

Web Processing Service

Deploying a WPS for Copernicus Climate Data Store

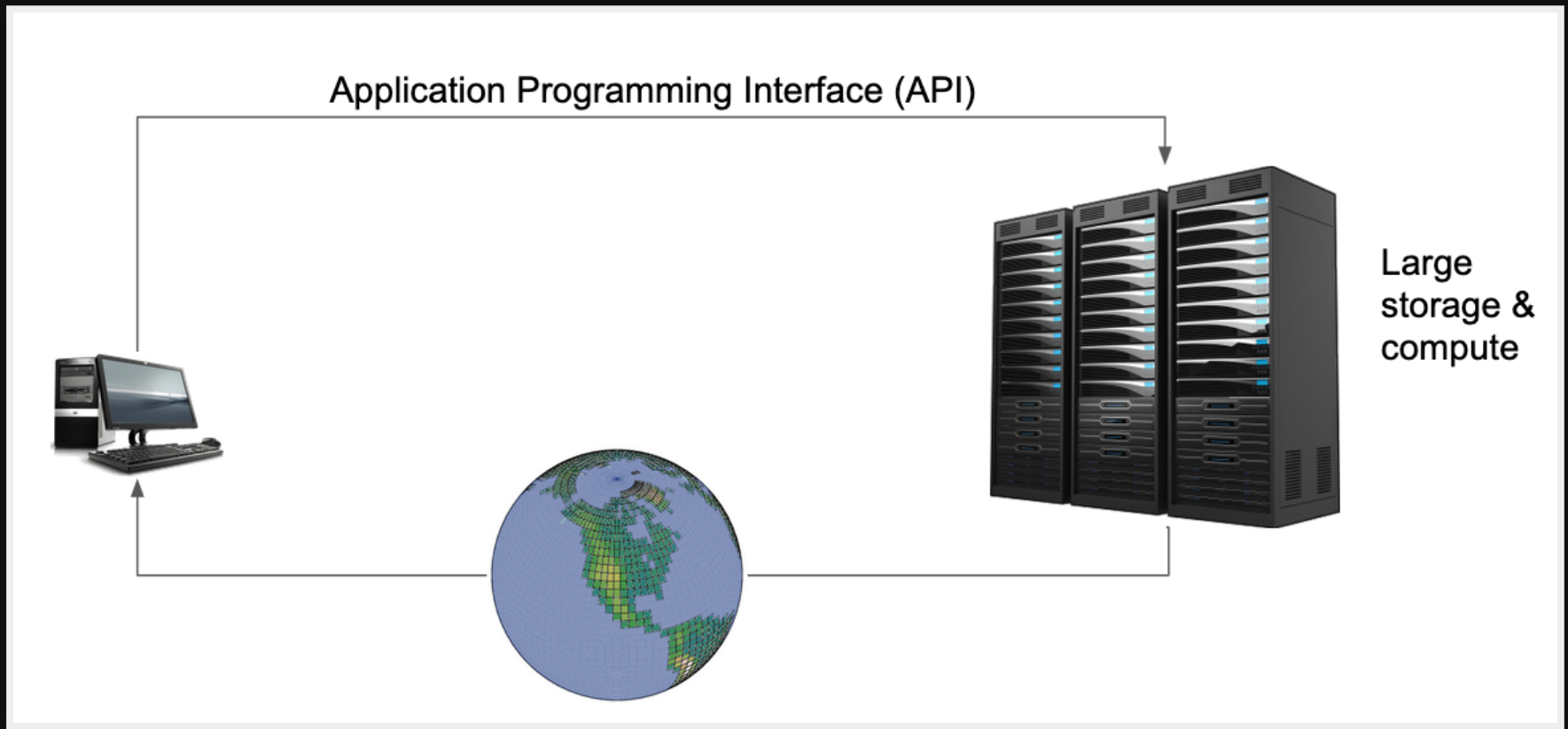
Copernicus CORDEX4CDS Meeting
DMI, Copenhagen, 23-24 September 2019

