Birdhouse - supporting Web Processing Services

Carsten Ehbrecht[1] Stephan Kindermann[1] Nils Hempelmann[2]

[1] DKRZ - German Climate Compute Center

[2] GIZ - German Development Cooperation

28th of November 2017/ Python Frameworks Workshop at ECMWF

Outline

Motivation

The OGC Web Processing Service

GET

Post

SOAP

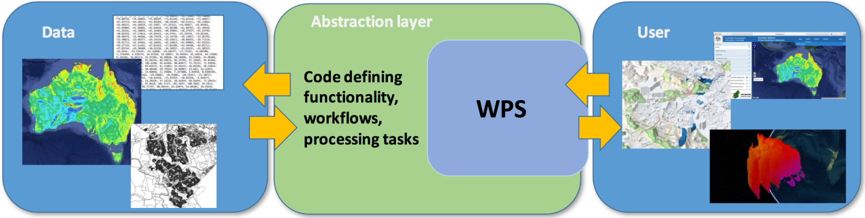
GetCapabilit

DescribePro

Execute

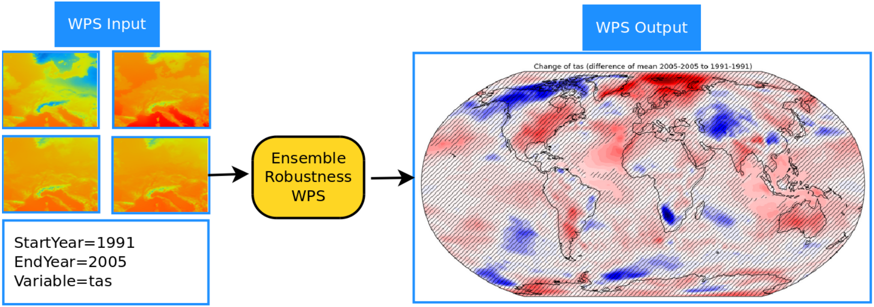
http://www.slideshare.net/TheodorFoerster/restful-web-processing-service

WPS Use Case



WPS for Point Clouds by Adam Steer, NCI, Australia

WPS Inputs and Outputs



WPS Input

StartYear=1991
EndYear=2005
Variable=tas

Ensemble Robustness WPS

WPS Output

Function as a Service

```
@wps  # wps decorator
def myplot(nc_file, variable):
    """
    Generates a plot for given dataset and variable.

    nc_file application/netcdf Dataset
    variable string Variable name
    """
    # ... create a nice plot here
    return plot.png
```

Running Function as Web Processing Service

```
$ curl -s -o result.xml \
http://localhost/wps? \
 &service=WPS \
 &version=1.0.0 \
 &request=execute \
 &identifier=myplot \
 &DataInputs=nc_file=http://;variable=tas
```

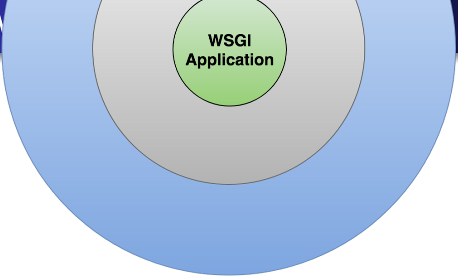
PyWPS

What is PyWPS?

- An implementation of the OGC Web Processing Service standard
- Implements WPS 1.0.0 standard (WPS 2.0.0 in progress)
- Coded in the Python language (researcher friendly)
- Easy to hack (developer friendly)
- Relevant contributions by over a dozen individuals
- OSGeo accreditation around the corner . . .

`http://pywps.org`

The W



- Nginx - security, reverse proxy, load balancing
- Gunicorn - concurrency, WSGI
- WSGI App - PyWPS WSGI application

It’s complicated!

