Birdhouse Documentation

Release 0.7.0

Birdhouse

Apr 01, 2020

CONTENTS

Birdhouse is a GitHub organization comprised of Python projects related to Web Processing Services to support climate data analysis.

The full documentation is available on ReadTheDocs and in the *docs/* folder.

CHAPTER

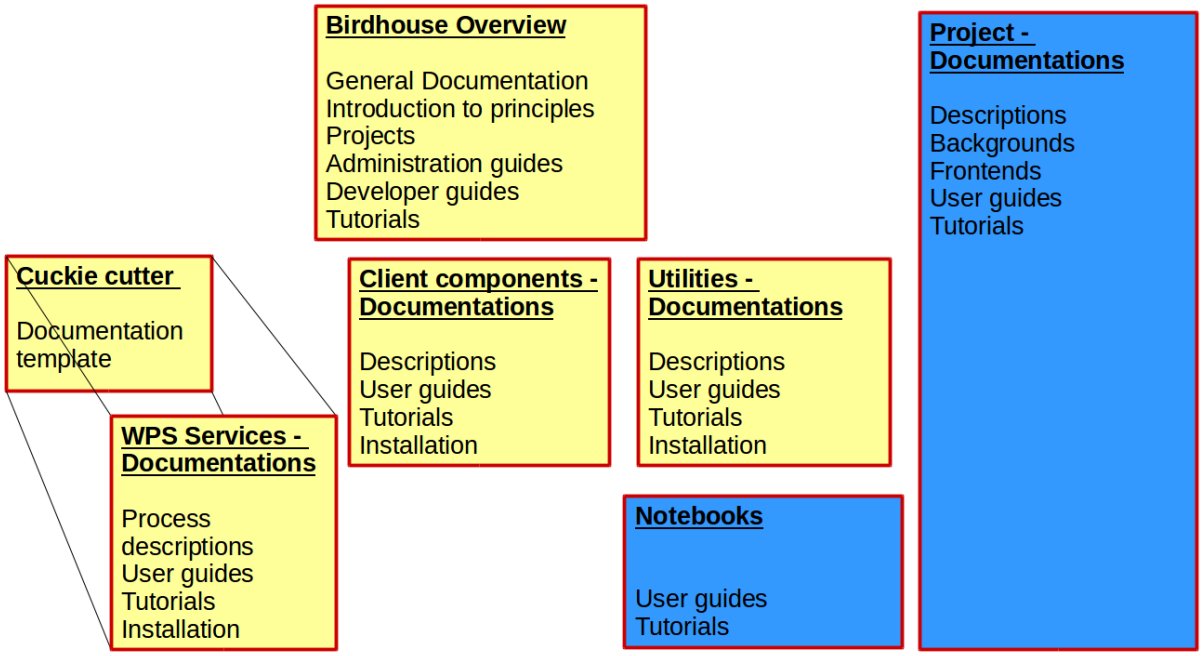
ONE

OVERVIEW

Birdhouse is a collaborative project open for the community to participate. It is a software framework containing a collection of Web Processing Service (WPS). The deployed algorithms are focusing on Earth Systems and environmental data processing with the philosophy of streamlining the software development and deployment. By supporting climate, earth observation and biodiversity data and processes, Birdhouse can be used in a wide array of Earth sciences projects and workflows. The core benefit of this project is to allow the seamless use of climate services developed by a diverse network of national meteorological offices, regional climate service providers, academics, not-for-profit research centers and private industry. As governments move toward open-data policies, there will be a need for analytical services that extract value out of the deluge of information. Using an interoperable software architecture, institutions can provide both data and services allowing users to process the data remotely from a laptop, instead of having to acquire and maintain large storage infrastructures.

1.1 Documentation structure

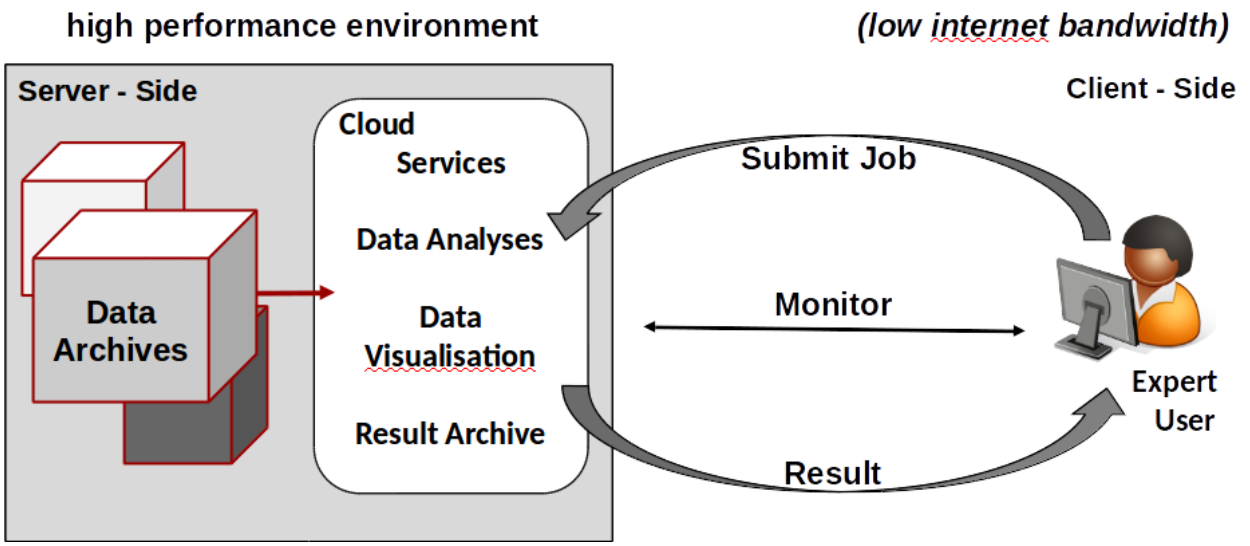
The birdhouse documentation reflects the fact that it is an assemblage of independent software components. It's therefore organized according to the *birdhouse framework structure*. Birdhouse is being used by international working groups who deploy subsets of components tailored to their user base. The following graph shows an overview of the documentation's organization:



1.2 What is WPS?

Geographic Information Processing for the Web *The Web Processing Service (WPS) offers a simple web-based method of finding, accessing, and using all kinds of calculations and models.*

A WPS is a technical solution (WPS Concepts) in which processes are hosted on a server and accessed over the web (Fig. 1). These processes con- form to a standardized format, ensuring that they follow the principle of reusable design: they can be instantiated multiple times for different input arguments or data sources, customized following the same structure to handle new inputs, and are modular, hence can be combined to form new processes. In addition, a WPS can be installed close to the data to enable processing directly out of the archive. A WPS can also be linked to a theoretically limit- less combination of several other WPSs, or generally OpenGIS Web Services (OWS). Our understanding of process is used in the same sense as in the OGC standard: 'for any algorithm, calculation or model that either generates new data or trans- forms some input data into output data'. A submitted process is a job. A service provides a collection of processes containing scientific methods that focus on climate impact and extreme weather events. A combination of processes is called a workflow, and a collection of WPS-related software compartments is a framework. WPS divides the operation into server and client side, with appropriate security in between to avoid misuse.

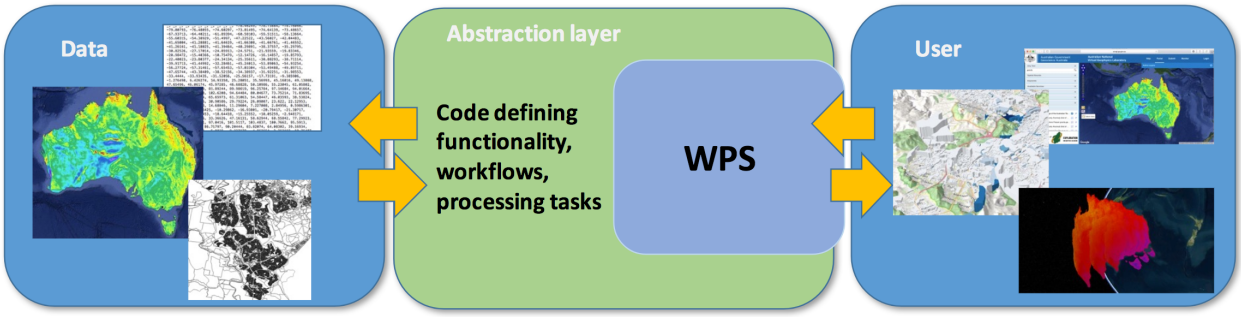


Note: Read the documentation on Geographic Information Processing for the Web

1.3 WPS Use Case

Todo: needs to be updated.

A user runs WPS processes *remotely* on a machine with direct access to climate data archives.



CHAPTER

TWO

BIRDHOUSE FRAMEWORK

Birdhouse is organized in separate stand-alone software components. Most components are named after birds, which gives the project its name birdhouse. The components can be categorized into *Client Side Components*, i.e. tools for end-users, and *Server Side Components*, i.e. back-end elements of the architecture.

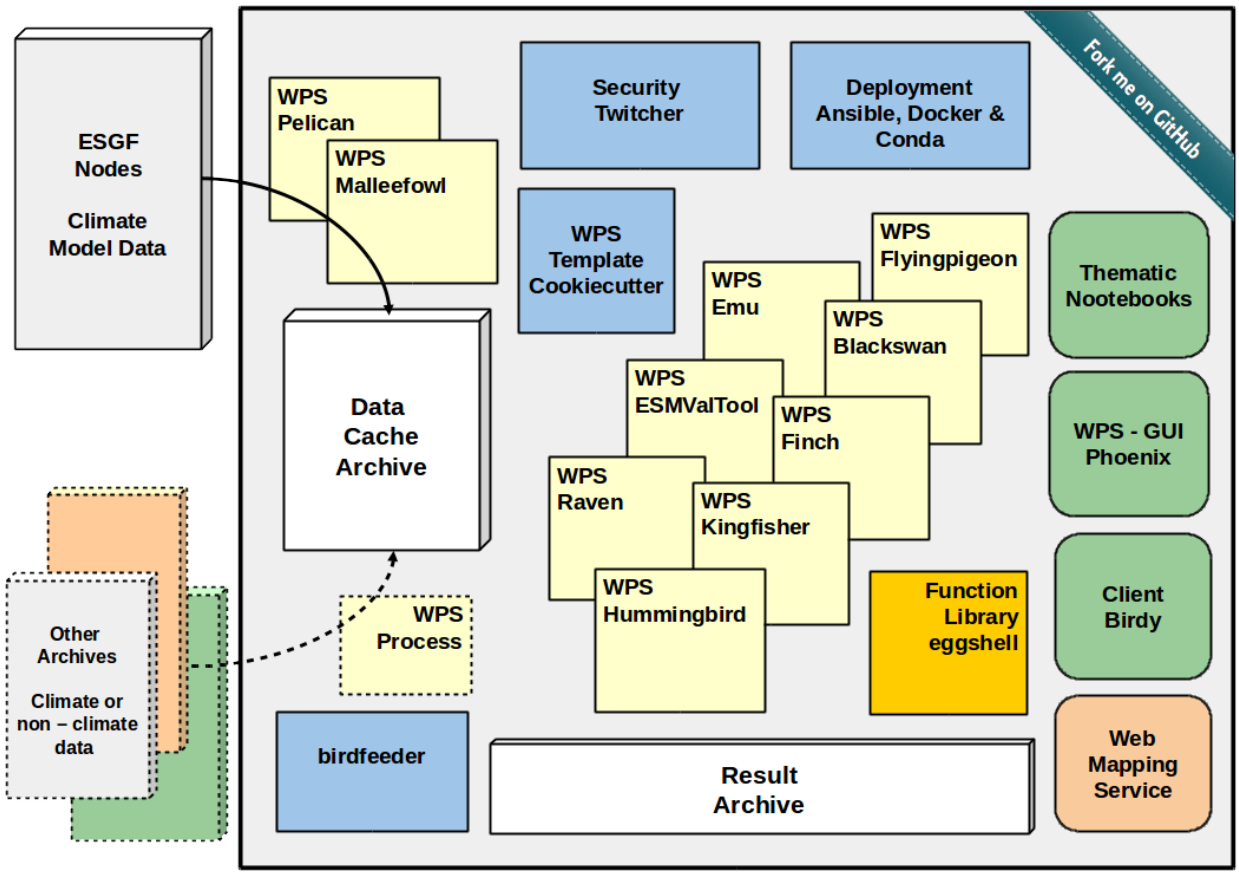
2.1 Framework structure

There are several WPS services. Malleefowl is the main one for the Phoenix client. Malleefowl is used to search, download (with caching) ESGF data and to retrieve certificates. Malleefowl has also a workflow engine (dispel4py) to chain WPS processes.