Table of Contents

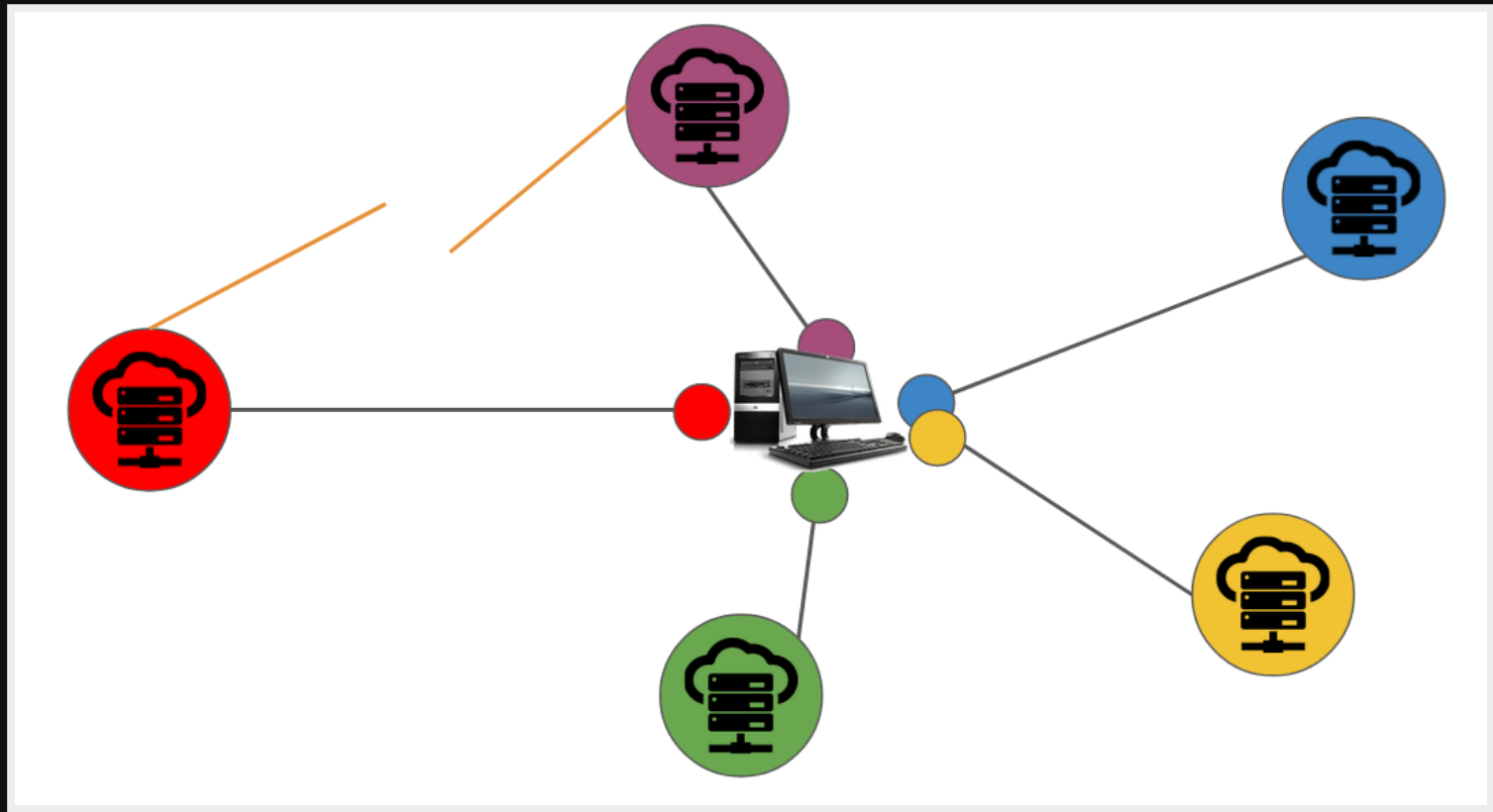
- What is a Web Processing Service?
- Build your own WPS
- Using your WPS
- Deploy your WPS

What is a WPS?

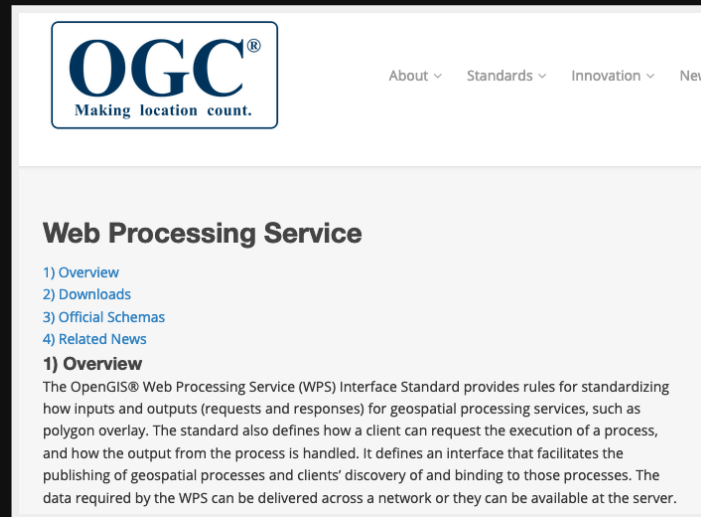
Scientific number-crunching is moving into the cloud