- Scalability requires the orchestration of various software layers:
  - Nginx
  - Gunicorn
  - PyWPS
- Many packages to install
- Many configurations to set-up
- No clients for testing

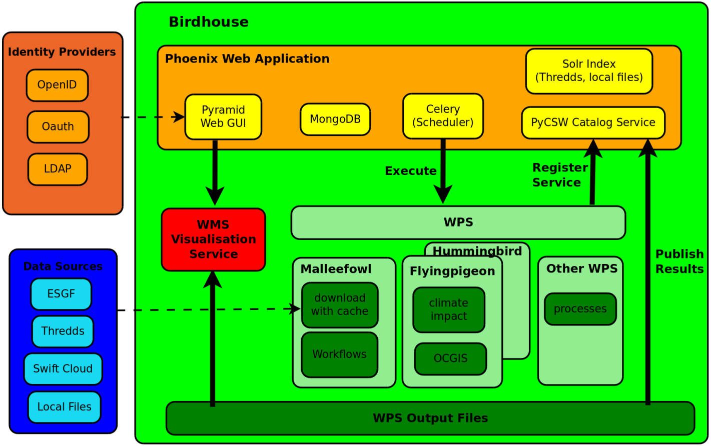
Too much work?

Birdhouse

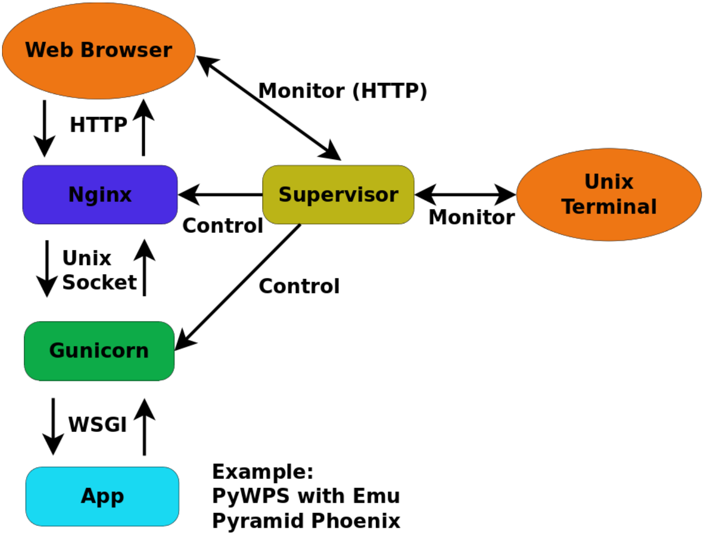
What is Birdhouse?

- Supporting OGC Web Processing Services in the climate science community.
- Making it easier to set-up WPS services (Birdhouse-Builder).
- Providing Python library and WPS processes to access climate data.
- Providing a security proxy middleware to protect OGC/WPS services.
- Providing a web application and command-line client to interact with WPS services.
- `http://bird-house.github.io/`

Birdhouse Overview



WSGI Application controlled by Supervisor



Manage

- Many components: PyWPS, Nginx, Gunicorn, ncWMS, . . .
- Lots of packages: cdo, cfchecker, OCGIS, numpy, R, ESMValTool, . . .
- Many configurations to set-up
- Reproducible installation
- Different Linux distributions (Centos, Debian, . . .)

Conda Package Manager

- Originally for python ... but has a general concept
- Does not need admin rights

Install PyWPS from birdhouse channel

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

Create conda environment **pywps**

```
$ conda create -n pywps -c birdhouse pywps
```

Deployment with Buildout

- Makefile to wrap Buildout commands
- A common deployment pattern for all Birds

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

Update configuration like hostname, port

```
$ vim custom.cfg
$ make update
$ make restart
```

Docker

- Docker Hub is a public repository for dockerfiles
- Birdhouse repository with automatically updated docker images

`https://hub.docker.com/u/birdhouse/`

Try a Docker . . .

