Birdhouse Architecture

Carsten Ehbrecht

ehbrecht@dkrz.de

German Climate Computing Center (DKRZ)

February 2016

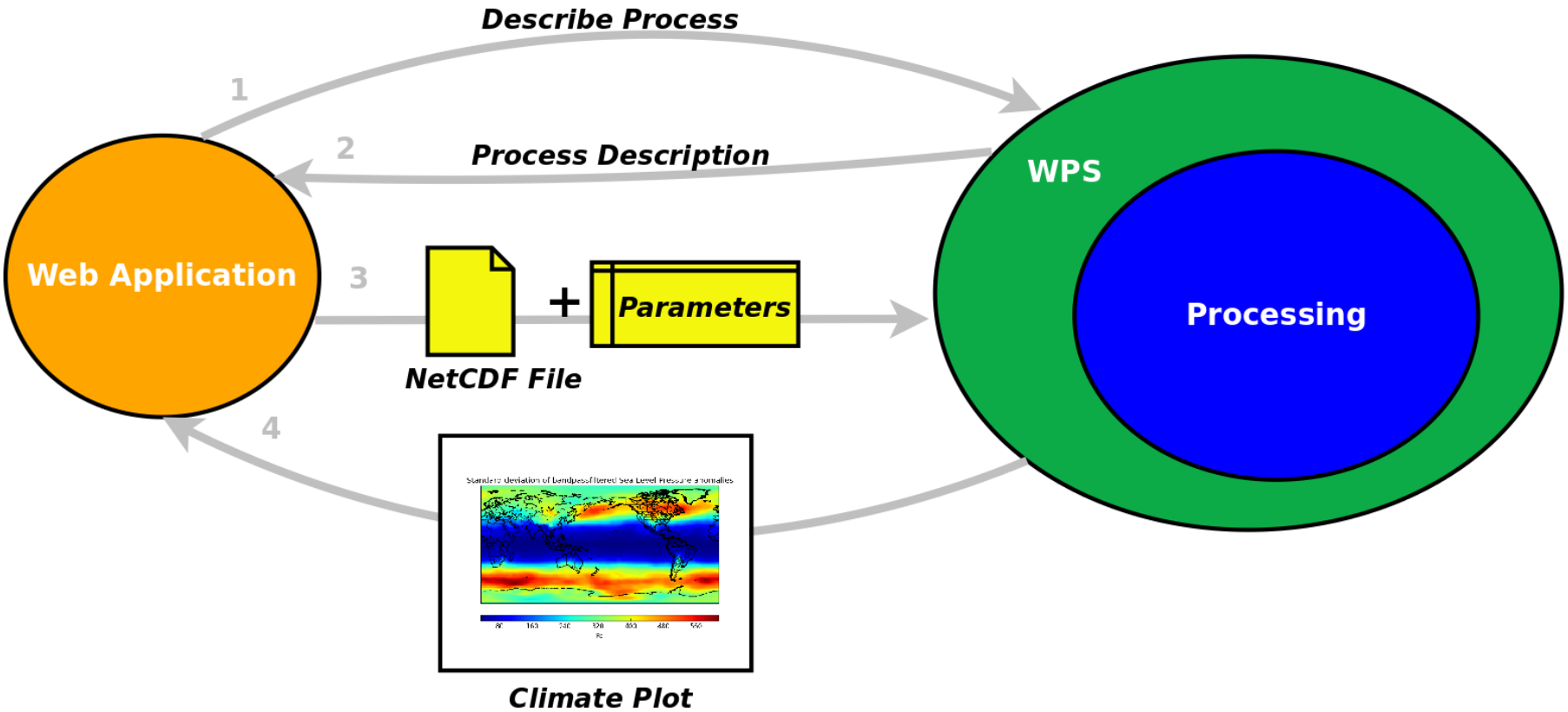
Outline

Overview

Web Processing Service

A web service interface to standardize the way that algorithms are made available on the Internet

WPS Use Case


Describe Process
1
2
Process Description
Web Application
3
+
Parameters
NetCDF File
WPS
Processing
4
Climate Plot

What does WPS provide?