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*"

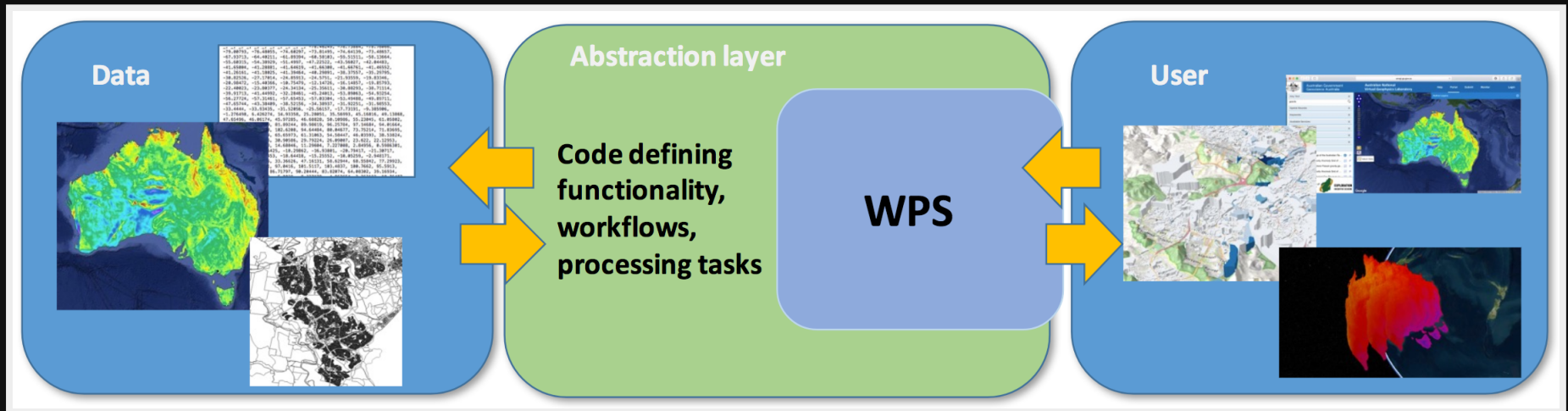
WPS operations

- **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 - Server

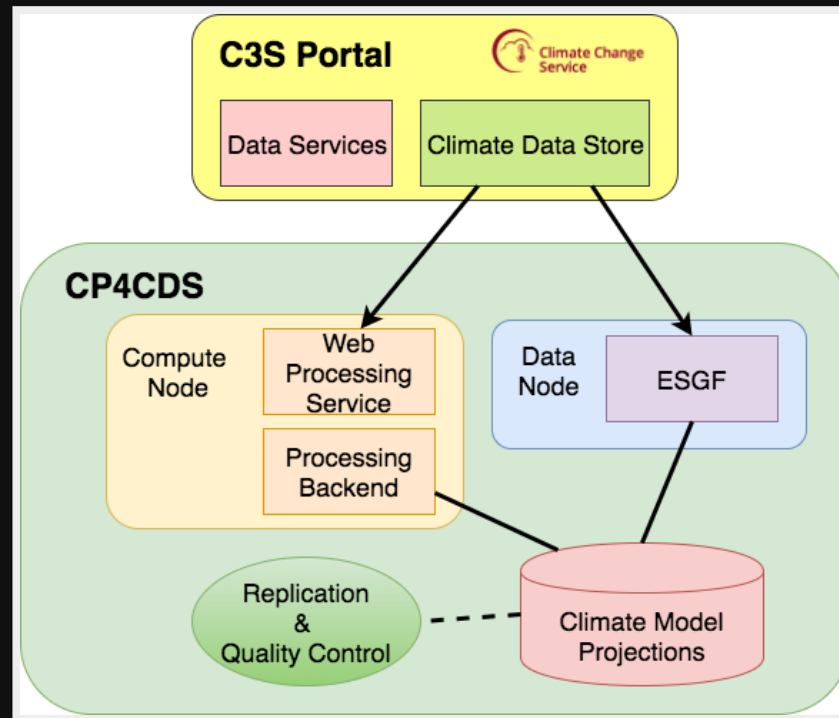


- Python implementation of WPS
- Like a bicycle easy to use
- Open Source and active community

What is the Goal?

