

# Birdhouse

Build your own Web Processing Service

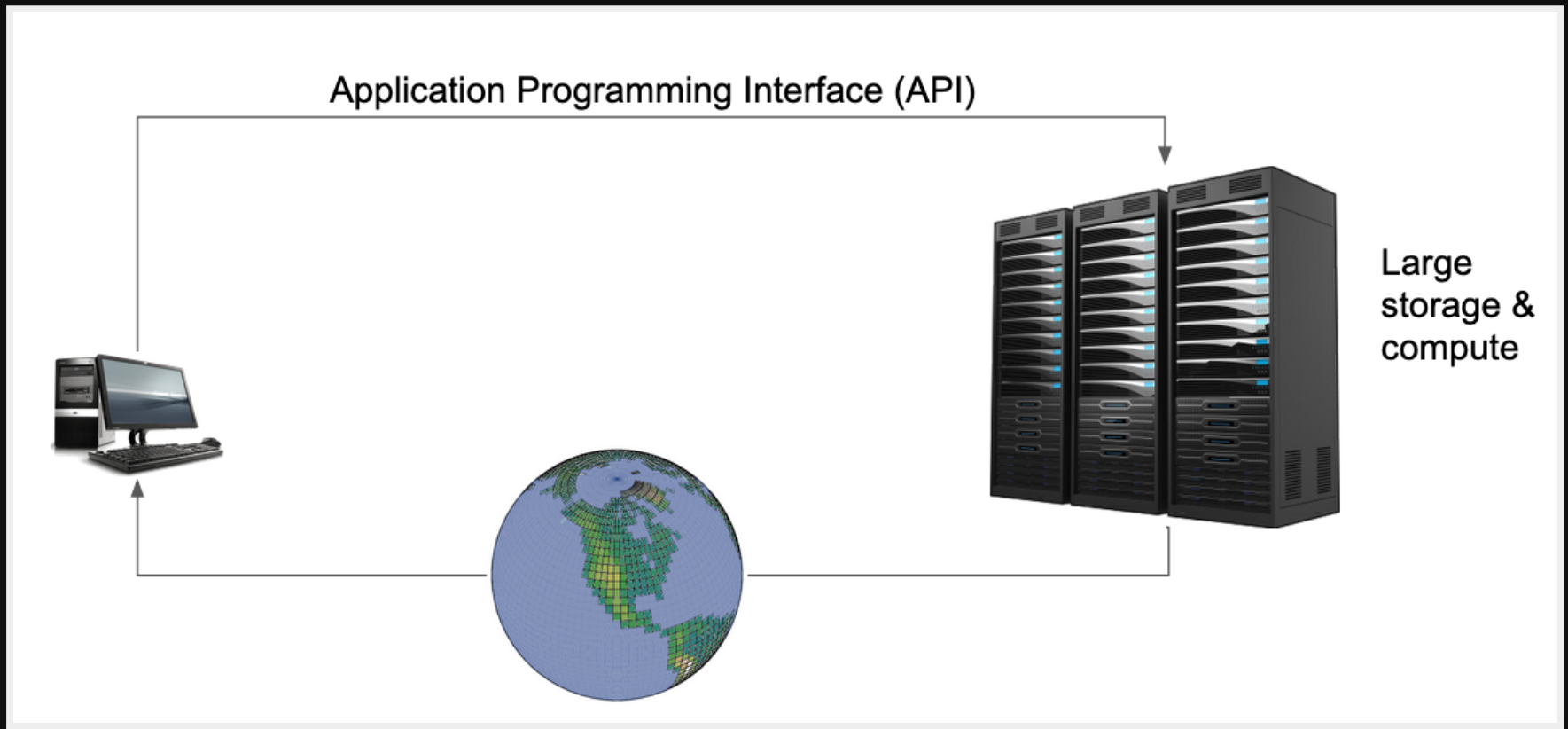
EGU, Vienna, 8 May 2020



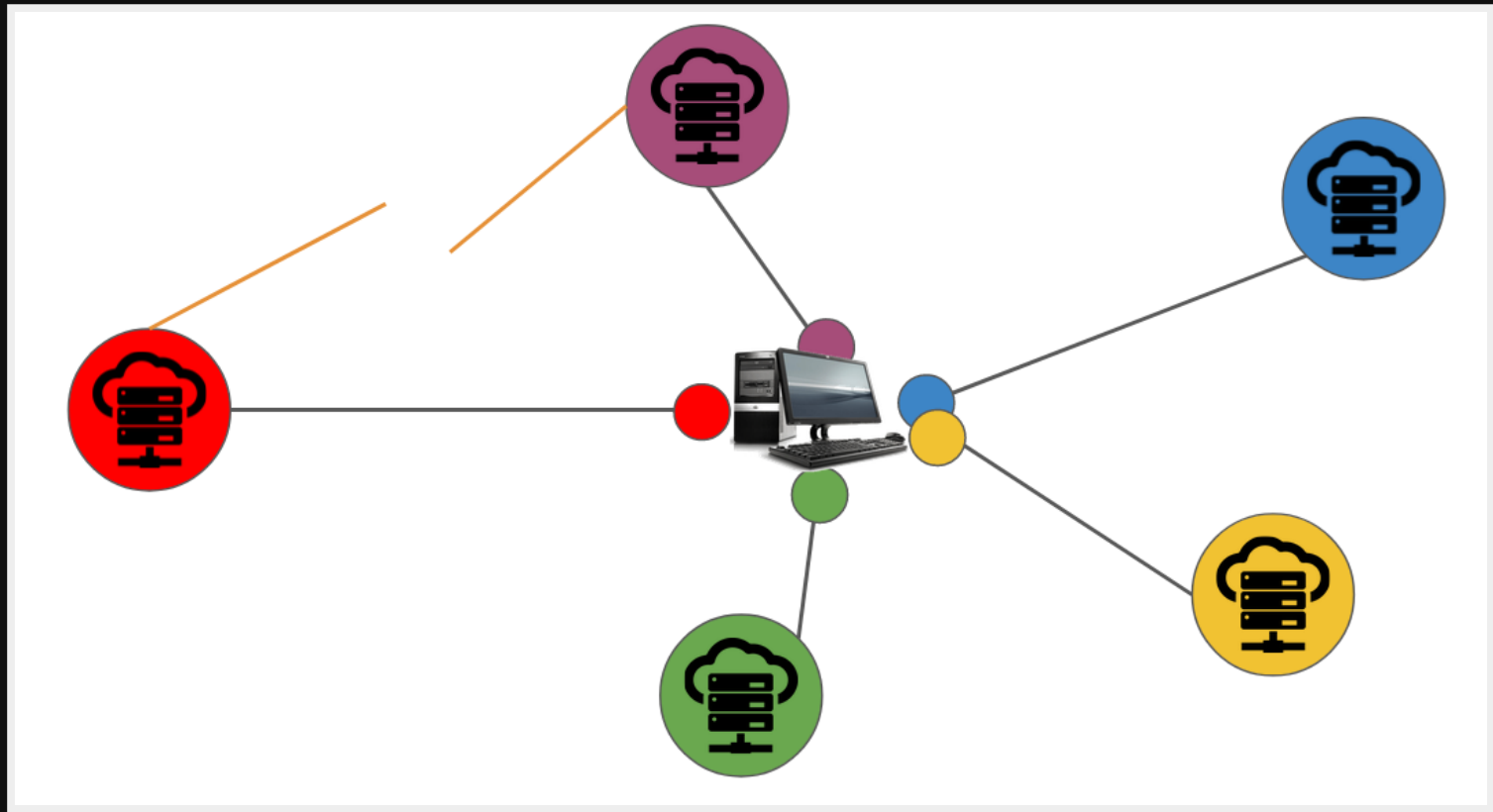
# 2 Minute Introduction

**What is a Web  
Processing Service?**

# Scientific number-crunching is moving into the cloud



# But we could get stuck with multiple APIs and clients



# WPS is an OGC standard for remote processing



- Define inputs and outputs of your *processes* ("functions")
- Like "*Function as a Service*"