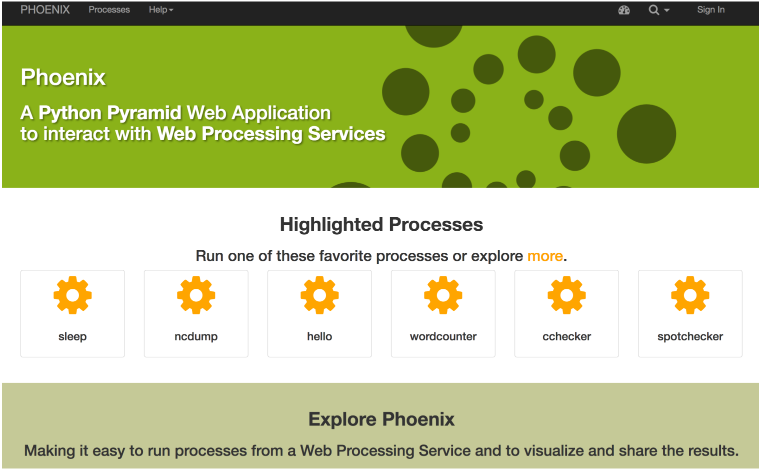
Running a docker image with Emu WPS on port 8080

```
$ docker pull birdhouse/emu
$ docker run -it -p 8000:8000 -p 8080:8080 \
       --name=emu_wps birdhouse/emu
```

Running a WPS GetCapabilities request

```
$ curl -s -o caps.xml \
http://localhost:8080/wps? \
  service=WPS& \
  version=1.0.0& \
  request=GetCapabilities
```

Phoenix web-based WPS client



Phoenix

A Python Pyramid Web Application to interact with **Web Processing Services**

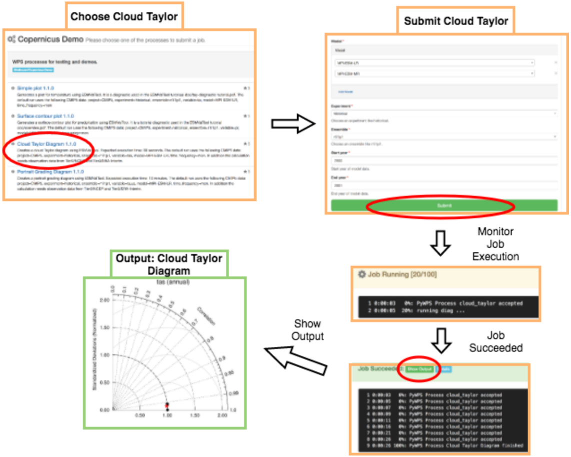
Highlighted Processes

Run one of these favorite processes or explore more.

Explore Phoenix

Making it easy to run processes from a Web Processing Service and to visualize and share the results.

Phoenix Example



Choose Cloud Taylor

Copernicus Demo Please choose one of the processes to submit a job.

WPS processes for testing and demos.

- Simple plot 1.1.0 ★ 1
  [illegible]
- Surface contour plot 1.1.0 ★ 1
  [illegible]
- Cloud Taylor Diagram 1.1.0 ★ 1
  [illegible]
- Portrait Grading Diagram 1.1.0 ★ 1
  [illegible]

Submit Cloud Taylor

Monitor Job Execution

Job Running [20/100]

```
1 0:00:03   0%: PyWPS Process cloud_taylor accepted
2 0:00:05  20%: running diag ...
```

Job Succeeded

Job Succeeded: Show Output Details

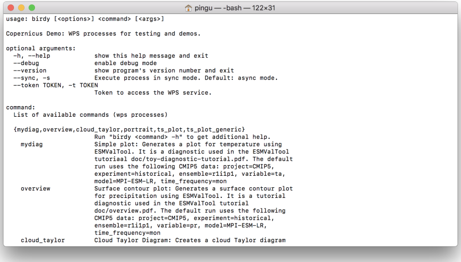
```
1 0:00:03   0%: PyWPS Process cloud_taylor accepted
2 0:00:05   0%: PyWPS Process cloud_taylor accepted
3 0:00:07   0%: PyWPS Process cloud_taylor accepted
4 0:00:09   0%: PyWPS Process cloud_taylor accepted
5 0:00:11   0%: PyWPS Process cloud_taylor accepted
6 0:00:16   0%: PyWPS Process cloud_taylor accepted
7 0:00:21   0%: PyWPS Process cloud_taylor accepted
8 0:00:26   0%: PyWPS Process cloud_taylor accepted
9 0:00:26 100%: PyWPS Process Cloud Taylor Diagram finished
```