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

Enable your code as WPS process

Use wps decorator for your function

```
1 @wps
2 def myplot(nc_file, variable):
3     """
4     nc_file application/netcdf
5     variable string
6     """
7     # plot variable of nc_file
8     return plot.png
```

Execute your function with WPS

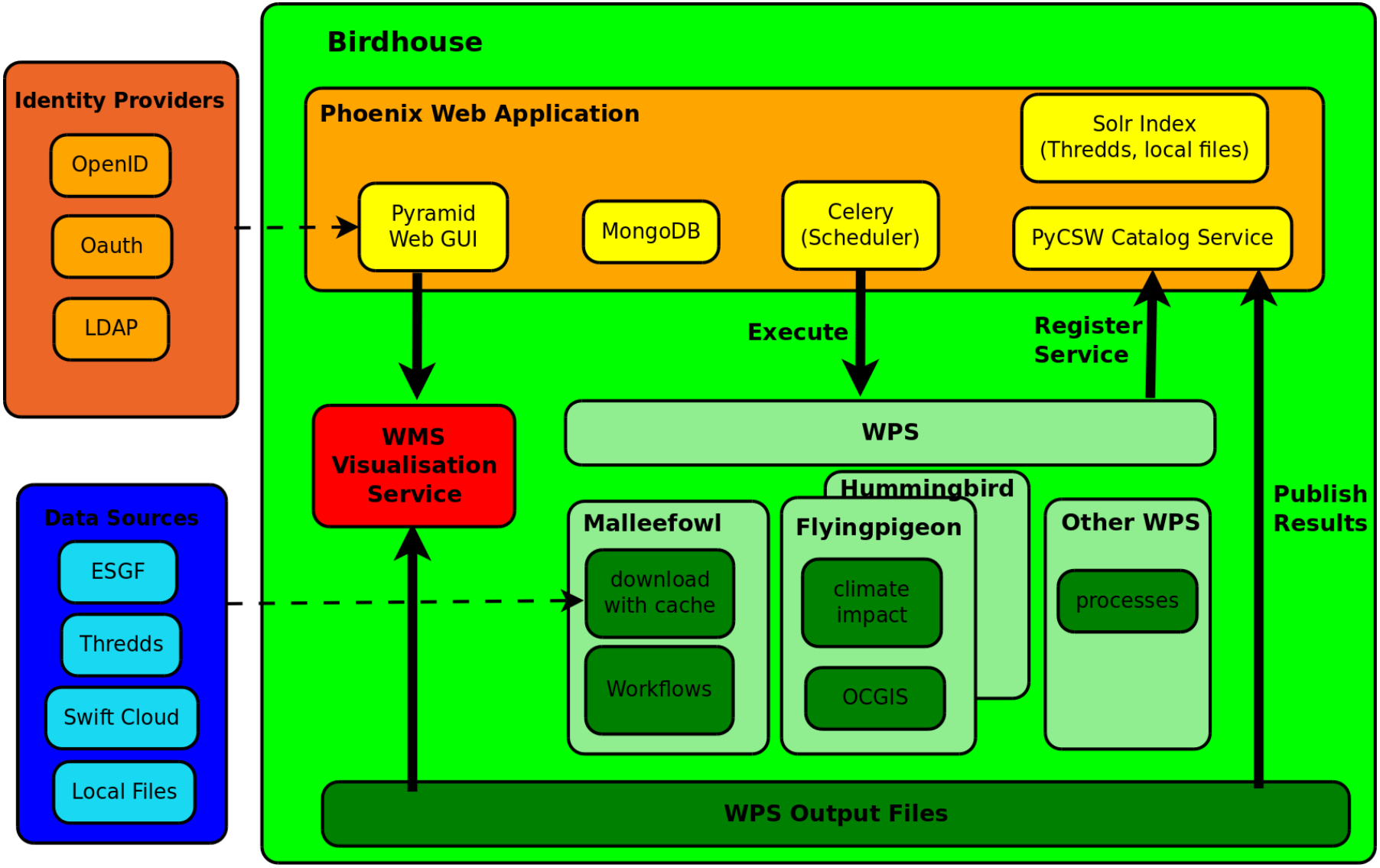
```
http://localhost/wps?service=WPS&version=1.0.0 \
   &request=execute \
   &identifier=myplot \
   &DataInputs=nc_file=http://;variable=tas
```

