KNMI, Netherlands

- KNMI: http://www.knmi.nl/
- Climate4Impact Portal: https://climate4impact.eu/

8.2.5 ESGF: lead by LLNL, US

- LLNL: https://www.llnl.gov/
- ESGF Project: https://esgf.llnl.gov/
- Compute WPS https://github.com/ESGF/esgf-compute-wps

8.2.6 CRIM/Ouranos, Canada

- CRIM: http://www.crim.ca/en/
- Ouranos: https://www.ouranos.ca/en/
- Talk at AGU 2016

8.2.7 NCI, Australia

- NCI: http://nci.org.au/
- Talk about WPS for Pointclouds

8.2.8 DKRZ, Germany

- DKRZ: https://www.dkrz.de/dkrz-en
- Birdhouse: http://bird-house.github.io/
- Talk at ESGF F2F, 2016

8.2.9 KIT, Germany

- KIT: http://www.kit.edu/english/
- Talk at EGU 2017

8.2.10 APEC Climate Center, South Korea

- APCC: http://www.apcc21.org/
- Talk at FOSS4G, Bonn, 2016

8.3 Links

Birdhouse:

- http://bird-house.github.io/
- Birdhouse Workshop: http://birdhouse-workshop.readthedocs.io/en/latest/
- Birdhouse talks: http://birdhouse.readthedocs.io/en/latest/index.html#presentations-blog-posts

WPS:

- http://geoprocessing.info/index.html
- PyWPS: http://pywps.org/
- PyWPS Workshop: https://github.com/PyWPS/pywps-workshop

Conda:

- https://conda.io/docs/
- https://www.anaconda.com/blog/developer-blog/conda-data-science/

Python:

- pytest: https://docs.pytest.org/en/latest/
- Python argparse: https://docs.python.org/3/howto/argparse.html

Jupyter Notebooks:

- Notebook Gallery: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks
- IOOS Demos: https://ioos.github.io/notebooks_demos/

Cartopy/Matplotlib:

- Matplotlib: http://matplotlib.org/
- Cartopy: http://scitools.org.uk/cartopy/

- Using cartopy with matplotlib
- GeoScience Notebook
- PyEarthScience matplotlib examples: https://github.com/KMFleischer/PyEarthScience/

netCDF:

- http://nbviewer.jupyter.org/github/julienchastang/unidata-python-workshop/blob/master/reading_netCDF.ipynb
- http://schubert.atmos.colostate.edu/~cslocum/netcdf_example.html

OpenDAP:

- https://www.seegrid.csiro.au/wiki/pub/AUKEGGS/FinalWorkshop/seminar.pdf

Docker:

- What is Docker?: https://www.docker.com/what-docker
- Docker Training: http://slides.com/dataduke/docker-001#/
- Play with Docker: http://labs.play-with-docker.com/

8.4 Todo List

Todo: Add ssh tunnel or jupyter console example.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse-workshop/checkouts/latest/docs/source/advanced/owslib.rst, line 12.)

Todo: add travis example

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse-workshop/checkouts/latest/docs/source/advanced/travis.rst, line 6.)