The results of the WPS processes are stored on the file system and are accessible via URL (with a token id). Results can be shown on a Map using a Web Mapping Service (ncWMS, adagucserver). The PyCSW Catalog Service is used to register WPS services and also to publish WPS outputs. Published results in the PyCSW can also used as input source for processes again.

ESGF is currently the main climate data resource (but more resources are possible). ESGF Solr-index is used to find ESGF data. The ESGF identity provider with OpenIDs and X509 certificate is used for authentication.

WPS serivces can be accessed through web-applications like Phoenix or from scripts.



Note: See also the *Publications and Presentations* for more information and details.

2.2 Client Side Components

- Phoenix: a web-based WPS client with ESGF data access
- Birdy: a WPS command line client and native library

2.3 Server Side Components

WPS services for climate data analysis:

- Emu: some example WPS processes for demo
- Flyingpigeon: Testbed for new process development
- Black Swan: services for the extreme weather event assessments
- Hummingbird: provides cdo and compliance-checker as a service
- Finch: services for climate indices calculation

- Pelican: Supporting ESGF compute API
- Kingfisher: Services for Earth-Observation data analysis

Many climate analysis operations are implemented using OpenClimateGIS including the python package icclim.

Supporting Services and libraries:

- Twitcher: an OWS Security Proxy
- Malleefowl: access to climate data (ESGF, ...) as a service
- Eggshell: provides common functionallity for Birdhouse WPS services

You can find the source code of all birdhouse components on GitHub. Docker images with birdhouse components are on Docker Hub

2.4 Files and Folders

This is an overview of folder structure and important files for *administration of a server-side* birdhouse ecosystem.

It is recommended to clone the separated WPS services (birds) into one top level folder like:

```
$ ~/birdhouse/emu
$ ~/birdhouse/pyramid-pheonix
$ ~/birdhouse/finch
$ ~/birdhouse/malleefowl
...
```

The dependencies of each bird is deployed as *conda environment* and per default located at:

```
$ ~/.conda/envs/
```

The environment of a bird is defined in *./{birdname}/environment.yml*.

Process descriptions are placed in *./{birdname}/{birdname}/processes/* while modules designed and used for the service are situated in *./{birdname}/{birdname}/*. Here are also static data like shapefiles, templates or additional data used by the processes.

```
$ ./{birdname}/{birdname}/data/shapefiles
$ ./{birdname}/{birdname}/templates
```

Each birdhouse compartment has a documentation build with *Sphinx* and the corresponding files are situated in

```
$ ./{birdname}/docs
```

When running a service, files and folders for input data, result storage, file cache of simply logfiles are defined in the *./{birdname}/.config.cfg*. Default configuration is defined in *./{birdname}/{birdname}/default.cfg* as well as an example can be found in *~./{birdname}/etc*. For more options of configuration see the pywps configuration instructions

For development and deployment testing the installations be checked running tests (*make test*). Test descriptions testdata are situated in:

```
$ ./{birdname}/tests
$ ./{birdname}/tests/testdata
```

CHAPTER

THREE

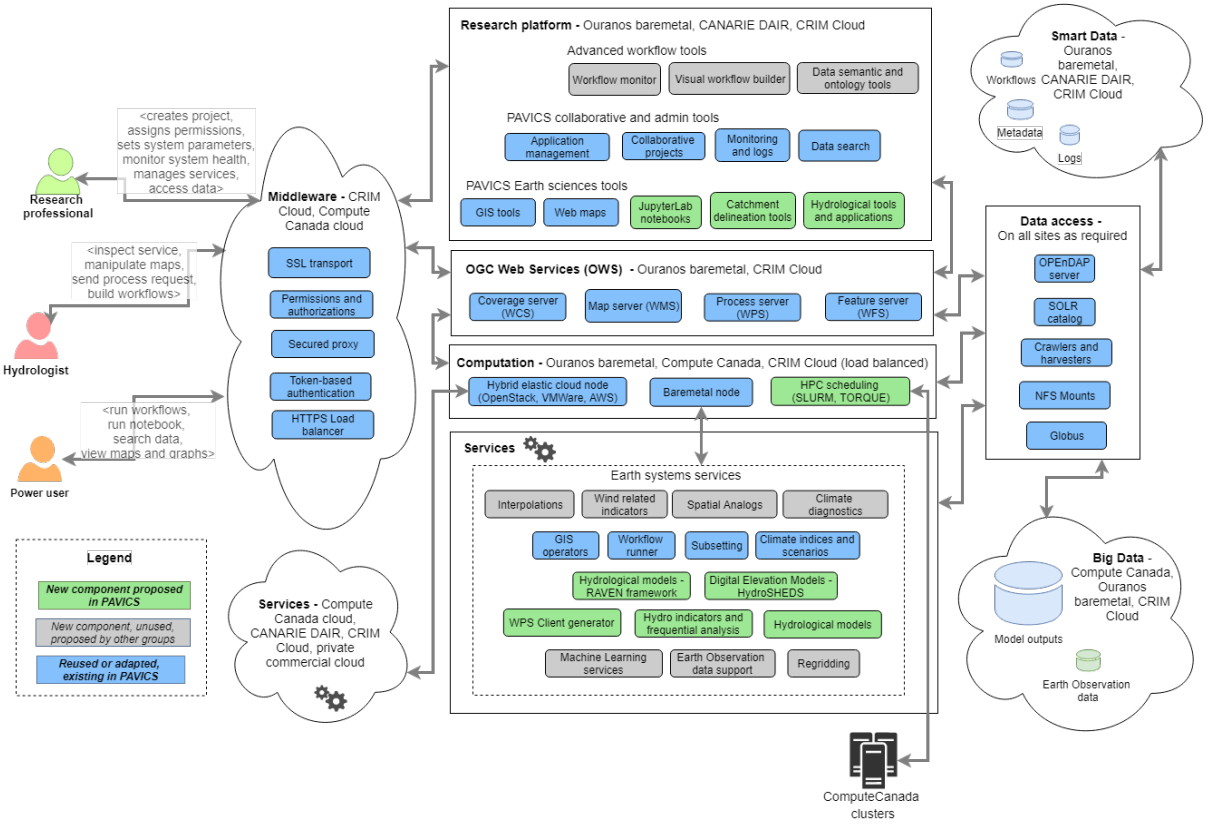
PROJECT EXAMPLES

The birdhouse *framework is modular organized* to enable a flexible architecture design depending on the projects needs. Due to the OCG Standard, software components non-birdhouse components can be combined for interoperability. Here are some examples of real projects to show the flexibility and potential of the birdhouse framework.

3.1 PAVICS

- PAVICS: Platform for climate analysis and visualization by Ouranos and CRIM, Canada.
- PAVICS-Hydro : Additional services for PAVICS allowing users to perform hydrological modeling and analysis.

3.1.1 Backend - PAVICS Node



PAVICS nodes are data, compute and index endpoints accessed through the PAVICS platform or external clients. The Node service is the backend that provides data storage, metadata harvesting, indexation and discovery of local and federated data, user authentication and authorization, server registration and management. The *node service* is therefore composed of several services that are briefly described below, accompanied by links to the full documentation of each individual building block.

The backend of PAVICS-SDI is built entirely with Free and Open Source Software. All of the backend projects (source code and documentation) are open to be inspected, built upon, or contributed to.

Data storage

Data is stored on two different servers: THREDDS for gridded netCDF data, and GeoServer for GIS features (region polygons, river networks).

THREDDS The *Thematic Real-time Environmental Distributed Data Services* (THREDDS) is a server system for providing scientific data and metadata access through various online protocols. The PAVICS platform relies on THREDDS to provide access to all netCDF data archives, as well as output files created by processes. The code is hosted on this GitHub repository. THREDDS support direct file access as well as the OPeNDAP protocol, which allows the netCDF library to access segments of the hosted data without downloading the entire file. Links to files archived on THREDDS are thus used as inputs to WPS processes. File content cannot however be directly displayed by the frontend and require an intermediary (see ncWMS).

GeoServer GeoServer is an OGC compliant server system built for viewing, editing, and presenting geospatial data. PAVICS uses GeoServer as its database for vector geospatial information, such as administrative regions, watersheds and river networks. The frontend sends requests for layers that can be overlayed on the map canvas. See the GeoServer documentation for more information on its capabilities.

Indexation

Although information about file content is stored in the netCDF metadata fields, accessing and reading those fields one by one takes a considerable amount of time. The strategies used here mimic those used by ESGF, and comprises running a crawler over all netCDF files hosted on THREDDS, extracting relevant metadata and storing them in a SOLR database. Search queries are thus directed at SOLR, which returns a list of links matching the search terms. The crawler is part of the PAVICS-DataCatalog library.

SOLR SOLR is a search platform part of the Apache Lucene project. It is used in this project for its faceted search capability. Search queries are relayed from the UI or WPS processes to the SOLR database, which returns a json file with the links to matching files.

PAVICS-DataCatalog PAVICS-DataCatalog is a database system for storing and serving information about available climate data.

Climate Analytic Processes with Birdhouse

The climate computing aspect of PAVICS is largely built upon the many components developed as part of the Birdhouse Project. The goal of Birdhouse is to develop a collection of easy-to-use Web Processing Service (WPS) servers providing climate analytic algorithms. Birdhouse servers are called 'birds', each one offering a set of individual processes:

Birdhouse/Finch Provides access to a large suite of climate indicators, largely inspired by **`ICCLIM`_**. Finch Official Documentation

Raven Provides hydrological modeling capability using the Raven framework, along with model calibration utilities, regionalization tools, hydrological indicators and frequency analysis.