- Make climate data accessible to a wide audience
- Use a standards based compute service
- Example: data-reduction as a service next to a large data data pool (CORDEX)

WPS for Climate Data Store



- Climate projections for Climate Data Store
- ESGF data nodes for CMIP5, CORDEX
- WPS for compute nodes

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

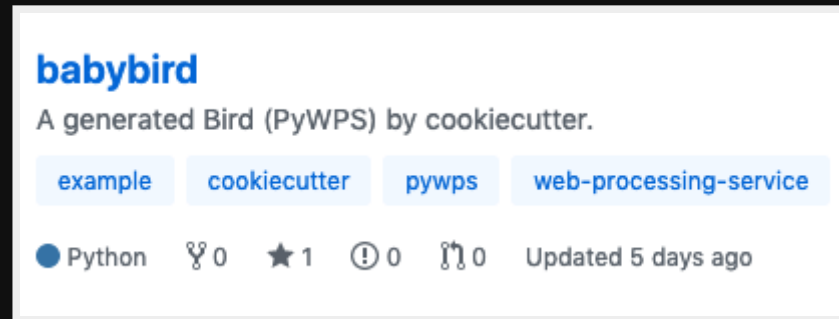
# Run cookiecutter with PyWPS template
$ cookiecutter https://github.com/bird-house/cookiecutter-birdhou

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

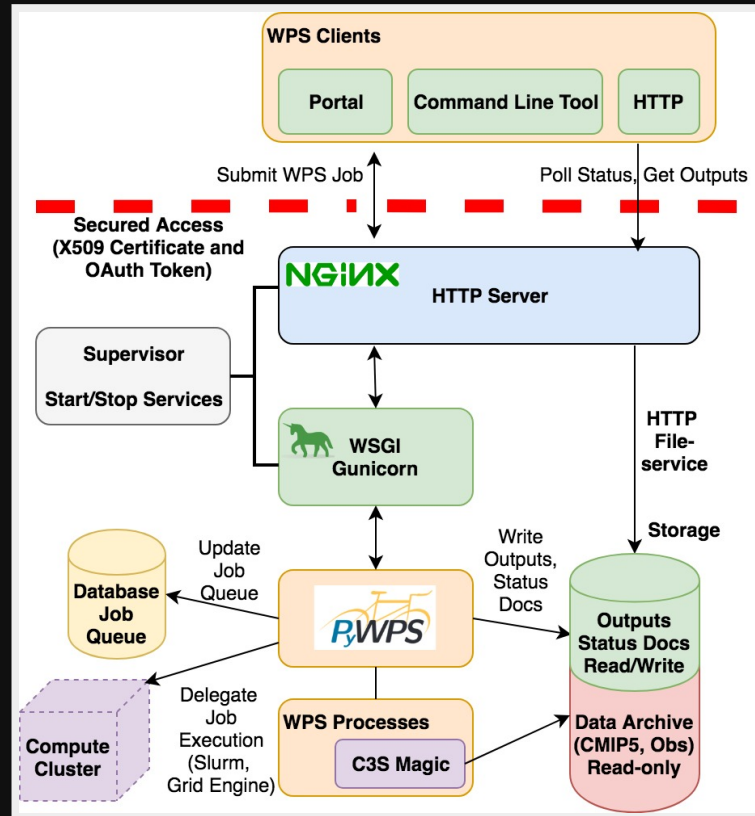
```
from birdy import WPSClient
babybird = WPSClient(url='http://localhost:5000/wps')
output = babybird.hello(name='Stranger')
print(output.get())
'Hello Stranger'
```