# WPS operations

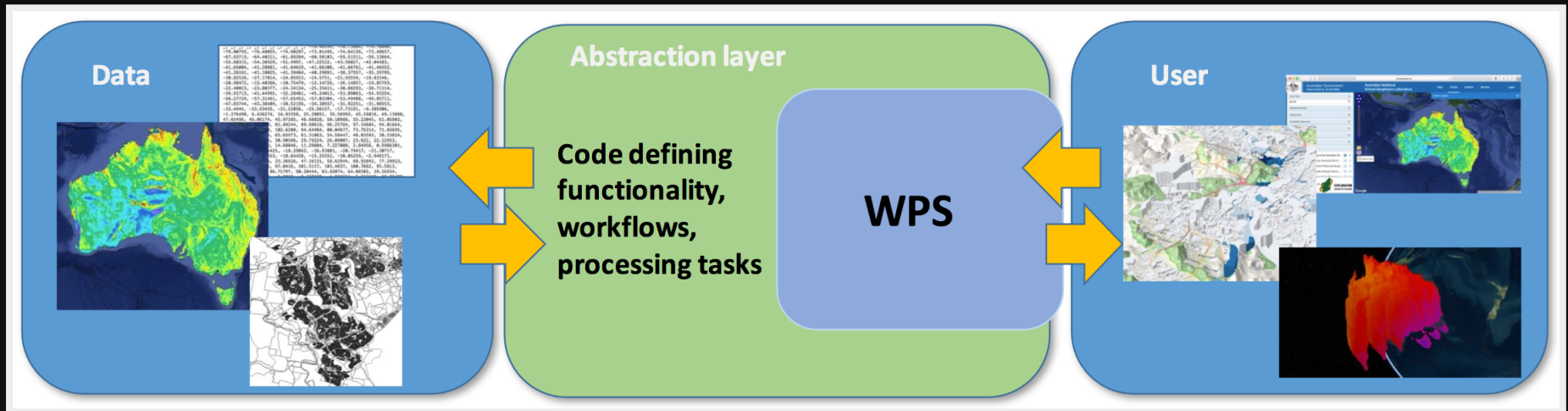
- **GetCapabilities** – List available processes
- **DescribeProcess** – Inputs and outputs of a process
- **Execute** – Launch a process

# Usually it looks like this

```
http://localhost:5000/wps?  
service=WPS&  
version=1.0.0&  
request=Execute&  
identifier=hello&  
DataInputs=name=Stranger
```



# Mostly used by user-friendly clients



Like portals, Jupyter notebooks, ...

# PyWPS



- **PyWPS**: Python implementation of WPS
- Lightweight like a bicycle
- Open Source and active community

# What does Birdhouse provide?

- A template to build your own Web Processing Service
- Tools for automatic deployment
- WPS Client to be used in Jupyter notebooks

# Example

Climate indicators calculation as a service

# Calculate “frost days”

Using **xclim** Python library

```
import xclim
import xarray as xr

# load local data with xarray
tasmin = xr.open_dataset('tasmin.nc')

# calculate frost days
result = xclim.indices.frost_days(tas=tasmin)
```

[Online Notebook](#)

# Calculate “frost days” remotely

Use the Web Processing Service **Finch**

```
# init wps client
from birdy import WPSClient
wps = WPSClient('http://demo/finch/wps')

# use web accessible data
tasmin = "https://demo/thredds/dodsC/tasmin.nc"

# calculate frost days remotely
result = wps.frost_days(tasmin)
```

... and the Birdy WPS client.

**Online Notebook**

# Summary

- *Web Processing Service* is a standard interface for remote processing
- *“Call a function remotely”*
- Birdhouse has tools to build your own Web Processing Service

# Next ...

- Looking at the Birdhouse tools
- A Web Processing Service Example for **Freva**



**Build your own WPS**

# Use a Cookiecutter Template

- **Cookiecutter**: Python tool to create projects from templates
- We have a cookiecutter template for a PyWPS project
- Generated PyWPS project works out of the box