Birdhouse/Malleefowl Provides processes to access ESGF data nodes and THREDDS catalogs, as well as a workflow engine to string different processes together. Malleefowl Official Documentation

Birdhouse/Flyingpigeon Provides a wide array of climate services including indices computation, spatial analogs, weather analogs, species distribution model, subsetting and averaging, climate fact sheets, etc. FlyingPigeon is the sand box for emerging services, which eventually will make their way to more stable and specialized birds. Flyingpigeon Official Documentation

Birdhouse/Hummingbird Provides access to climate Data Operators (CDO) functions and compliance-checker for netCDF files. Hummingbird Official Documentation

Virtually all individual processes ingest and return netCDF files (or OPeNDAP links), such that one process' output can be used as the input of another process. This lets scientist create complex workflows. By insisting that process inputs and outputs comply with the CF-Convention, we make sure that data is accompanied by clear and unambiguous metadata.

Authentication and authorization

Access to files and services is controlled by a security proxy called **`Twitcher`_**, also part of Birdhouse. Upon login, the proxy issues access tokens that allow users to access services behind the proxy. CRIM developed a Twitcher extension called Magpie that provides a higher level of granularity for service access.

Twitcher Proxy service issuing access tokens necessary to run WPS processes or any other OWS service.

Magpie Manages user/group/resource permissions for services behind Twitcher.

Gridded data visualization

The UI can display 2D netCDF fields by making a request to a ncWMS server. The UI will specify which time step of which file to map, and ncWMS will fetch the data from the THREDDS server, then convert the array into an image embedded into a WMS response. This conversion requires a mapping of numerical value to a color scale: a colormap and min/max values. The colormap is defined by the user through the UI, while default min/max values are stored in our SOLR database by the metadata crawler. Users may also specify min/max values directly within the UI.

ncWMS ncWMS is an implementation of the OGC's Web Mapping Service (WMS) specifically built for multidimensional gridded data such as the netCDF format. The PAVICS platform uses it to convert gridded netCDF data layers from a file or an OPeNDAP link to an image that can be accessed through WMS `GetMap` requests. See this reference paper for more information.

3.1.2 Backend - PAVICS Node

PAVICS nodes are data, compute and index endpoints accessed through the PAVICS platform or external clients. The Node service is the backend that provides data storage, metadata harvesting, indexation and discovery of local and federated data, user authentication and authorization, server registration and management. The *node service* is therefore composed of several services that are briefly described below, accompanied by links to the full documentation of each individual building block.

The backend of PAVICS-SDI is built entirely with Free and Open Source Software. All of the backend projects (source code and documentation) are open to be inspected, built upon, or contributed to.

Data storage

Data is stored on two different servers: THREDDS for gridded netCDF data, and GeoServer for GIS features (region polygons, river networks).

THREDDS The *Thematic Real-time Environmental Distributed Data Services* (THREDDS) is a server system for providing scientific data and metadata access through various online protocols. The PAVICS platform relies on THREDDS to provide access to all netCDF data archives, as well as output files created by processes. The code is hosted on this GitHub repository. THREDDS support direct file access as well as the OPeNDAP protocol, which allows the netCDF library to access segments of the hosted data without downloading the entire file. Links to files archived on THREDDS are thus used as inputs to WPS processes. File content cannot however be directly displayed by the frontend and require an intermediary (see ncWMS).

GeoServer GeoServer is an OGC compliant server system built for viewing, editing, and presenting geospatial data. PAVICS uses GeoServer as its database for vector geospatial information, such as administrative regions, watersheds and river networks. The frontend sends requests for layers that can be overlayed on the map canvas. See the GeoServer documentation for more information on its capabilities.

Indexation

Although information about file content is stored in the netCDF metadata fields, accessing and reading those fields one by one takes a considerable amount of time. The strategies used here mimic those used by ESGF, and comprises running a crawler over all netCDF files hosted on THREDDS, extracting relevant metadata and storing them in a SOLR database. Search queries are thus directed at SOLR, which returns a list of links matching the search terms. The crawler is part of the PAVICS-DataCatalog library.

SOLR SOLR is a search platform part of the Apache Lucene project. It is used in this project for its faceted search capability. Search queries are relayed from the UI or WPS processes to the SOLR database, which returns a json file with the links to matching files.

PAVICS-DataCatalog PAVICS-DataCatalog is a database system for storing and serving information about available climate data.

Climate Analytic Processes with Birdhouse

The climate computing aspect of PAVICS is largely built upon the many components developed as part of the Birdhouse Project. The goal of Birdhouse is to develop a collection of easy-to-use Web Processing Service (WPS) servers providing climate analytic algorithms. Birdhouse servers are called 'birds', each one offering a set of individual processes:

Birdhouse/Finch Provides access to a large suite of climate indicators, largely inspired by **`ICCLIM`_**. Finch Official Documentation

Raven Provides hydrological modeling capability using the Raven framework, along with model calibration utilities, regionalization tools, hydrological indicators and frequency analysis.

Birdhouse/Malleefowl Provides processes to access ESGF data nodes and THREDDS catalogs, as well as a workflow engine to string different processes together. Malleefowl Official Documentation

Birdhouse/Flyingpigeon Provides a wide array of climate services including indices computation, spatial analogs, weather analogs, species distribution model, subsetting and averaging, climate fact sheets, etc. FlyingPigeon is the sand box for emerging services, which eventually will make their way to more stable and specialized birds. Flyingpigeon Official Documentation

Birdhouse/Hummingbird Provides access to climate Data Operators (CDO) functions and compliance-checker for netCDF files. Hummingbird Official Documentation

Virtually all individual processes ingest and return netCDF files (or OPeNDAP links), such that one process' output can be used as the input of another process. This lets scientist create complex workflows. By insisting that process inputs and outputs comply with the CF-Convention, we make sure that data is accompanied by clear and unambiguous metadata.

Authentication and authorization

Access to files and services is controlled by a security proxy called **`Twitcher`_**, also part of Birdhouse. Upon login, the proxy issues access tokens that allow users to access services behind the proxy. CRIM developed a Twitcher extension called Magpie that provides a higher level of granularity for service access.

Twitcher Proxy service issuing access tokens necessary to run WPS processes or any other OWS service.

Magpie Manages user/group/resource permissions for services behind Twitcher.

Gridded data visualization

The UI can display 2D netCDF fields by making a request to a ncWMS server. The UI will specify which time step of which file to map, and ncWMS will fetch the data from the THREDDS server, then convert the array into an image embedded into a WMS response. This conversion requires a mapping of numerical value to a color scale: a colormap and min/max values. The colormap is defined by the user through the UI, while default min/max values are stored in our SOLR database by the metadata crawler. Users may also specify min/max values directly within the UI.

ncWMS ncWMS is an implementation of the OGC's Web Mapping Service (WMS) specifically built for multidimensional gridded data such as the netCDF format. The PAVICS platform uses it to convert gridded netCDF data layers from a file or an OPeNDAP link to an image that can be accessed through WMS `GetMap` requests. See this reference paper for more information.

3.2 COPERNICUS

- CP4CDS: Climate Projects for the Climate Data Store (part of the European Union's Copernicus Climate Change Service).

3.3 OGC-Testbeds

Todo: Add references to OGC testbed.

- OGC Testbed 13: Enhancement of scheduling services
- OGC Testbed 14: Enhancement of security

3.4 A2C2

- A2C2: Atmospheric flow Analogues for Climate Change

CHAPTER

FOUR

GUIDELINES

To guide you through the learning curve of installation modules of birdhouse and set up an running birdhouse ecosystem, administer the server-side birdhouse components or even improve and develop your own specific functions, here are some general guidelines:

4.1 Installation Guidelines

Warning: This section is outdated . . .

Birdhouse consists of several components like Malleefowl and Emu. Each of them can be installed individually. The installation is done using the Python-based build system *Buildout*. Most of the dependencies are maintained in the *Anaconda Python distribution*. For convenience, each birdhouse component has a Makefile to ease the installation so you don't need to know how to call the Buildout build tool.

4.1.1 Requirements

Birdhouse uses *Anaconda Python distribution* for most of the dependencies. If Anaconda is not already installed, it will be installed during the installation process. Anaconda has packages for Linux, MacOSX and Windows. But not all packages used by birdhouse are already available in the default package channel of Anaconda. The missing packages are supplied by birdhouse on *Binstar*. But we currently maintain only packages for Linux 64-bit and partly for MacOSX.

So the short answer to the requirements is: **you need a Linux 64-bit installation**.

Birdhouse is currently used on Ubuntu 14.04 and CentOS 6.x. It should also work on Debian, LinuxMint and Fedora.

Birdhouse also installs a few system packages using *apt-get* on Debian based distributions and *yum* on RedHat/CentOS based distributions. For this you need a user account with *sudo* permissions. Installing system packages can be done in a separate step. So your installation user does not need any special permissions. All installed files will go into a birdhouse Anaconda environment in the home folder of the installation user.

4.1.2 Installing from source

The installation of birdhouse components from source is done with some few commands. Here is an example for the Emu WPS service:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean install
$ make start
$ firefox http://localhost:8094/wps
```

All the birdhouse components follow the same installation pattern. If you want to see all the options of the *Makefile* then type:

```
$ make help
```

You will find more information about these options in the Makefile documentation.

Read the documention of each birdhouse component for the details of the installation and how to configure the components. The birdhouse bootstrap documentation gives some examples of the different ways of making the installation.

On the WPS client side we have:

- Phoenix: a Pyramid web application.
- Birdy: a simple WPS command line tool.

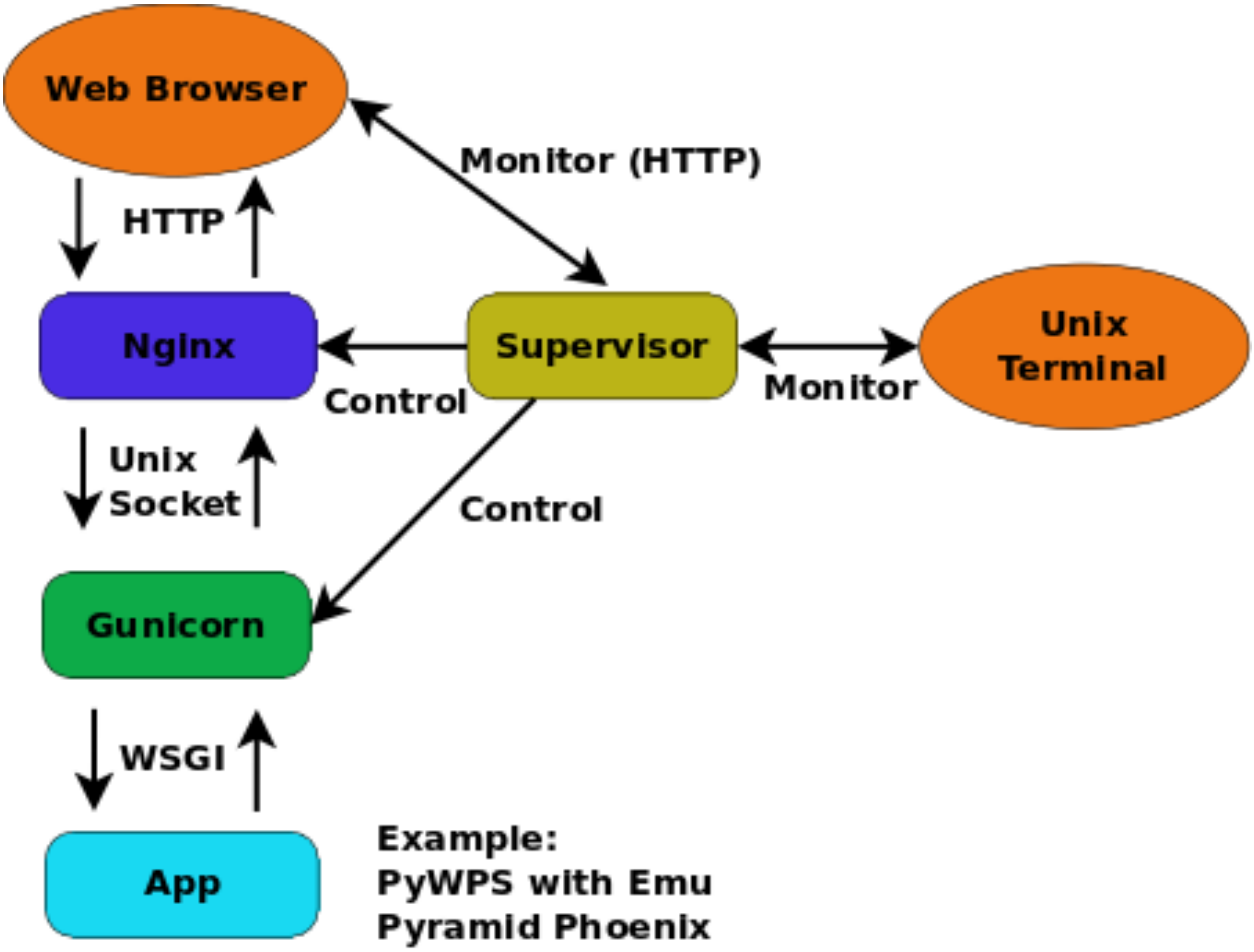
On the WPS server side we have:

- Malleefowl: provides base WPS services to access data.
- Flyingpigeon: provides WPS services for the climate impact community.
- Hummingbird: provides WPS services for CDO and climate metadata checks.
- Emu: just some WPS processes for testing.

4.1.3 Nginx, gunicorn and supervisor

Birdhouse sets up a *PyWPS* server (and also the Phoenix web application) using *Buildout*. We use the *Gunicorn* HTTP application server (similar to Tomcat for Java servlet applications) to run these web applications with the *WSGI* interface. In front of the Gunicorn application server, we use the *Nginx* HTTP server (similar to the Apache web server). All these web services are started/stopped and monitored by a *Supervisor* service.

See the following image for how this looks like:

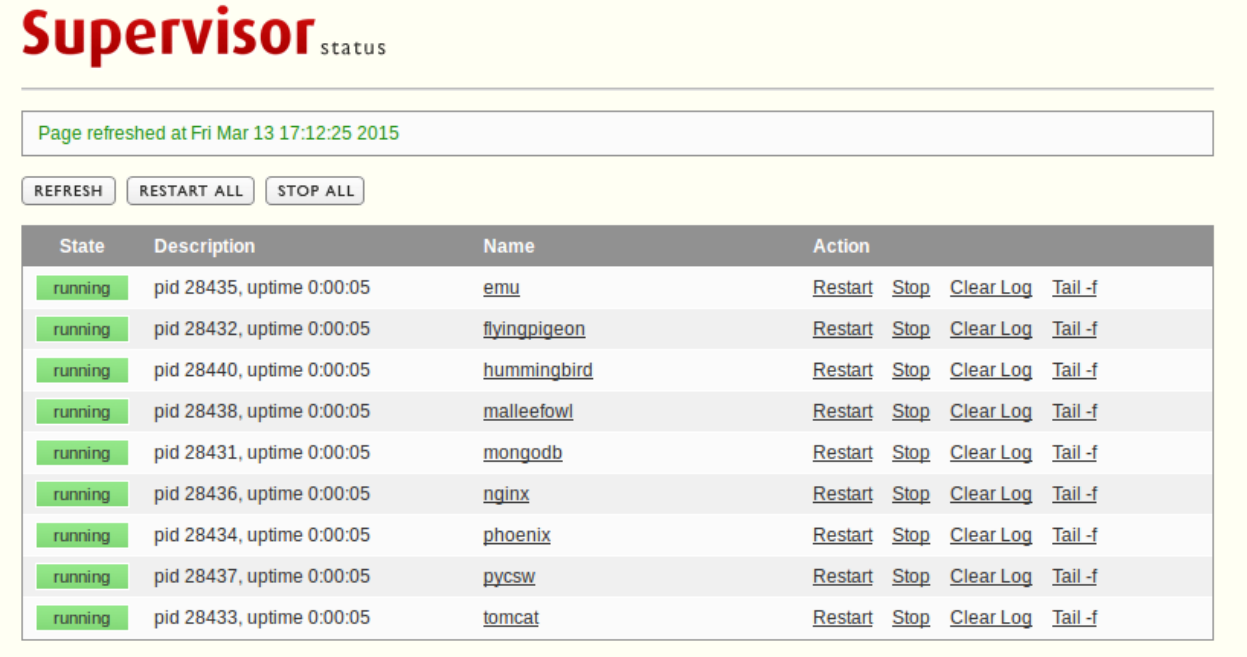


When installing a birdhouse WPS service, you don't need to care about this setup. This is all done by Buildout and using some extensions provided by birdhouse.

The Makefile of a birdhouse application has convenience targets to start/stop a WPS service controlled by the Supervisor and to check the status:

```
$ make start    # start wps service
$ make stop     # stop wps service
$ make status   # show status of wps service
Supervisor status ...
/home/pingu/.conda/envs/birdhouse/bin/supervisorctl status
emu                              RUNNING   pid 25698, uptime 0:00:02
malleefowl                       RUNNING   pid 25702, uptime 0:00:02
mongodb                          RUNNING   pid 25691, uptime 0:00:02
nginx                            RUNNING   pid 25699, uptime 0:00:02
phoenix                          RUNNING   pid 25694, uptime 0:00:02
pycsw                            RUNNING   pid 25700, uptime 0:00:02
tomcat                           RUNNING   pid 25693, uptime 0:00:02
```

You can also use the Supervisor monitor web service which by default is available on port http://localhost:9001/. The Supervisor monitor app looks like in the following screenshot.



4.1.4 Using birdhouse with Docker

An alternative way to install and deploy birdhouse Web Processing Services is by using *Docker*. The birdhouse WPS servers are available as a Docker image on Docker Hub. See an example on how to use them with the Emu WPS Docker image.

4.2 Administrator Guidelines

Warning: This section needs is outdated and needs to be rewritten!

4.2.1 Set up a birdhouse ecosystem server

If you are already familiar with installing single standalone WPS (follow the *Installation Guidelines* guides in the documentations of e.g. emu), then you are ready to set up a birdhouse containing flyingpigeon (providing scientific analyses methods), malleefowl (to search and fetch data) and the pheonix (a graphic interface for a web browser including a WMS).

General Remarks

Check the *Requirements* of your system!
The installation is done as **normal user**, root rights are causing conflicts.

Prepare Installation

It is recommended to collect the repositories in a separate folder (e.g. birdhouse, but can have a name of your choice):

```
$ mkdir birdhouse
$ cd birdhouse
```

Get the source code from GitHub

```
$ git clone https://github.com/bird-house/flyingpigeon.git
$ git clone https://github.com/bird-house/pyramid-phoenix.git
$ git clone https://github.com/bird-house/malleefowl.git
```

Run Installation

You can run the installation with default settings. It will create a conda environment and deploy all required software dependencies there.

Note: Read the *changing the default configuration* if you want to customize the configuration.

In **all** of the tree folders (malleefowl, flyingpigeon and pyramid-phoenix) run:

```
$ make install
```

This installation will take some minutes to fetch all dependencies and install them into separate conda environments.

Start the Services

in **all** of the birds run:

```
$ make start
```

Launching the Phoenix Web App

If the services are running, you can launch the GUI in a common web browser. By default, phoenix is set to port 8081:

```
firefox http://localhost:8081
```

or:

```
firefox https://localhost:8443/
```

Now you can log in (upper right corner) with your Phoenix password created previously. Phoenix is just a graphical interface with no more function than looking nice ;-).

Register a service in Phoenix Web App

Note: Please read the Phoenix documentation

Your first administration step is to register *flyingpigeon* as a service. For that, log in with your phoenix password. In the upper right corner is a tool symbol to open the *settings*. Click on *Services* and the *Register a Service*.

Flyingpigeon is per default on port 8093.

The appropriate url is:

```
http://localhost:8093/wps
```

Provide service title and name as you like: * Service Title: Flyingpigeon * Service Name: flyingpigeon

check *Service Type*: *Web Processing Service* (default) and register.

Optionally, you can check *Public access?*, to allow unregistered users to launch jobs. (**NOT recommended**)

Launching a Job

Now your birdhouse ecosysem is set up. The also installed malleefowl is already running in the background and will do a lot of work silently. There is **no need to register malleefowl** manually!

Launching a job can be performed as a process (Process menu) or with the wizard. To get familliar with the processes provided by each of the birds, read the approriate documentation for each of the services listed in the overview:

Changing the default configuration

You can customize the configuration of the service. Please read the documentation, for example:

- Phoenix documentation
- Flyingpigeon documentation

Furthermore, you might change the hostname (to make your service accessible from outside), ESGF-node connection, the port or the log-level for more/less information in the administrator logfiles. Here is an example *pyramid-phoenix/custom.cfg*:

```
[settings]
hostname = localhost
http-port = 8081
https-port = 8443
log-level = DEBUG
# run 'make passwd' and to generate password hash
phoenix-password = sha256:513....
# generate secret
# python -c "import os; print(''.join('%02x' % ord(x) for x in os.urandom(16)))"
phoenix-secret = d5e8417....30
esgf-search-url = https://esgf-data.dkrz.de/esg-search
wps-url = http://localhost:8091/wps
```

Update Phoenix Password

To be able to log into the Phoenix GUI once the services are running, it is necessary to generate a password: go into the pyramid-phoenix folder and run:

```
$ make passwd
```

This will automatically write a password hash into pyramid-phoenix/custom.cfg

4.2.2 Backups

See the mongodb documentation on how to backup the database. With the following command you can make a dump of the `users` collection of the Phoenix database:

```
$ mongodump --port 27027 --db phoenix_db --collection users
```

4.2.3 Asking for Support

In case of questions or trouble shooting, feel welcome to join the birdhouse chat and get into contact with the developers directly.

4.3 Developer Guidelines

4.3.1 Code of Conduct

Note: Before we start please be aware that contributors to this project are expected to act respectfully toward others in accordance with the OSGeo Code of Conduct.

4.3.2 Contribution Workflow

The Birdhouse project openly welcomes contributions (bug reports, bug fixes, code enhancements/features, etc.). This document will outline some guidelines on contributing to birdhouse. As well, the birdhouse *Communication* is a great place to get an idea of how to connect and participate in birdhouse community and development where everybody is welcome to rise questions and discussions.

Here are some basic guides to smoothly contribute to birdhouse:

Source code

The source code of all birdhouse components is available on GitHub. Respecting the git mechanisms you can fork, clone and pull source-code into your repositories for modification and enhancement. Once your improvement is ready, make a pull request to integrate your work into the origin birdhouse repositories.

Note: Please keep your forks close to the origin repositories and don't forget the pull requests.

Issue tracker

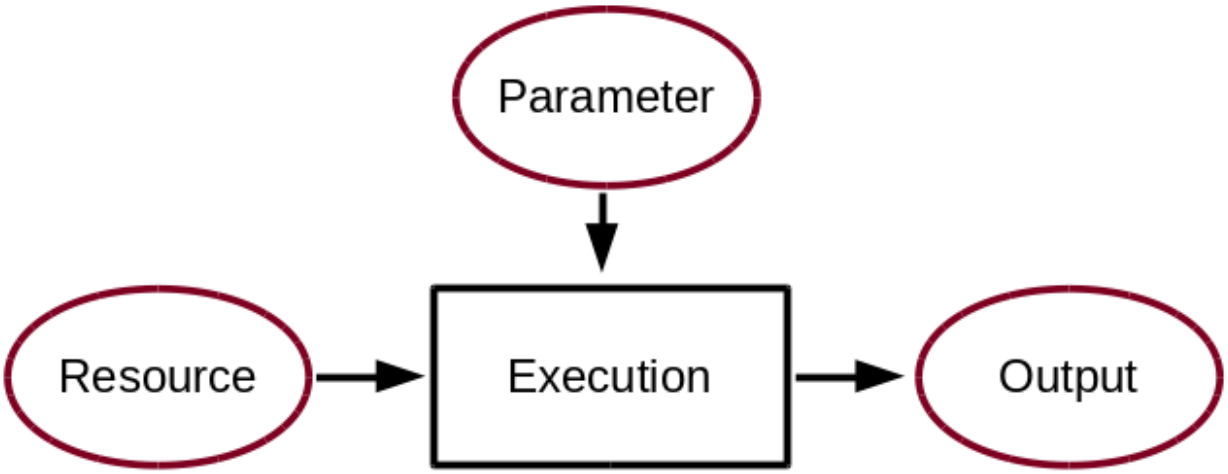
To keep track on the contribution and development, please use the issue tracker on GitHub for the corresponding birdhouse component.

Please find the coding guide in the Wiki.

4.3.3 Writing a WPS process

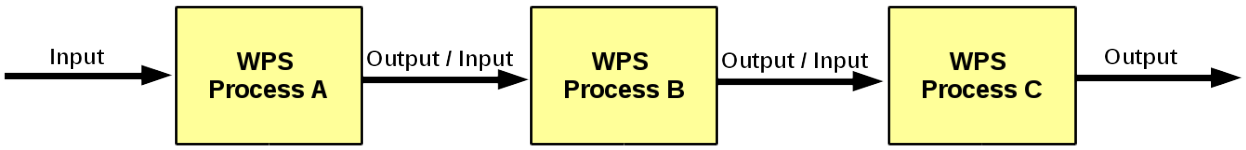
In birdhouse, we are using the PyWPS implementation of a *Web Processing Service*. Please read the PyWPS documentation on how to implement a WPS process.

Note: To get started quickly, you can try the Emu WPS with some example processes for PyWPS.



Another point to think about when designing a process is the possibility of chaining processes together. The result of a process can be a final result or be used as an input for another process. Chaining processes is a common practice but depends on the user you are designing the service for. Technically, for the development of WPS process chaining, here are a few summary points:

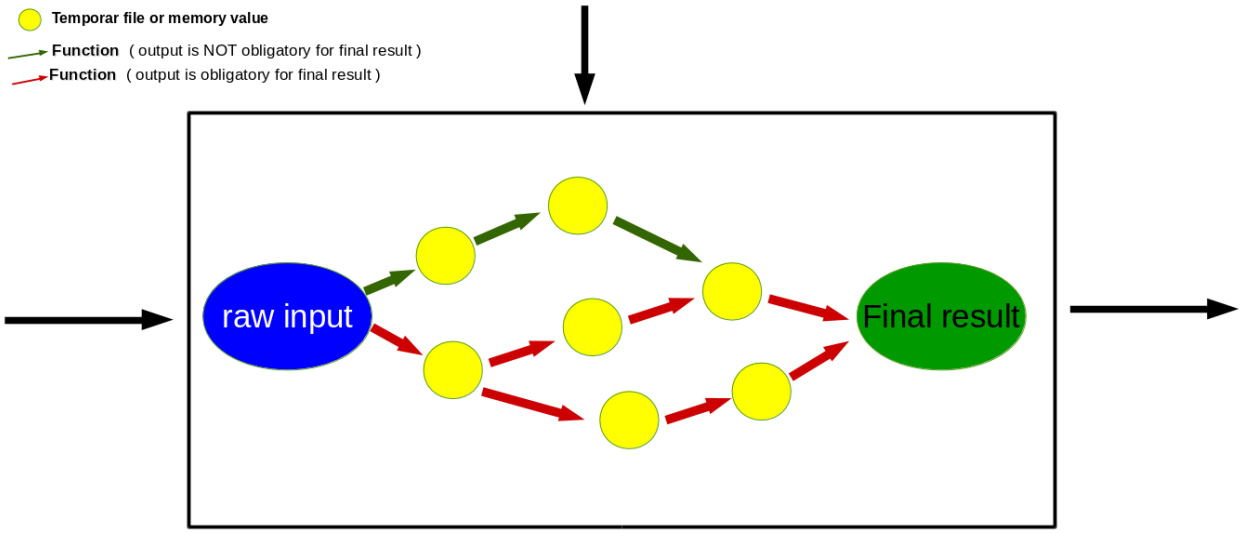
- the functional code should be modular and provide an interface/method for each single task
- provide a wps process for each task
- wps processes can be chained, manually or within the code, to run a complete workflow
- wps chaining can be done manually, with workflow tools, direct wps chaining or with code scripts
- a complete workflow chain could also be started by a wps process.



4.3.4 Writing functions

A Process is calling several functions during the performance. Since WPS is a autonom running process several eventualities needs to be taken into account. If irregularities are occurring, it is a question of the process design if the performance should stop and return an error or continue with may be an modified result.

In practice, the functions should be encapsulated in **try** and **except** calls and appropriate information given to the logfile or shown as a status message. The logger has several options to to influence the running code and the information writing to the logfile:




```
# the following two line needs to be in the beginning of the *.py file.
# The ._handler will find the appropriate logfile and include timestemps
# and module information into the log.

import logging
LOGGER = logging.getLogger("PYWPS")

# set a status message
per = 5  # 5 will be 5% in the status line
response.update_status('execution started at : {}'.fromat(dt.now()), per)

try:
    response.update_status('the process is doing something: {}'.fromat(dt.now()),10)
    result = 42
    LOGGER.info('found the answer of life')
except Exception as ex:
    msg = 'This failed but is obligatory for the output. The process stops now, because: {} '.format(ex)
    LOGGER.error(msg)

try:
    response.update_status('the process is doing something else : {}'.fromat(dt.now()), 20)
    interesting = True
    LOGGER.info(' Thanks for reading the guidelines ')
    LOGGER.debug(' I need to know some details of the process: {} '.format(interesting)
except Exception as ex:
    msg = 'This failed but is not obligatory for the output. The process will continue. Reason for the failure: {} '.format(ex)
```

(continues on next page)

(continued from previous page)

```
27      LOGGER.exception(msg)
```

4.3.5 Writing tests

Todo: Guideline to write tests. Look at the Emu to see examples.

4.3.6 Writing documentation

Last but not least, a very very important point is to write a good documentation about your work! Each WPS (bird) has a docs folder for this where the documentation is written in reStructuredText and generated with Sphinx.

- http://sphinx-doc.org/tutorial.html
- http://quick-sphinx-tutorial.readthedocs.io/en/latest/

The documentation is automatically published to ReadTheDocs with GitHub webhooks. It is important to keep the *Code Style* and write explanations to your functions. There is an auto-api for documentation of functions.

Todo: explanation of enabling spinx automatic api documentation.

The main documentation (which you are reading now) is the starting point to get an overview of birdhouse. Each birdhouse component comes with its own Sphinx documentation and is referenced by the main birdhouse document. Projects using birdhouse components like **PAVICS_** or COPERNICUS Data Store generally have their own documentation as well. To include documentation from external repository here, two custom made sphinx directives can be used. The *gittoctree* directive behaves like a normal table of content directive (*toctree*), but takes as an argument the URL to the git repo and refers to files inside this directory through their full path. The *gitinclude* directive acts like an normal *include* directive, but takes as a first argument the URL to the git repo this file belongs to. For example:

```
Here is the text of the birdhouse main documentation. At the place where you want to␣
↪integrate
a part of a remote sphinx documentation stored in a `git` repository you can fetch␣
↪the docs
parts and integrated it with a table of content referring to external files:

.. gittoctree:: https://github.com/Ouranosinc/pavics-sdi.git

   docs/source/arch/backend.rst

or include an individual file:

.. gitinclude:: https://github.com/Ouranosinc/pavics-sdi.git docs/source/arch/backend.
↪rst

The directive will clone and checkout the repository, then include these external␣
↪files as if
they were part of the native documentation.
```

4.3.7 Code Style

A good start to contribute is an enhancement of existing code with better or new functions. To respect a common coding style, Birdhouse uses PEP8 checks to ensure a consistent coding style. Currently the following PEP8 rules are enabled in `setup.cfg`:

```
[flake8]
ignore=F401,E402
max-line-length=120
exclude=tests
```

See the flake8 documentation on how to configure further options.

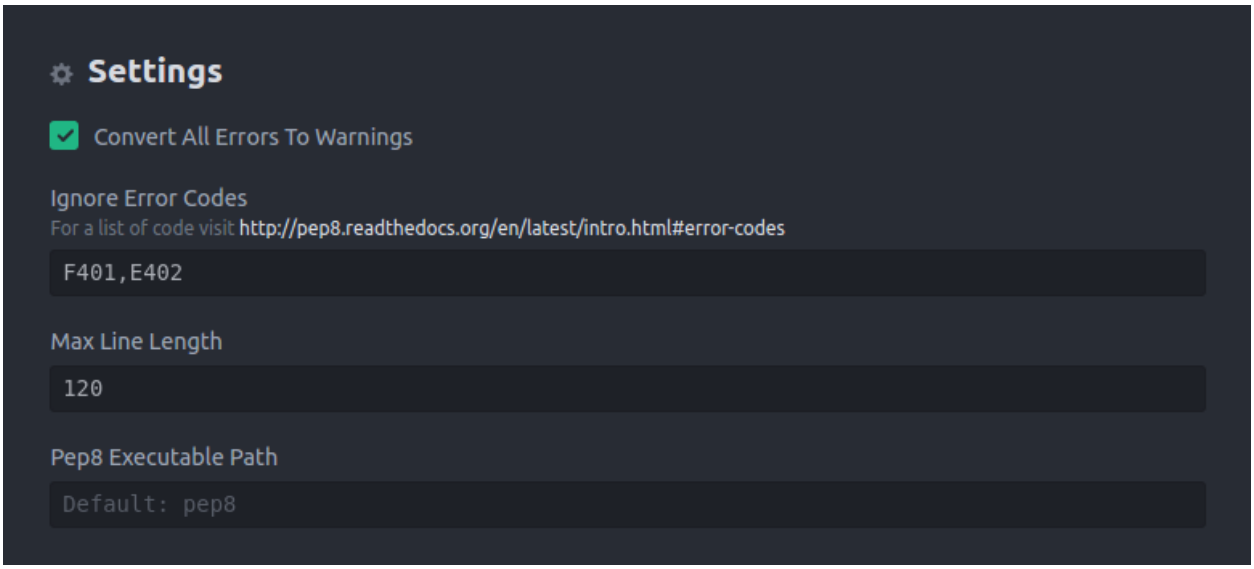
To check the coding style run `flake8`:

```
$ flake8 emu    # emu is the folder with python code
# or
$ make pep8     # make calls flake8
```

To make it easier to write code according to the PEP8 rules enable PEP8 checking in your editor. In the following we give examples how to enable code checking for different editors.

Atom

- Homepage: https://atom.io/
- PEP8 Atom Plugin: https://github.com/AtomLinter/linter-pep8



Sublime

- Install package control if you don't already have it: https://packagecontrol.io/installation
- Follow the instructions here to install Python PEP8 Autoformat: https://packagecontrol.io/packages/Python%20PEP8%20Autoformat
- Edit the settings to conform to the values used in birdhouse, if necessary
- To show the ruler and make wordwrap default, open Preferences → Settings—User and use the following rules

```
{
 // set vertical rulers in specified columns.
 "rulers": [79],

 // turn on word wrap for source and text
 // default value is "auto", which means off for source and on for text
 "word_wrap": true,

 // set word wrapping at this column
 // default value is 0, meaning wrapping occurs at window width
 "wrap_width": 79
 }
```

Todo: Add PEP8 instructions for more editors: PyCharm, Kate, Emacs, Vim, Spyder.

4.3.8 Environment with conda

Todo: How to create a conda package

4.3.9 Make your own Bird

If you are familiar with all the upper chapters you are ready to create your own WPS. The WPS in birdhouse are named after birds, so this section is giving you a guidline of how to make your own bird. Birds are sorted thematically, so before setting up a new one, make sure it is not already covered and just missing some processes and be clear in the new thematic you would like to provide.

We have now a Cookiecutter template to create a new bird (PyWPS application). It is the recommended and fastest way to create your own bird:

https://github.com/bird-house/cookiecutter-birdhouse

Note: The cookiecutter is brand-new. Please give feedback and help to improve it.

4.3.10 Release Notes and Versions:

The development of birdhouse is following a release cycle of around three month. Updates of modules are coordinated by the developers over the communication channels (gitter chat or Video Conference). New releases are documented in the release notes and communicated over the mailing list. A release of a birdhouse module is taged with a version number and appropriate git repository version branch.

For an orientation of when to release a new version:

- Full version (v1.0) with scientific publication in a reviewed journal
- subversion (v1.1) by major changes
- subsub versions (v1.1.1) by minor changes

out of the release cycles bug fix patches can be released every time (communication is not mandatory)

- patch v1.1.1_patch1 bugfix

4.4 User Guidelines

Warning: Work in progress. Examples will come soon.

You can connect to a WPS service in the following ways:

- using a command-line tool in your terminal.
- using a web based application from your browser.
- using a Python library from a jupyter notebook or your Python scripts.

4.4.1 Command-line

Todo: birdy example

4.4.2 Phoenix Web App

Todo: Screen-shot of Phoenix

4.4.3 Python Library

Python syntax:

```
"""Python WPS execute"""

from owslib.wps import WebProcessingService, monitorExecution
from os import system
```

```
wps = WebProcessingService(url="http://localhost:8093/wps", verbose=False)
print("Service '{}' is running".format(wps.identification.title))
```

```
Service 'Flyingpigeon' is running
```

```
for process in wps.processes:
    print( '{} : \t {}'.format(process.identifier, process.abstract))
```

```
subset :     Return the data for which grid cells intersect the selected polygon for␣
↪each input dataset as well asthe time range selected.
subset_bbox :        Return the data for which grid cells intersect the bounding box␣
↪for each input dataset as well asthe time range selected.
subset_continents :          Return the data whose grid cells intersect the selected␣
↪continents for each input dataset.
subset_countries :   Return the data whose grid cells intersect the selected␣
↪countries for each input dataset.
pointinspection :    Extract the timeseries at the given coordinates.
subset_WFS :         Return the data for which grid cells intersect the selected␣
↪polygon for each input dataset.
plot_timeseries :    Outputs some timeseries of the file field means. Spaghetti and␣
↪uncertainty plot
```

```
# define some data urls

url1 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2000.nc'
url2 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2001.nc'
url3 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2002.nc'
url4 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2003.nc'
```

```
execute = wps.execute(
    identifier="plot_timeseries", #indices_clipping",
    inputs=[
        ("resource",url1),
        ("resource",url2),
```

(continues on next page)

(continued from previous page)

```
       ("resource",url3),
       ("resource",url4),
       # ("variable" , "slp"),
       ])

monitorExecution(execute, sleepSecs=5)
print(execute.getStatus())

for o in execute.processOutputs:
    print(o.reference)
```

```
 owslib.wps.WPSException : {'code': 'NoApplicableCode', 'locator': 'None', 'text':
↪'Process failed, please check server error log'}
ProcessFailed
```