- 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

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
```


Summary

- WPS is standard interface for remote processing
- Use Cookiecutter template to create a new WPS project
- New WPS is ready to use without extra installation steps
- Ansible can be used for production deployment

Links

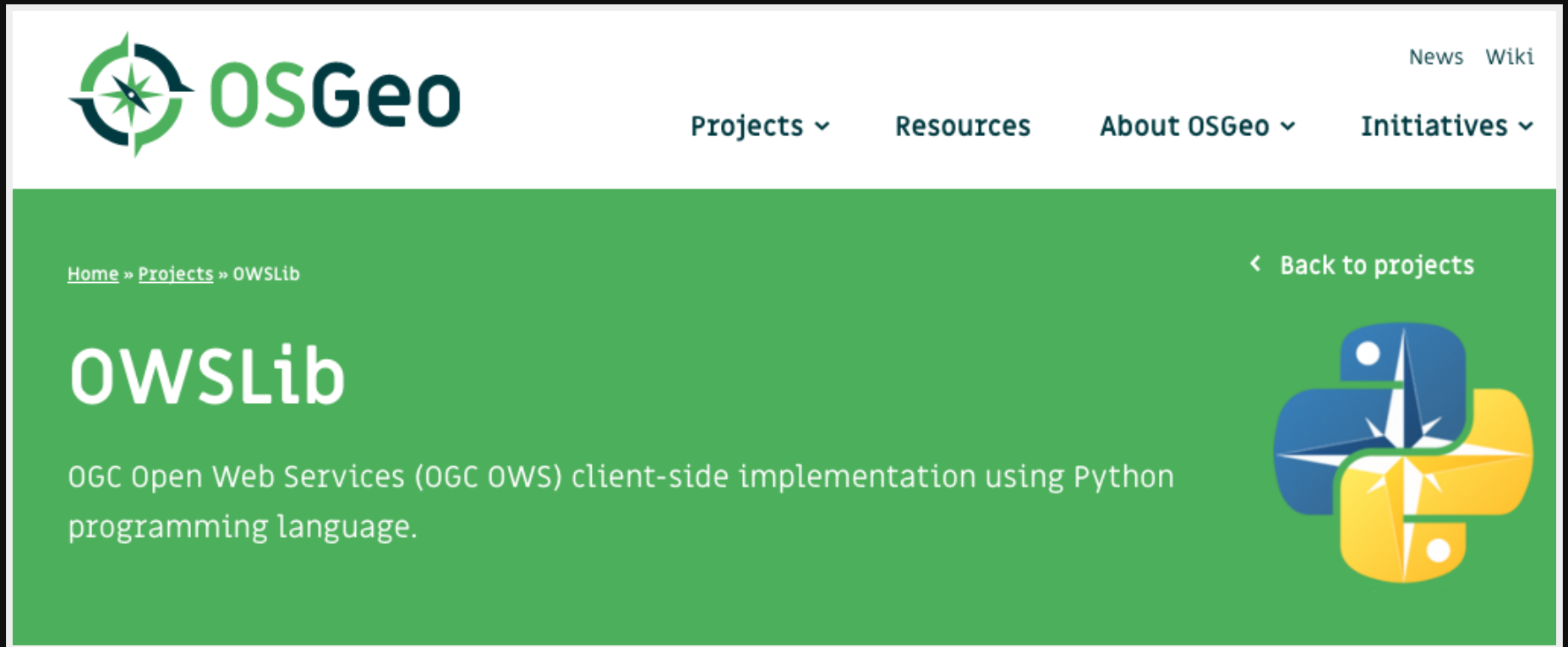
- Website: <http://bird-house.github.io/>
- Development: <https://github.com/bird-house>
- CP4CDS: <https://cp4cds.github.io/>