Show Output

Output: Cloud Taylor Diagram

Birdy command line WPS client

```
$ conda install -c birdhouse birdhouse-birdy
$ export WPS_SERVICS=http://localhost:8094/wps
$ birdy -h
```

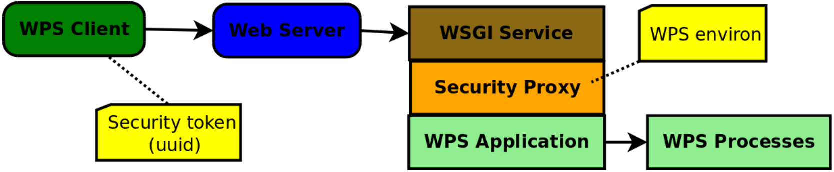


pingu — -bash — 122×31

```
usage: birdy [<options>] <command> [<args>]

Copernicus Demo: WPS processes for testing and demos.

optional arguments:
  -h, --help            show this help message and exit
  --debug               enable debug mode
  --version             show program's version number and exit
  --sync, -s            Execute process in sync mode. Default: async mode.
  --token TOKEN, -t TOKEN
                        Token to access the WPS service.

command:
  List of available commands (wps processes)

  {mydiag,overview,cloud_taylor,portrait,ts_plot,ts_plot_generic}
                        Run "birdy <command> -h" to get additional help.
    mydiag              Simple plot: 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
    overview            Surface contour plot: Generates a surface contour plot
                        for precipitation using ESMValTool. It is a tutorial
                        diagnostic used in the ESMValTool tutorial
                        doc/overview.pdf. The default run uses the following
                        CMIP5 data: project=CMIP5, experiment=historical,
                        ensemble=r1i1p1, variable=pr, model=MPI-ESM-LR,
                        time_frequency=mon
    cloud_taylor        Cloud Taylor Diagram: Creates a cloud Taylor diagram
```

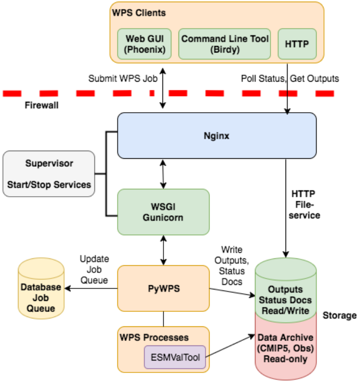
WPS Security Proxy

- Using string token (uuid) as part of URL or in request header to protect WPS execute access
- X509 certificates to access (remote) data from ESGF are provided by proxy (using environ)
- Implemented as WSGI middleware service
- `http://twitcher.readthedocs.io/en/latest/`