```
from eggshell.nc.nc_utils import get_coordinates
```

CHAPTER

FIVE

IDEAS

In this section we are collection ideas how we could improve our coding and design in the Birdhouse/WPS context.

5.1 PyWPS Profiles

Warning: Work in progress.

5.1.1 Motivation

It happens quite often that we have a set of processes with common input (and output) parameters. In WPS the process signature (inputs+outputs) is called a WPS profile. In the following we show examples how to avoid *copy+paste* of these process parameters.

5.1.2 Python Mixins

One could use Python *mixin classes* to define a commonly used profile which can be adapted by each individual process.

See how a mixin class looks like:

https://www.ianlewis.org/en/mixins-and-python

See notebook examples how it could be used with PyWPS:

https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_mixin.ipynb

5.1.3 Python Decorators

We can also use function decorator to define a WPS profile for PyWPS.

See how a function decorator looks like:

https://krzysztofzuraw.com/blog/2016/python-class-decorators.html

Here are some notebook examples how it could be used with PyWPS:

- notebooks: https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_decorator.ipynb
- Emu subset with ESGF-API: https://github.com/bird-house/emu/blob/esgfwps/emu/processes/wps_esgf_subset.py

5.1.4 Simple Alternative: Shared Profile Module/Class

Relatively few developers will be familiar with the concepts of *mixins* and *decorators*. In other words, it might look a bit too much like magic. We could also simply create a module with all the common inputs and outputs used throughout the different WPS processes (*wpsio.py*). For a given Process definition, one then just import *wpsio* and refer to the objects in the inputs and outputs fields of the *Process.init* method.

See for example:

https://github.com/Ouranosinc/raven/blob/master/raven/processes/wps_regionalisation.py

Here is a notebook showing this approach which includes also an optional decorator:

https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_simple_profile_and_decorator.ipynb

CHAPTER

SIX

COMMUNICATION

There are numerous ways to interact with the Birdhouse community, for example join the chat or follow our blog. Also we are present on several conferences where you can enjoy one of our good presentations.

6.1 Chat-room

The most easiest way to drop a line to the developers is our Gitter chat room. If you want to have a quick technical question to one of the developers, or just wants to follow the discussions, feel welcome to join.

6.2 Meetings

More complex and real discussions are done regularly in video conferences. Check out the information for upcoming birdhouse meetings. Here you also find the minutes of previews video conferences and feel welcome to join an upcoming one.

6.3 Blog-post

In the blog you can find interesting articles and information related to birdhouse in general. We also inform regularly abut the main steps forward in the software development that you can keep track on whats going on in the birdhouse. If you want to receive a notification of new articles follow birdhouse news on our blog:

- The IT Landscape for Climate Services
- Cyberinfrastructures for Sustainable Development

6.4 Newsletter

To be informed about the main progress in the birdhouse development as well as related information you can subscribe to our newsletter.

6.5 Wiki

The birdhouse wiki provides an area for supporting information that frequently changes and / or is outside the scope of the formal documentation.

CHAPTER

SEVEN

PUBLICATIONS

7.1 Talks and articles

Articles, book sections and conference proceedings and presentations related to the birdhouse projects:

2019:

- WPS Deployment at CORDEX Copernicus Workshop, Copenhagen
- UN GIS Initiative Workshop at FOSS4G Bucharest

2018:

- Birdhouse in ISPRS photogrammetry and remote-sensing [ELH+18]
- FOSS4G 2018 in Dar-Es-Salaam
- Open Climate GIS and Birdhouse at Pangeo Developer Workshop, 2018
- IGARSS 2018
- D-GEO Days, 2018
- GIZ Fachtagung, 2018
- Copernicus/Birdhouse at EGU 2018, Vienna
- Flyingpigeon in Computes and Geosciences, January 2018 [HEAC+18]

2017:

- Birdhouse in LSDMA book, 2017 [JMS17]
- UNCCC Subgroup 2017 in Kigali

2016:

- AGU 2016 in San Francisco
- ESGF F2F 2016 in Washington
- FOSS4G 2016 in Bonn
- EGI Workshop 2016 in Amsterdam

- EGU 2016 in Vienna
- ICRC-CORDEX 2016
- Model Animation LSCE
- Talk on USGS WebEx 2016/02/18

2015:

- Paris Coding Spring 2015 at IPSL

2014:

- EGI Community Forum 2014 at Helsinki
- Prag
- Optimization of data life cycles [JGG+14]

2013:

- Gerics Hamburg User-Developer Workshop

7.2 References

CHAPTER

EIGHT

RELEASE NOTES

8.1 Bucharest (October 2019, v0.8.0)

PyWPS was present at FOSS4G 2019 in Bucharest.

Highlighted Changes:

- Skipped buildout in Twitcher.
- Skipped conda handling in Makefile.
- Working on OAuth support in Twitcher and birdy.
- Released OWSLib extension for ESGF compute API.

Released Birds:

- Twitcher WPS Proxy: 0.5.2
- Ansible Playbook for PyWPS 0.2.2
- Cookiecutter Template for PyWPS 0.4.1
- Birdy WPS Client: 0.6.5
- Emu WPS: 0.11.0

- FlyingPigeon WPS: 1.5
- Finch WPS: 0.2.5
- Hummingbird WPS: 0.8.0
- Malleefowl WPS: 0.9.0
- OWSLib extension for ESGF: 0.2.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.10.0

New Birds in the making:

- Kingfisher: https://github.com/bird-house/kingfisher
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell
- Pelican: https://github.com/bird-house/pelican

8.2 San Francisco (May 2019, v0.7.0)

Highlighted Changes:

- All released birds support only Python >3.6.
- Support for the ESGF WPS profile with a Pelican WPS demo and an OWSLib extension.
- Support for MetaLink in Birdy and PyWPS to return multiple files as WPS output.
- Release of Finch, a WPS for climate indicators.

Released Birds:

- Ansible Playbook for PyWPS 0.2.1
- Cookiecutter Template for PyWPS 0.4.0
- Birdy WPS Client: 0.6.0
- Emu WPS: 0.10.0
- FlyingPigeon WPS: 1.4.1
- Finch WPS: 0.2.0
- Hummingbird WPS: 0.7.0
- Malleefowl WPS: 0.8.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.9.0
- Twitcher WPS Proxy: 0.4.0

New Birds in the making:

- Kingfisher: https://github.com/bird-house/kingfisher
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell

- Pelican: https://github.com/bird-house/pelican
- OWSLib extension for ESGF: https://github.com/bird-house/OWSLib-esgfwps

8.3 Washington (December 2018, v0.6.1)

Birdhouse was present at the AGU 2018 and ESGF Face to Face 2018 both in Washington D.C.

Highlighted Changes:

- Improved *Birdy WPSClient* as a pythonic library for WPS client with support for Jupyter Notebooks.
- Converted *Malleefowl* and *FlyingPigeon* to new deployment layout without buildout.
- New birds: *Finch* WPS for Climate Indicators and *Kingfisher* for Earth Observation Data Analysis.
- *FlyingPigeon* has been reborn as the *Curious Climate Explorer*. Most of its original functionallity has moved to other birds: *BlackSwan*, *Kingfisher* and *Finch*.

Released Birds:

- Ansible Playbook for PyWPS 0.2.0
- Cookiecutter Template for PyWPS 0.3.1
- Birdy WPS Client: 0.5.0
- Emu WPS: 0.9.1
- Hummingbird WPS: 0.6.1
- Malleefowl WPS: 0.7.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.8.3
- Twitcher WPS Proxy: 0.3.8

New Birds in the making:

- FlyingPigeon (reborn): https://github.com/bird-house/flyingpigeon
- Kingfisher: https://github.com/bird-house/kingfisher
- Finch: https://github.com/bird-house/finch
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell

8.4 Dar es Salaam (September 2018, v0.6.0)

Birdhouse was present at the FOSS4G 2018 in Dar es Salaam.

Highlighted Changes:

- Ansible playbook to install PyWPS applications.
- Skipped Buildout deployment . . . not all birds are converted yet.
- Updated Cookiecutter template for new deployment.
- Using PyWPS OpenDAP support.

- Initial version of Birdy native client.

Released Birds:

- Ansible Playbook for PyWPS 0.1.0
- Cookiecutter Template for PyWPS 0.3.0
- Birdy WPS Client: 0.4.0
- Emu WPS: 0.9.0
- Hummingbird WPS: 0.6.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.8.2
- Twitcher WPS Proxy: 0.3.8
- Flyingpigeon WPS: 1.2.1
- Malleefowl WPS: 0.6.8

New Birds in the making:

- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell

8.5 Montréal (March 2018, v0.5.0)

We had a workshop in Montréal with CRIM and Ouranos.

Highlighted Changes:

- Birdhouse has a Logo :)
- A Cookiecutter template for Birdhouse WPS birds is available.
- A new WPS Bird Black Swan for extreme weather event assessments is started by LSCE, Paris. This bird is spawned off Flyingpigeon.
- A new Python library, Eggshell, is started to provide common base functionallity to WPS birds like Flyingpigeon and Black Swan.
- The Twitcher security proxy supports now X509 certificates for authentication to WPS services.

Released Birds:

- Phoenix 0.8.1
- Birdy 0.2.1
- Twitcher 0.3.7
- Flyingpigeon 1.2.0
- Hummingbird 0.5.7
- Malleefowl 0.6.7
- Emu 0.6.3

New Birds in the making:

- Black Swan: https://github.com/bird-house/blackswan

- Eggshell: https://github.com/bird-house/eggshell
- Cookiecutter: https://github.com/bird-house/cookiecutter-birdhouse

8.6 Bonn (August 2016, v0.4.0)

Birdhouse was present at the FOSS4G 2016 in Bonn.

Highlighted Changes:

- Leaflet map with time-dimension plugin.
- using twitcher security proxy.
- using conda environments for each birdhouse compartment.
- using ansible to deploy birdhouse compartments.
- added weather-regimes and analogs detection processes.
- allow upload of files to processes.
- updated Phoenix user interface.

8.7 Paris (October 2015, v0.3.0)

- updated documents on readthedocs
- OAuth2 used for login with GitHub, Ceda, ...
- LDAP support for login
- using ncWMS and adagucwms
- register and use Thredds catalogs as data source
- publish local netcdf files and Thredds catalogs to birdhouse Solr
- qualtiy check processes added (cfchecker, qa-dkrz)
- generation of docker images for each birdhouse component
- using dispel4py as workflow engine in Malleefowl
- using Celery task scheduler/queue to run and monitor WPS processes
- improved Phoenix web client
- using birdy wps command line client

8.8 Paris (September 2014, v0.2.0)

- Phoenix UI as WPS client with ESGF faceted search component and a wizard to chain WPS processes
- PyWPS based processing backend with supporting processes of Malleefowl
- WMS service (inculded in Thredds) for visualization of NetCDF files
- OGC CSW catalog service for published results and OGC WPS services
- ESGF data access with wget and OpenID
- Caching of accessed files from ESGF Nodes and Catalog Service
- WPS processes: cdo, climate-indices, ensemble data visualization, demo processes
- IPython environment for WPS processes
- initial unit tests for WPS processes
- Workflow engine Restflow for running processing chains. Currently there is only a simple workflow used: get data with wget - process data.
- Installation based on anaconda and buildout
- buildout recipes (birdhousebuilder) available on PyPI to simplify installation and configuration of multiple WPS server
- Monitoring of all used services (WPS, WMS, CSW, Phoenix) with supervisor
- moved source code and documentation to birdhouse on GitHub

8.9 Helsinki (May 2014, v0.1.2)

- presentation of birdhouse at EGI, Helsinki
- stabilized birdhouse and CSC processes
- updated documenation and tutorials

8.10 Vienna (April 2014, v0.1.1)

- presentation of birdhouse at EGU, Vienna.
- “quality check” workflow for CORDEX data.

8.11 Hamburg (December 2013, v0.1.0)

- First presentation of Birdhouse at GERICS (German Climate Service Center), Hamburg.

CHAPTER

NINE

LICENSE

Birdhouse is Open Source and released under the Apache License, Version 2.0.

CHAPTER

TEN

GLOSSARY

Anaconda

Anaconda Python distribution Python distribution for large-scale data processing, predictive analytics, and scientific computing. https://www.continuum.io/

Binstar

Anaconda Server

Anaconda cloud Binstar is a service that allows you to create and manage public and private *Anaconda* package repositories. https://anaconda.org/ https://docs.continuum.io/

Bokeh Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, but also deliver this capability with high-performance interactivity over very large or streaming datasets. http://bokeh.pydata.org/en/latest/

Buildout *Buildout* is a Python-based build system for creating, assembling and deploying applications from multiple parts, some of which may be non-Python-based. It lets you create a buildout configuration and reproduce the same software later. http://www.buildout.org/en/latest/

CDO

Climate Data Operators *CDO* is a collection of command line Operators to manipulate and analyse Climate and NWP model Data. https://code.zmaw.de/projects/cdo

cfchecker The NetCDF Climate Forcast Conventions compliance checker. https://pypi.python.org/pypi/cfchecker

climate indice A climate index is a calculated value that can be used to describe the state and the changes in the climate system. http://icclim.readthedocs.io/en/latest/intro.html#climate-indices-label

CMIP5 In climatology, the Coupled Model Intercomparison Project (CMIP) is a framework and the analog of the Atmospheric Model Intercomparison Project (AMIP) for global coupled ocean-atmosphere general circulation models. https://en.wikipedia.org/wiki/Coupled_model_intercomparison_project

Conda The *conda* command is the primary interface for managing Anaconda installations. http://conda.pydata.org/docs/index.html

CORDEX The CORDEX vision is to advance and coordinate the science and application of regional climate downscaling through global partnerships. http://www.cordex.org/

COWS The COWS Web Processing Service (WPS) is a *generic* web service and offline processing tool developed within the Centre for Environmental Data Archival (CEDA). http://cows.ceda.ac.uk/cows_wps.html