- Presentation: <https://github.com/cehbrecht/wps-talk-copernicus-cordex-dmi-meeting-2019>
- Poster:
<https://github.com/cehbrecht/copernicus-poster-egu-2018/blob/master/copernicus-poster-egu-2018.pdf>

Thank You

- Pierre Logerais, IPSL
- Katharina Berger, DKRZ
- Carsten Ehbrecht, DKRZ

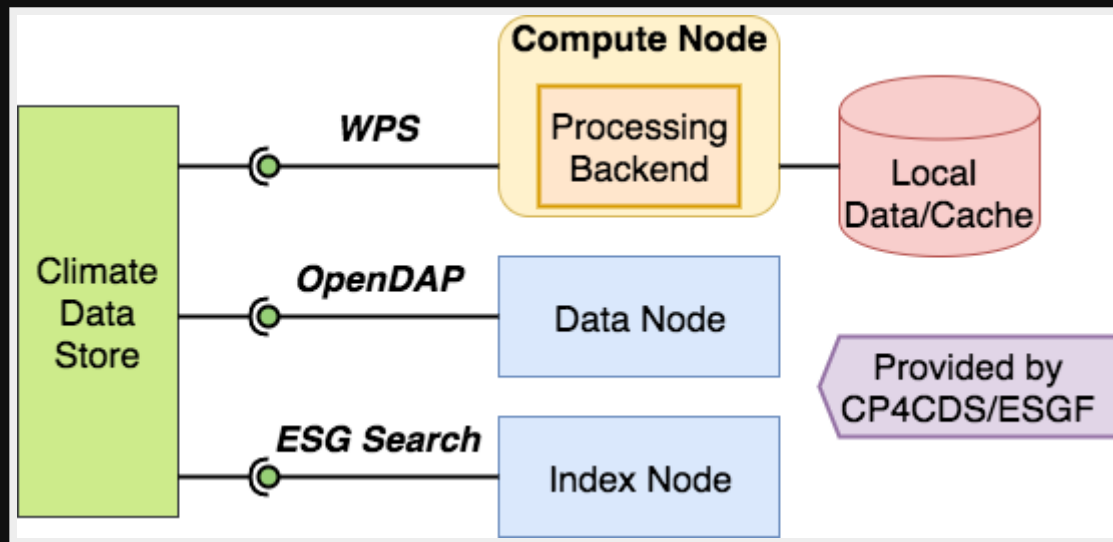
Extra slides

OWSLib - Client



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

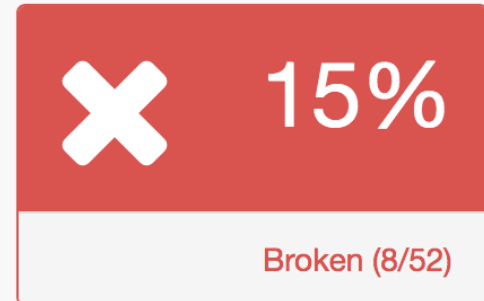
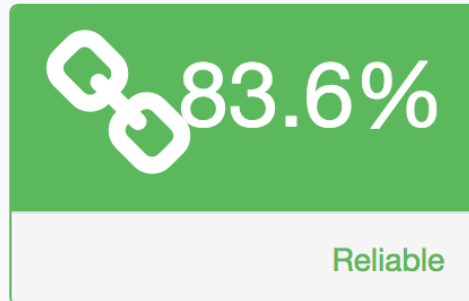
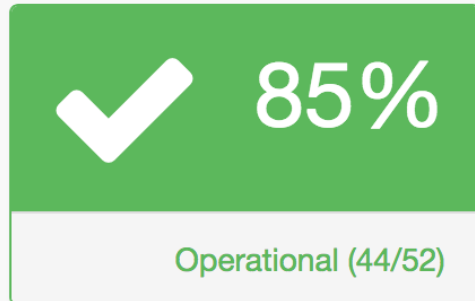
CP4CDS Interfaces