<https://cookiecutter-birdhouse.readthedocs.io/en/latest/>

# Example

```
# Install cookiecutter
$ conda install -c conda-forge cookiecutter

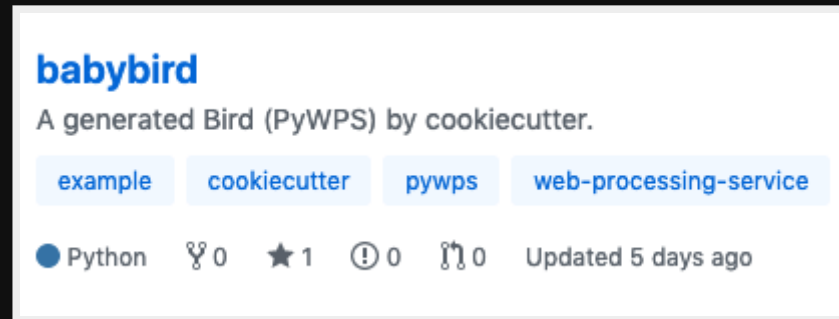
# Run cookiecutter with PyWPS template
$ cookiecutter https://github.com/bird-house/cookiecutter-birdhou

full_name [Full Name]: Daphne du Maurier
github_username [bird-house]: bird-house
project_name [Babybird]: Babybird
project_slug [babybird]: babybird
project_short_description [Short description]: A Web Processing S
version [0.1.0]: 0.1.0
http_port [5000]: 5000
```

Creates a PyWPS project named *babybird*.

# Babybird

Add your new WPS service to GitHub



<https://github.com/bird-house/babybird>

# **Working with the new WPS**

# Install your WPS

```
# Get source from GitHub
$ git clone https://github.com/bird-house/babybird.git
$ cd babybird

# Create a conda environment
$ conda env create -f environment.yml
$ source activate babybird

# Run Python installation
$ pip install -e .[dev]
OR
$ make develop
```

- Use **Conda** to manage dependencies
- Normal Python installation

# Start the Service

```
# start service with custom config
$ make start -c custom.cfg

# run GetCapabilities request
$ curl -o caps.xml \
  "http://localhost:5000/wps?service=WPS&request=GetCapabilities"

# check logs
$ tail -f pywps.log
```

No additional installation steps necessary to run service (using **Werkzeug** library)

# Try with Birdy as WPS client

```
from birdy import WPSClient
babybird = WPSClient(url='http://localhost:5000/wps')
output = babybird.hello(name='Stranger')
print(output.get())
'Hello Stranger'
```

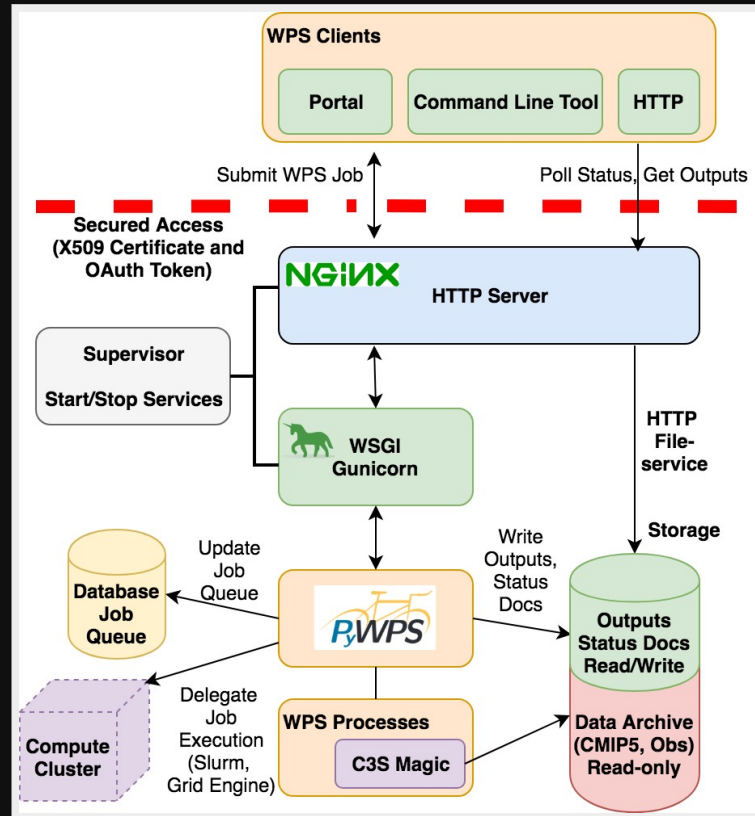
- Can be used in Jupyter Notebooks
- WPS functions feel like normal Python functions

<https://birdy.readthedocs.io>



**Deploy your WPS**

# PyWPS full-stack



Need several other components to run in production: Nginx, Postgres, ...