CSW

Catalog Service *Catalog Service for the Web (CSW)*, sometimes seen as Catalog Service - Web, is a standard for exposing a catalogue of geospatial records in XML on the Internet (over HTTP). The catalogue is made up of records that describe geospatial data (e.g. KML), geospatial services (e.g. WMS), and related resources. https://en.wikipedia.org/wiki/Catalog_Service_for_the_Web

Dispel4py *Dispel4Py* is a Python library for describing abstract workflows for distributed data-intensive applications. http://www2.epcc.ed.ac.uk/~amrey/VERCE/Dispel4Py/index.html

Docker *Docker* - An open platform for distributed applications for developers and sysadmins. https://www.docker.com/

Docker Hub Docker Hub manages the lifecycle of distributed apps with cloud services for building and sharing containers and automating workflows. https://hub.docker.com/

Emu *Emu* is a Python package with some test proccess for *Web Processing Services*. http://emu.readthedocs.io/en/latest/

ESGF

Earth System Grid Federation An open source effort providing a robust, distributed data and computation platform, enabling world wide access to Peta/Exa-scale scientific data. http://esgf.llnl.gov/

GeoPython GitHub organisation of Python projects related to geospatial. https://geopython.github.io/

GeoServer GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. http://docs.geoserver.org/stable/en/user/index.html

GitHub GitHub is a web-based Git repository hosting service. https://github.com/ https://en.wikipedia.org/wiki/GitHub

Gunicorn Gunicorn *Green Unicorn* is a Python WSGI HTTP Server for UNIX. http://gunicorn.org/

Homebrew The missing package manager for OS X. http://brew.sh/

ICCLIM

Indice Calculation CLIMate *ICCLIM* (Indice Calculation CLIMate) is a Python library for computing a number of *climate indices*. http://icclim.readthedocs.io/en/latest/

Linuxbrew Linuxbrew is a fork of Homebrew, the Mac OS package manager, for Linux. http://brew.sh/linuxbrew/

Malleefowl *Malleefowl* is a Python package to simplify the usage of *Web Processing Services*. http://malleefowl.readthedocs.io/en/latest/

NetCDF NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. https://en.wikipedia.org/wiki/NetCDF

Nginx nginx [engine x] is an HTTP and reverse proxy server. http://nginx.org/

ocgis

OpenClimateGIS *OpenClimateGIS* (OCGIS) is a Python package designed for geospatial manipulation, subsetting, computation, and translation of climate datasets stored in local *NetCDF* files or files served through *THREDDS* data servers. https://www.earthsystemcog.org/projects/openclimategis/ https://github.com/NCPP/ocgis

OGC

Open Geospatial Consortium The *Open Geospatial Consortium (OGC)* is an international voluntary consensus standards organization, originated in 1994. https://en.wikipedia.org/wiki/Open_Geospatial_Consortium, http://www.opengeospatial.org/standards/wps

OpenID OpenID (OID) is an open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain co-operating sites (known as Relying Parties or RP) using a third party service. https://en.wikipedia.org/wiki/OpenID, http://openid.net/

OWSLib OWSLib is a Python package for client programming with *Open Geospatial Consortium* web service interface standards, and their related content models. OWSLib has *WPS* client library which is used in Birdhouse to access WPS services. http://geopython.github.io/OWSLib/, http://geopython.github.io/OWSLib/#wps

Phoenix Pyramid *Phoenix* is a web-application build with the Python web-framework pyramid. Phoenix has a user interface to make it easier to interact with *Web Processing Services*. http://pyramid-phoenix.readthedocs.io/en/latest

PyCSW pycsw is an *OGC* CSW server implementation written in Python. Started in 2010 (more formally announced in 2011), pycsw allows for the publishing and discovery of geospatial metadata, providing a standards-based metadata and catalogue component of spatial data infrastructures. http://pycsw.org/, https://github.com/geopython/pycsw

PyPi

Python Package Index The Python Package Index is a repository of software for the Python programming language. https://pypi.python.org/pypi

Pyramid Pyramid is a Python web framework. http://www.pylonsproject.org/

PyWPS *Python Web Processing Service* is an implementation of the *Web processing Service* standard from *Open Geospatial Consortium*. http://pywps.org/

RestFlow *RestFlow* is a dataflow programming language and runtime engine designed to make it easy for scientists to build and execute computational pipelines. https://github.com/restflow-org/restflow/wiki

Supervisor Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. http://supervisord.org/

Taverna *Taverna* is an open source and domain-independent Workflow Management System – a suite of tools used to design and execute scientific workflows. http://www.taverna.org.uk/

TDS

THREDDS The THREDDS Data Server (TDS) is a web server that provides metadata and data access for scientific datasets, using a variety of remote data access protocols. http://www.unidata.ucar.edu/software/thredds/current/tds/

VisTrails *VisTrails* is an open-source scientific workflow and provenance management system that supports data exploration and visualization. http://www.vistrails.org/index.php/Main_Page

WMS

Web Mapping Service A Web Map Service (WMS) is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database. https://en.wikipedia.org/wiki/Web_Map_Service

Workflow

Workflow Management System A workflow management system (WfMS) is a software system for the set-up, performance and monitoring of a defined sequence of tasks, arranged as a workflow. https://en.wikipedia.org/wiki/Workflow_management_system

WPS

Web Processing Service WPS is an open standard to search and run processes with a simple web-based interface. See: wps_tutorial.

WSGI WSGI is an interface specification by which server and application communicate. http://wsgi.tutorial.codepoint.net/

x509 In cryptography, X.509 is an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI). https://en.wikipedia.org/wiki/X.509

XML-RPC It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. http://xmlrpc.scripting.com/default.html

CHAPTER

ELEVEN

USEFUL LINKS

11.1 WPS Documentation

- What is WPS?
- WPS on OSGeo Live
- WPS tutorial
- OGC Web Processing Service Standard
- PyWPS Wiki
- GeoServer tutorial

Talks:

- The WPS 2.0 standard (preliminary information)
- WPS Application Patterns
- Using WPS (PyWPS) with Taverna Orchestration
- Pywps a tutorial for beginners and developers
- Zoo presentation foss4g.jp-2011

11.2 WPS Software

WPS Server Software:

- *PyWPS*
- GeoServer - http://docs.geoserver.org/stable/en/user/services/wps/index.html
- Zoo - `http://www.zoo-project.org/`
- *COWS*
- Deegree - http://www.deegree.org/
- 52 North - http://52north.org/communities/geoprocessing/wps/

WPS Client Software:

- *OWSLib Python Client*
- OpenLayers WPS Plugin - http://dev.openlayers.org/docs/files/OpenLayers/WPSClient-js.html
- GeoTools WPS Module - http://docs.geotools.org/latest/userguide/unsupported/wps.html

- 52 North Java Client - http://52north.org/communities/geoprocessing/wps/index.html
- 52 North Javascript Client - http://geoprocessing.demo.52north.org:8080
- WPS Javascript Client by Boundless - https://github.com/boundlessgeo/wps-gui

QGIS Desktop GIS with wps plugins:

- http://www.qgis.org/en/site/
- http://plugins.qgis.org/plugins/wps/
- http://geolabs.fr/plugins.xml

uDig Desktop GIS with wps plugins:

- http://udig.refractions.net/
- https://udig.github.io/docs/user/reference/Using%20the%20WPS%20plugin.html
- https://github.com/52North/uDig-WPS-plugin (outdated)

11.3 WMS Software

WMS server:

- ncWMS2 - http://reading-escience-centre.github.io/edal-java/
- adaguc - http://adaguc.knmi.nl/
- sci-wms - http://sci-wms.github.io/sci-wms/

WMS clients:

- OpenLayers - http://openlayers.org/
- **Leaflet - http://leafletjs.com/**
  - time dimension - http://apps.socib.es/Leaflet.TimeDimension/examples/
- GeoExt - http://geoext.github.io/geoext2/

11.4 Scientific Workflow Tools

Workflow Engines:

- *Dispel4py*
- *RestFlow*
- *Taverna*
- *VisTrails*
- Kepler - https://kepler-project.org/
- KNIME - http://www.knime.org/

Taverna with WPS:

- `http://rsg.pml.ac.uk/wps/generic.cgi?request=GetCapabilities&service=WPS`
- https://www.youtube.com/watch?v=JNAtoOejVIo
- https://taverna.incubator.apache.org/introduction/services-in-taverna.html

- https://github.com/myGrid/small-area-estimator
- http://comments.gmane.org/gmane.science.biology.informatics.taverna.user/1415
- http://dev.mygrid.org.uk/wiki/display/developer/SCUFL2

VisTrails with WPS:

- https://github.com/ict4eo/eo4vistrails
- http://proj.badc.rl.ac.uk/cows/wiki/CowsWps/CDOWPSWorkingGroup/WPSAndWorkflows
- `http://www.kitware.com/source/home/post/105`

Kepler with WPS:

- https://kepler-project.org/users/sample-workflows

Workflows with PyWPS:

- https://github.com/AnnaHomolka/PyWPS/blob/master/doc/tutorial_process_chaining.pdf

Other Workflow Engines:

- `http://www.yawlfoundation.org/`
- https://en.wikipedia.org/wiki/Scientific_workflow_system
- http://airavata.apache.org/
- http://search.cpan.org/~nuffin/Class-Workflow-0.11/

11.5 Scientific Python

- Anaconda - https://www.continuum.io/downloads

Completely free enterprise-ready Python distribution for large-scale data processing, predictive analytics, and scientific computing

- pandas - http://pandas.pydata.org/

Python Data Analysis Library

11.6 Python in Climate Science

- OpenClimateGIS - https://earthsystemcog.org/projects/openclimategis/

OpenClimateGIS is a Python package designed for geospatial manipulation, subsetting, computation, and translation of climate datasets stored in local NetCDF files or files served through THREDDS data servers. [..]

- ICCLIM (i see clim . . .) - https://github.com/cerfacs-globc/icclim

Python library for climate indices calculation. Documentation at http://icclim.readthedocs.io/en/latest/

11.7 Python Web Frameworks and Utils

- Pyramid - http://www.pylonsproject.org/
- Authomatic - http://peterhudec.github.io/authomatic/
- Bootstrap - http://getbootstrap.com/
- Bootstrap Tutorial - http://www.w3schools.com/bootstrap/default.asp
- Deform - https://github.com/Pylons/deform
- Deform with Bootstrap demo - http://deform2demo.repoze.org/
- Colander - http://docs.pylonsproject.org/projects/colander/en/latest/index.html
- TinyMCE - https://www.tinymce.com/
- Font Awesome - http://fontawesome.io/
- Leaflet - http://leafletjs.com/
- Leaflet TimeDimension - http://apps.socib.es/Leaflet.TimeDimension/examples/

11.8 Example WPS Services

List of available Web Processing Services:

- Zoo WPS for PublicaMundi project - `http://zoo.dev.publicamundi.eu/cgi-bin/zoo_loader.cgi?service=WPS&version=1.0.0&request=GetCapabilities`
- GeoServer Demo WPS - `http://demo.opengeo.org/geoserver/wps?request=GetCapabilities&service=WPS`
- USGS Geo Data Portal- `http://cida.usgs.gov/climate/gdp/process/WebProcessingService`
- KNMI climate4impact Portal - `http://climate4impact.eu//impactportal/WPS?request=GetCapabilities&service=WPS`
- BADC CEDA - `http://ceda-wps2.badc.rl.ac.uk/wps?request=GetCapabilities&service=WPS`
- delatres - `http://dtvirt5.deltares.nl/wps/?Request=GetCapabilities&Service=WPS`
- 52 North - `http://geoprocessing.demo.52north.org:8080/52n-wps-webapp-3.3.1/WebProcessingService?Request=GetCapabilities&Service=WPS`
- 52 North - `http://geoprocessing.demo.52north.org:8080/52n-wps-webapp-3.3.1-gt/WebProcessingService?Request=GetCapabilities&Service=WPS`
- ZOO Demo WPS - `http://zoo-project.org/cgi-bin/zoo_loader3.cgi?Request=GetCapabilities&Service=WPS`
- British Antarctic Survey WPS for Meteorological Data - `http://sosmet.nerc-bas.ac.uk:8080/wpsmet/WebProcessingService?Request=GetCapabilities&Service=WPS`
- PyWPS Demo - `http://apps.esdi-humboldt.cz/pywps/?request=GetCapabilities&service=WPS&vers0.0`

11.9 Alternatives to WPS

- XML-RPC: Simple cross-platform distributed computing, based on the standards of the Internet. - http://xmlrpc.scripting.com/
- Swagger is a simple yet powerful representation of your RESTful API. - http://swagger.io/

11.10 Related Projects

- http://geopython.github.io/
- http://geonode.org/
- http://esgf.llnl.gov/
- http://climate4impact.eu/impactportal/general/index.jsp
- http://adaguc.knmi.nl/
- http://wps-web1.ceda.ac.uk/ui/home
- `https://freva.met.fu-berlin.de/`
- https://climate.apache.org/

11.11 References

This list [Schnase2016] has no claim to be complete.

BIBLIOGRAPHY

[ELH+18] C. Ehbrecht, T. Landry, N. Hempelmann, D. Huard, and S. Kindermann. Projects based on the web processing service framework birdhouse. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W8:43–47, 2018. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W8/43/2018/, doi:10.5194/isprs-archives-XLII-4-W8-43-2018.

[HEAC+18] N. Hempelmann, C. Ehbrecht, C. Alvarez-Castro, P. Brockmann, W. Falk, J. Hoffmann, S. Kindermann, B. Koziol, C. Nangini, S. Radanovics, R. Vautard, and P. Yiou. Web processing service for climate impact and extreme weather event analyses. flyingpigeon (version 1.0). *Computers & Geosciences*, 110(Supplement C):65 – 72, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0098300416302801, doi:https://doi.org/10.1016/j.cageo.2017.10.004.

[JGG+14] C. Jung, M. Gasthuber, A. Giesler, M. Hardt, J. Meyer, F. Rigoll, K. Schwarz, R. Stotzka, and A. Streit. Optimization of data life cycles. *Journal of Physics: Conference Series*, 513(3):032047, 2014. URL: http://stacks.iop.org/1742-6596/513/i=3/a=032047.

[JMS17] Christopher Jung, Jörg Meyer, and Achim Streit, editors. *Helmholtz Portfolio Theme Large-Scale Data Management and Analysis (LSDMA)*. KIT Scientific Publishing, Karlsruhe, 2017. ISBN 978-3-7315-0695-9. 46.12.02; LK 01. doi:10.5445/KSP/1000071931.

INDEX

V

W

X