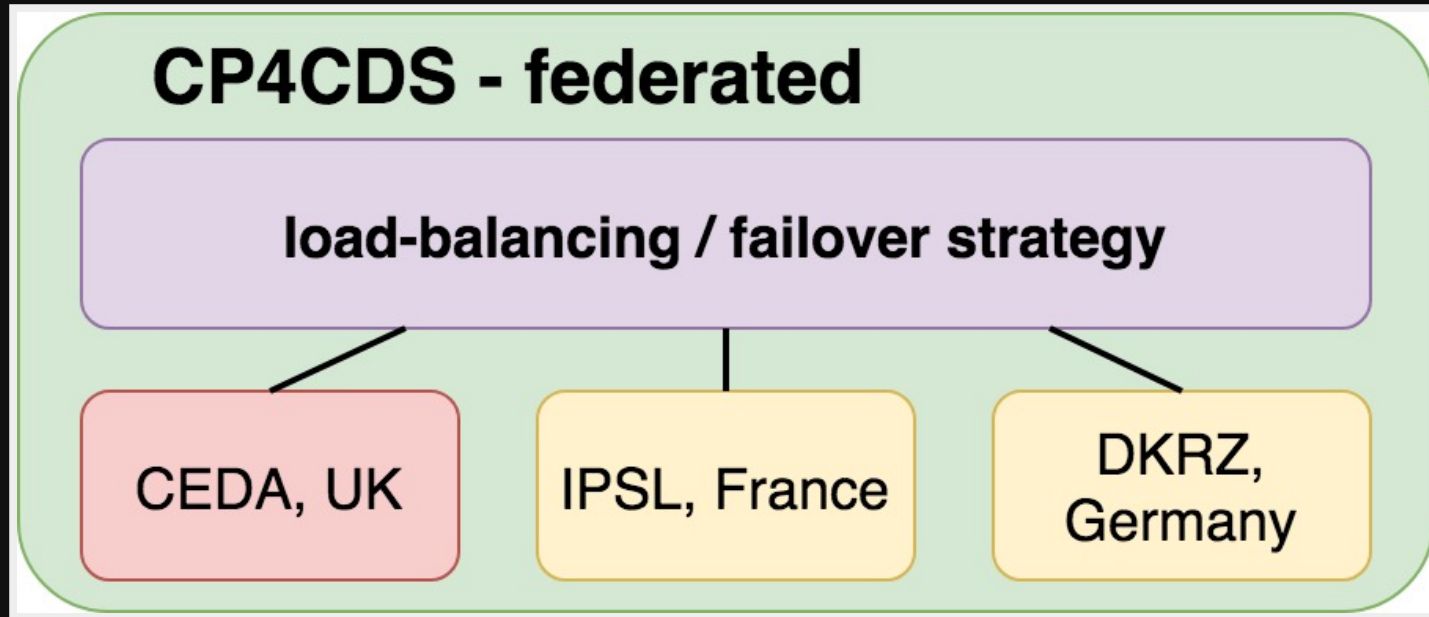
Uptime 99%

Dashboard

Monitoring Period: 2018-07-09T08:00:02Z - 2018-07-17T14:00:02Z

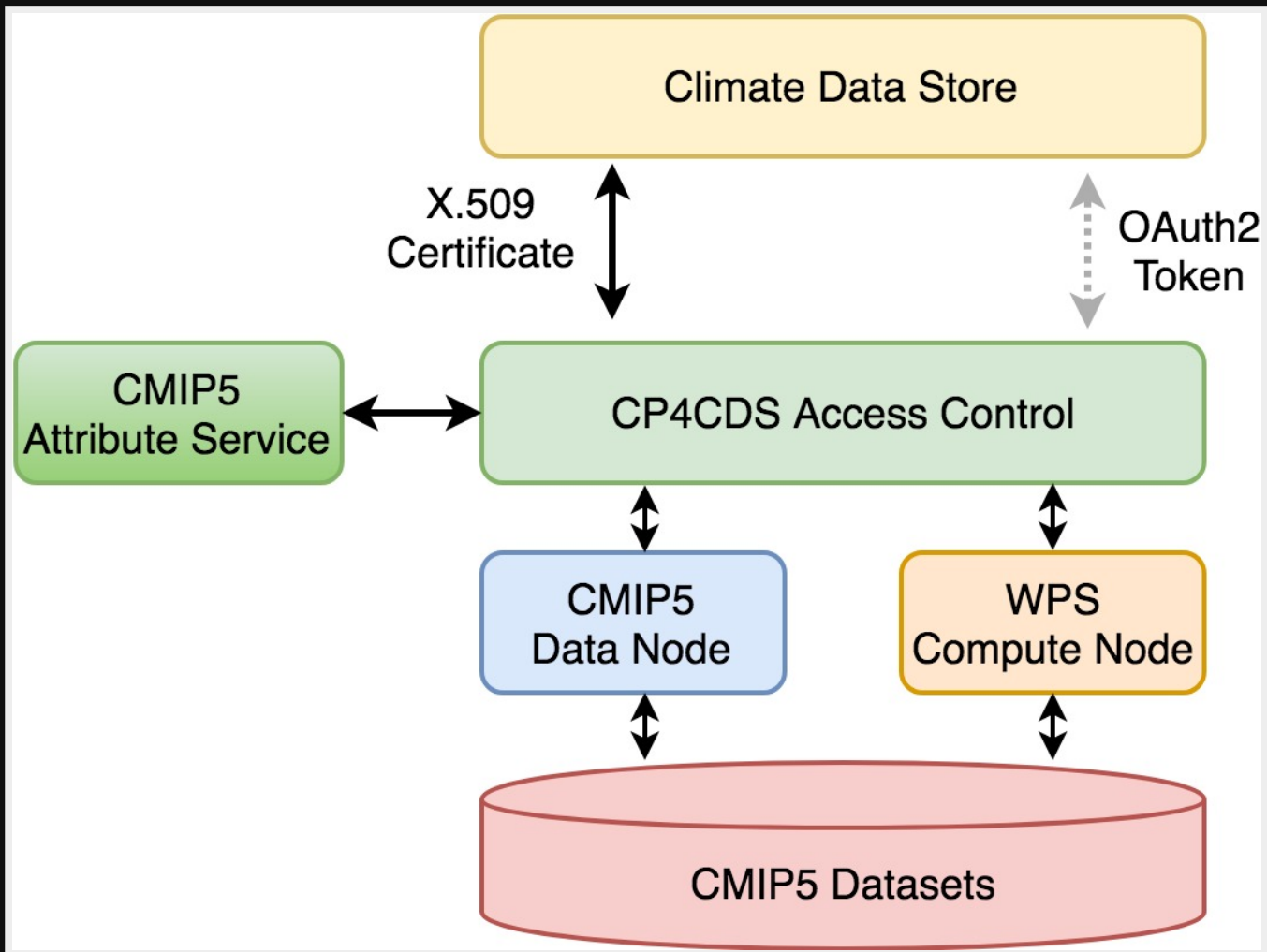


Can't do it alone



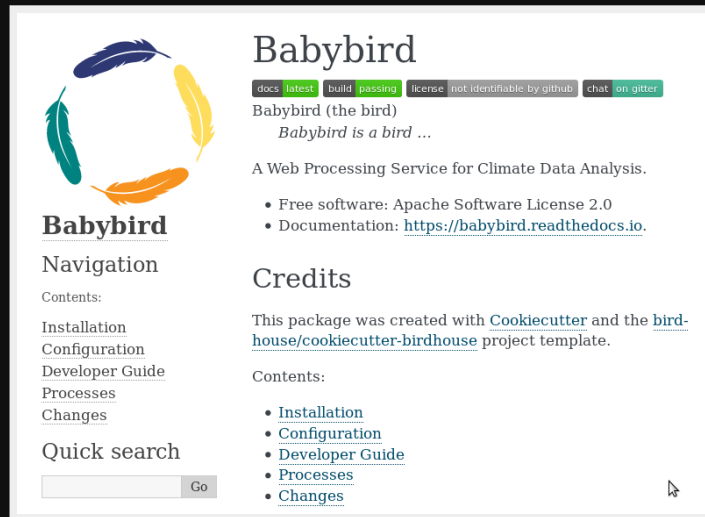
- Cloud based on Amazon Web Services
- Failover strategy for resilience

Security



Documentation

Add your WPS documentation to ReadTheDocs



<https://babybird.readthedocs.io/en/latest/index.html>

Tests included

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

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
```