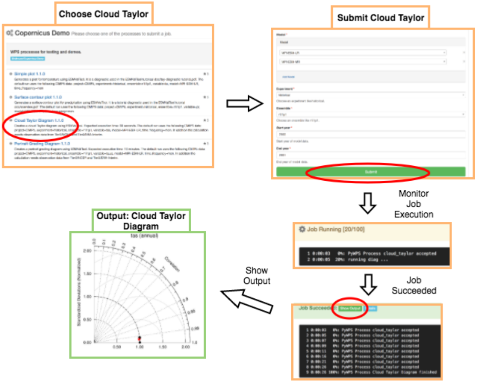
Examples

Copernicus: ESMValTool diagnostics as WPS



`https://github.com/cp4cds/copernicus-wps-demo`

Copernicus: Example Run



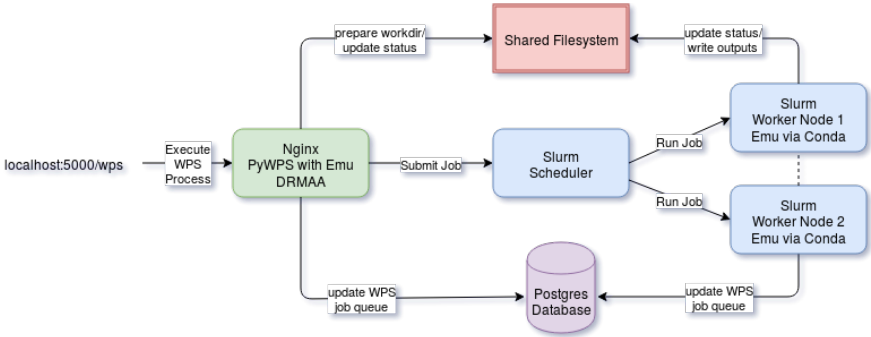
Choose Cloud Taylor
Copernicus Demo Please choose one of the processes to submit a job.
WPS processes for testing and demos.
Simple plot 1.1.0
Surface contour plot 1.1.0
Cloud Taylor Diagram 1.1.0
Portrait Grading Diagram 1.1.0
Submit Cloud Taylor
Submit
Monitor Job Execution
Job Running [20/100]
Job Succeeded
Job Succeeded: Show Output Details
Show Output
Output: Cloud Taylor Diagram
tas (annual)
Standard Deviations (Normalized)
Correlation
0.00
0.50
1.00
1.50
2.00

```
1 0:00:03   0%: PyWPS Process cloud_taylor accepted
2 0:00:05  20%: running diag ...
```

```
1 0:00:03   0%: PyWPS Process cloud_taylor accepted
2 0:00:05   0%: PyWPS Process cloud_taylor accepted
3 0:00:07   0%: PyWPS Process cloud_taylor accepted
4 0:00:09   0%: PyWPS Process cloud_taylor accepted
5 0:00:11   0%: PyWPS Process cloud_taylor accepted
6 0:00:16   0%: PyWPS Process cloud_taylor accepted
7 0:00:21   0%: PyWPS Process cloud_taylor accepted
8 0:00:26   0%: PyWPS Process cloud_taylor accepted
9 0:00:26 100%: PyWPS Process Cloud Taylor Diagram finished
```

https://bovec.dkrz.de/processes/list?wps=copernicus

Copernicus: PyWPS Scheduler Extension


localhost:5000/wps
Execute WPS Process
Nginx PyWPS with Emu DRMAA
prepare workdir/ update status
Shared Filesystem
update status/ write outputs
Submit Job
Slurm Scheduler
Run Job
Run Job
Slurm Worker Node 1 Emu via Conda
Slurm Worker Node 2 Emu via Conda
update WPS job queue
Postgres Database
update WPS job queue

This extension is part of PyWPS

PAVICS: A Platform for the Analysis and Visualization of Climate Science

- Project based on Birdhouse by Ouranos and CRIM, Canada
- Ouranos needs a platform for climate services
- Creating and delivering climate products
- Make climate research less painful

https://www.researchgate.net/project/PAVICS

Summary

Summary

- Deployment
  - Nginx + Gunicorn provide the infrastructure for scalable services
  - Birdhouse supports automatic deployment using Conda and Buildout
- Toolbox
  - Web portal and command-line tool for testing and demo of WPS services
  - Security middleware to protect the execution of WPS processes
- Get your hands dirty
  - Birdhouse Workshop: `http://birdhouse-workshop.readthedocs.io/en/latest/index.html`
  - PyWPS Workshop: `https://github.com/PyWPS/pywps-workshop`

The End

http://bird-house.github.io/

Appendix

The OGC Web Processing Service

- OGC open web standard for remote geo-spatial processing.
- Other widley used OGC web standards: **WMS**, **WFS**, **WCS**.
- Three basic requests:
  - *GetCapabilities*
  - *DescribeProcess*
  - *Execute*
- Three basic input/output classes:
  - *Literal*
  - *Complex* - for geo-spatial data and services
  - *BoundingBox* - for geo-spatial data extent

What does WPS provide?