# Deploy with Ansible

```
# Get Ansible playbook
$ git clone \
  https://github.com/bird-house/ansible-wps-playbook.git
$ cd ansible-wps-playbook

# Edit config: point it to your WPS on GitHub
$ vim custom.yml

# Run playbook
$ ansible-playbook -c local playbook.yml
```

- Use **Ansible** playbook for full-stack deployment of PyWPS
- Ansible: language for IT automation

# Example

- **Freva**: Evaluation System for Climate Data.
- Evaluation processes can be plugged into the system.
- Command line and web portal access.

Remote service access could be provided using a Web Processing Service.

# Freva: GetCapabilities

Show available plugins

```
$ freva --plugin
MoviePlotter: Plots 2D lon/lat movies in GIF format
MurCSS: Calculates the MSESS ...
PCA: Principal Component Analysis
```

*GetCapabilities* call in a Web Processing Service

```
http://demo/freva/wps?
  service=WPS&
  request=GetCapabilities
```

# Freva: DescribeProcess

Show details of MoviePlotter plugin

```
$ freva --plugin MoviePlotter --help
MoviePlotter (v1.0.0):
  Plots 2D lon/lat movies in GIF format

Options:
input      NetCDF file(s) to be plotted.
```

*DescribeProcess* call in a Web Processing Service

```
http://demo/freva/wps?
service=WPS&
request=DescribeProcess&
identifier=movieplotter&
```

# Freva: Execute

## Run MoviePlotter

```
$ freva --plugin movieplotter input=/path/to/tasmax.nc  
Searching Files  
Remapping Files  
Calculating ...  
Finished.
```

## *Execute* call in a Web Processing Service

```
http://demo/freva/wps?  
service=WPS&  
request=Execute&  
identifier=movieplotter&  
DataInputs=input=http://demo/thredds/dodsC/tasmax.nc
```

# Freva: Web Processing Service

Call Freva plugins via Web Processing Service

```
from birdy import WPSClient
wps = WPSClient('http://demo/freva/wps')
# show available plugins
wps?
# show movieplotter details
wps.movieplotter?
# run movieplotter
tasmax = "https://demo/thredds/dodsC/tasmax.nc"
result = wps.movieplotter(tasmax)
```

[Online Notebook](#)



# Links

- Website: <http://bird-house.github.io/>
- PyWPS: <https://pywps.org/>
- Finch: <https://finch.readthedocs.io/en/latest/>
- Freva: <https://www-miklip.dkrz.de/>

# Thank You

- Carsten Ehbrecht, **DKRZ**, Germany
- Stephan Kindermann, **DKRZ**, Germany
- Ag Stephens, **CEDA/STFC**, UK
- David Huard, **Ouranos**, CA