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'
```

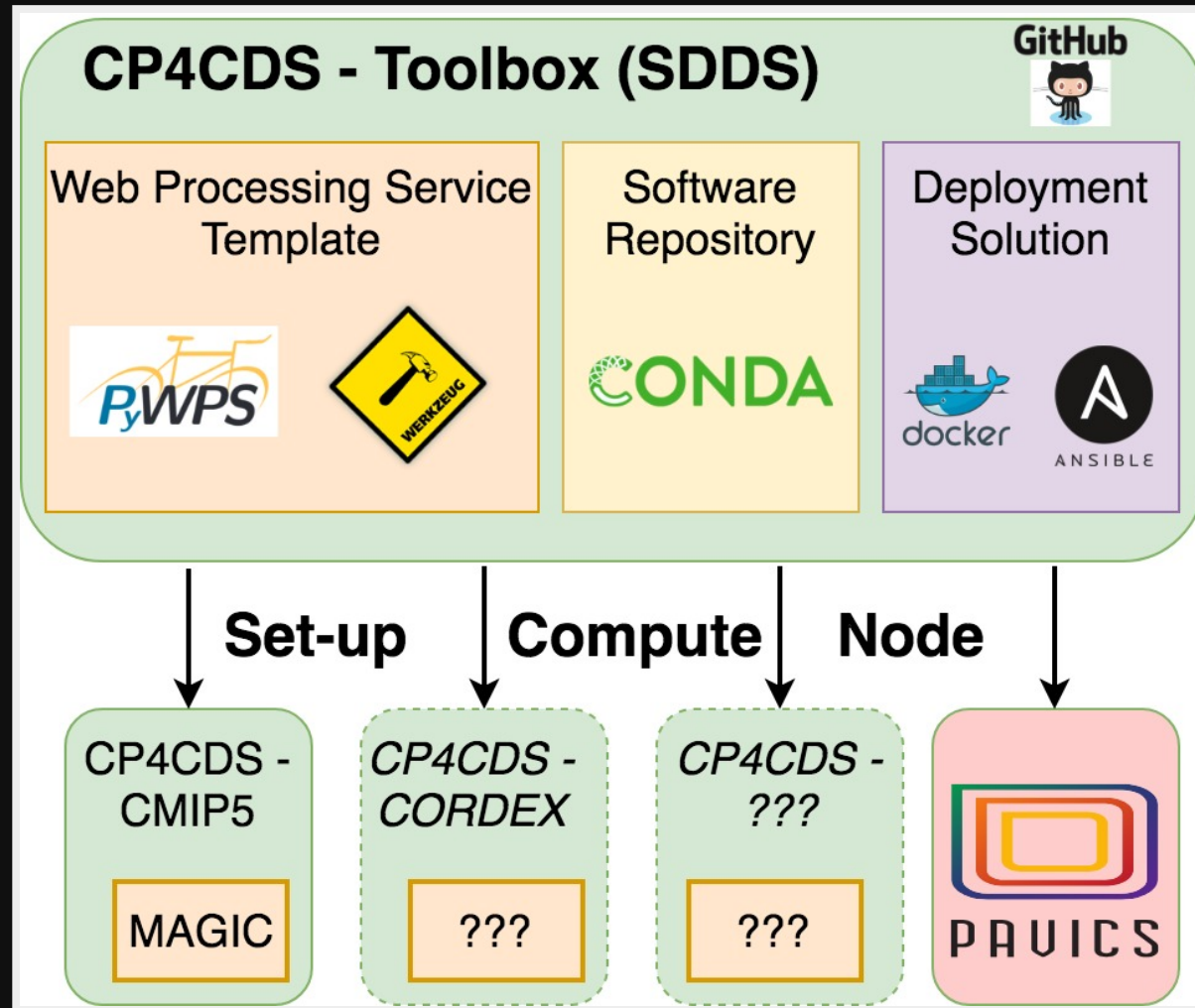
Using the Python OWSLib library for WPS

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.

Software Deployment Solution



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
```