- web access to your algorithms (GET request with key-value, POST request with xml)
- WPS knows about the inputs and outpus of a process
- processes are self-describing (GetCapabilites, DescribeProcess)
- sync and async calls (async calls with status document)
- its a standard interface ... several implementations are available (PyWPS, GeoServer, 52 North, COWS, ...)
- process definition is easy to write
- not restricted to a specific programming language
- can be used internally to provide enhanced functionality to web portals

What is PyWPS good for?

- Make your models available to the world
- Enables remote processing of complex and/or lengthy models
- Guarantees model inputs fit basic requirements (e.g. type, number)
- Guarantees interoperability of model inputs and outputs
  - using the OGC data standards
  - formalizing input/output data types

The role of a WSGI server

- WSGI - Common interface to multiple web applications frameworks
- WSGI Server - basic functions:
  - accepts HTTP requests
  - replies to HTTP requests
- WSGI provides *concurrency*, allowing multiple:
  - threads
  - workers
  - processes

Gunicorn (Green Unicorn)

- It is one of many WSGI servers out there
- Easy to configure and use with Python
- Promotes the concepts of “workers”
  - essentially OS processes
- Each worker can run on a different CPU core
  - a worker can be a Flask application instance

http://gunicorn.org

Nginx

- Essentially a web (HTTP) server
- But more used for reverse-proxy
- Acts as a single entrance point to all requests
- Redirects requests to Gunicorn
- Can redirect to multiple Gunicorns
- Gateway to multiple servers and applications from a single URL

http://nginx.org/

Buildout

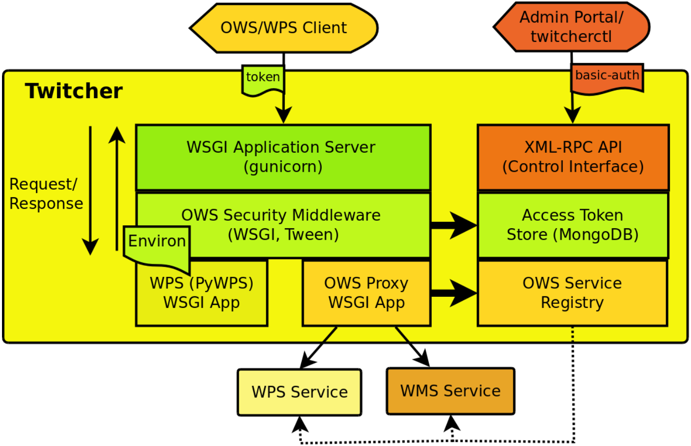
- Python based build system
- creates application with multiple components including configuration files
- works also for non-Python parts
- using a buildout configuration
- can be extended with recipes

Docker

- An OS level virtualisation engine
- Docker runs software *containers*
    - a very light weight virtual machine
- Uses Linux kernel namespaces to isolate available resources:
    - operating environment
    - process and user IDs
    - process trees
    - network
    - mounted file systems
- Virtualisation provided by the OS itself

https://docker.com

Twitcher Security Proxy


OWS/WPS Client
Admin Portal/ twitcherctl
token
basic-auth
Twitcher
Request/ Response
WSGI Application Server (gunicorn)
XML-RPC API (Control Interface)
OWS Security Middleware (WSGI, Tween)
Environ
Access Token Store (MongoDB)
WPS (PyWPS) WSGI App
OWS Proxy WSGI App
OWS Service Registry
WPS Service
WMS Service