**Extra slides**

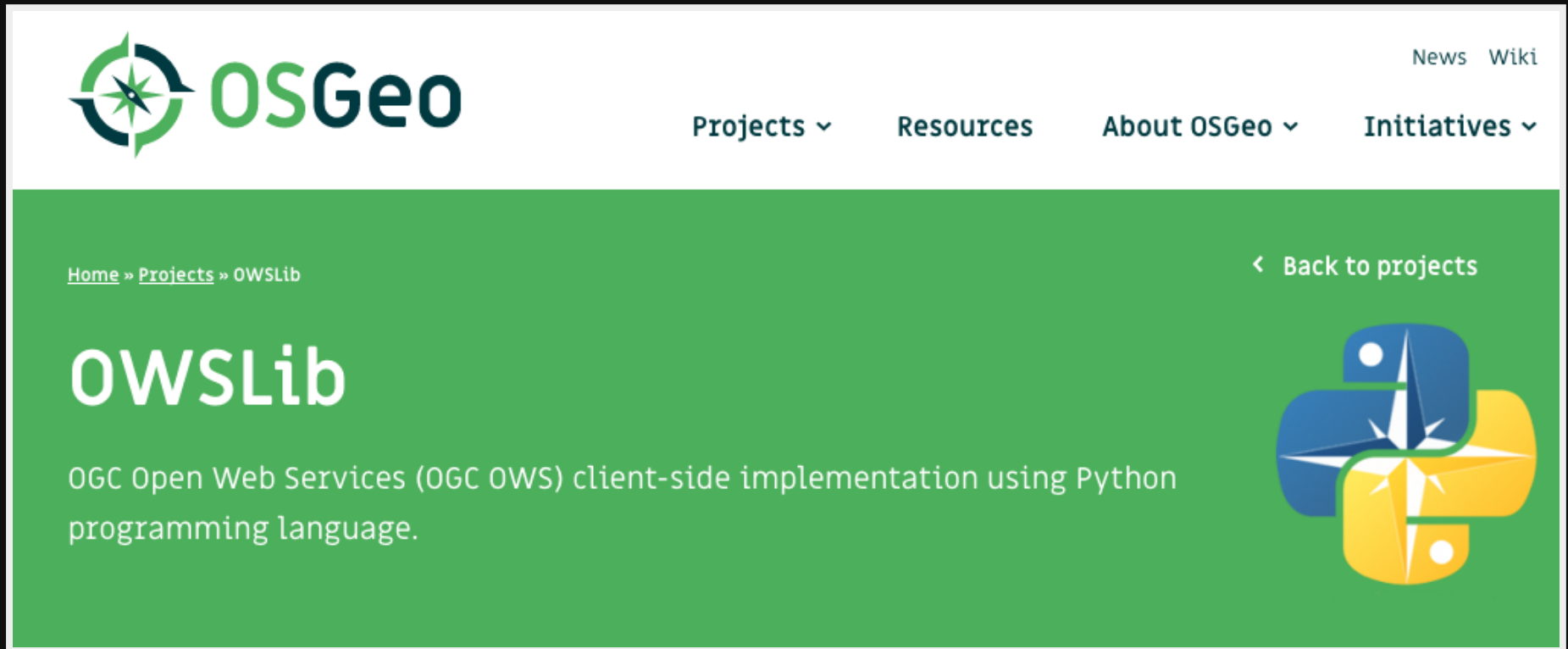
# Use the WPS with URL requests

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

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

```
http://localhost:5000/wps?service=WPS&version=1.0.0&  
request=Execute&  
identifier=hello&  
DataInputs=name=Stranger
```

# OWSLib - Client



The screenshot shows the OSGeo website header with the logo on the left and navigation links for News, Wiki, Projects, Resources, About OSGeo, and Initiatives on the right. Below the header, a green banner contains a breadcrumb trail (Home » Projects » OWSLib), a 'Back to projects' link, the title 'OWSLib', and a description: 'OGC Open Web Services (OGC OWS) client-side implementation using Python programming language.' To the right of the text is a logo combining the Python logo and the OSGeo compass logo.

OSGeo

News Wiki


Projects ▾ Resources About OSGeo ▾ Initiatives ▾

[Home](#) » [Projects](#) » OWSLib

[← Back to projects](#)

## OWSLib

OGC Open Web Services (OGC OWS) client-side implementation using Python programming language.



- Python client-side implementation of WPS, WMS, WCS and more

# Tests included

```
$ make test # quick  
$ make test-all # slow, online  
$ make lint # codestyle checks
```

# Birdy command line tool

```
# Set URL to WPS
$ export WPS_SERVICE=http://localhost:5000/wps
# GetCapabilities
$ birdy -h
# DescribeProcess: hello
$ birdy hello -h
# Execute: hello
$ birdy hello --name Stranger
'Hello Stranger'
```

Using the Python OWSLib library for WPS

# Modify your WPS

```
class SimplePlot(Process):
    def __init__(self):
        inputs = [
            ComplexInput('dataset', 'Dataset', supported_formats=[Format('application/x-netcdf')],
                        default=AIR_DS,
                        abstract='Example: {0}'.format(AIR_DS)),
            LiteralInput('variable', 'Variable', data_type='string',
                        default='air',
                        abstract='Enter the variable name.'),
        ]
        outputs = [
            ComplexOutput('output', 'Simple Plot', supported_formats=[Format('image/png')],
                        as_reference=True),
        ]
```

- Create a Python class
- Define the input and output parameters.
- Implement a *handler* method with the process code.



# Test with Vagrant

Deploy with Ansible into a test virtual machine set-up by **Vagrant**

```
# Use Ansible playbook
$ cd ansible-wps-playbook

# use vagrant config
$ cp etc/sample-vagrant.yml custom.yml

# Vagrant starts a VM and deploys with Ansible
$ vagrant up
```

# Deploy as docker container

Dockerfile was generated by the cookiecutter

```
# build
$ docker build -t bird-house/babybird .
# run
$ docker run -p 5000:5000 bird-house/babybird
# test it
http://localhost:5000/wps?request=GetCapabilities&service=WPS
```

# Security