Overview

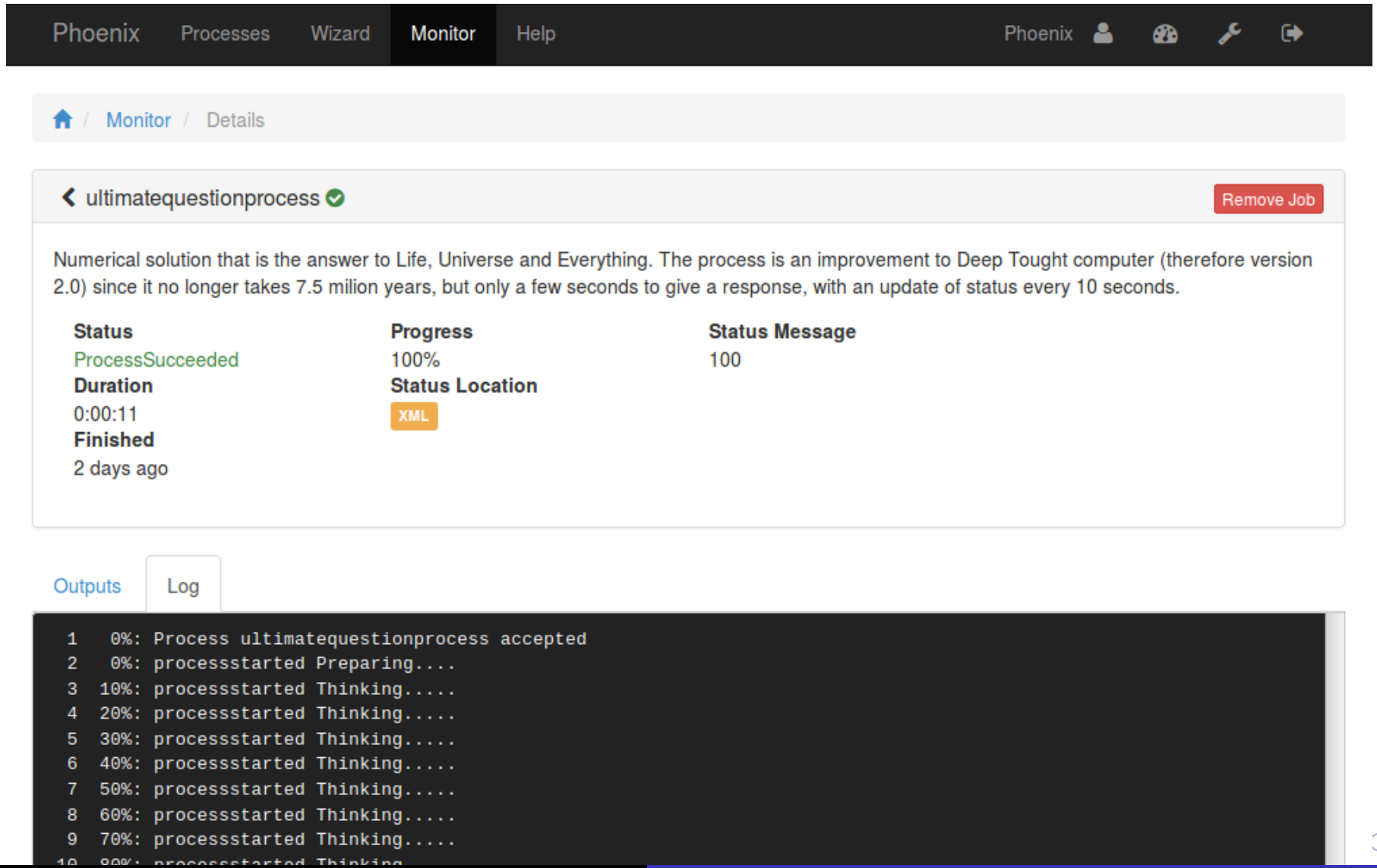
Birdhouse Aims

- Making it easier to setup WPS services (Birdhouse-Builder).
- Providing Python library and WPS processes to access climate data.
- Providing a workflow to chain data fetching and data processing.
- Providing a web and command line client to interact with WPS services.

Birdhouse Components

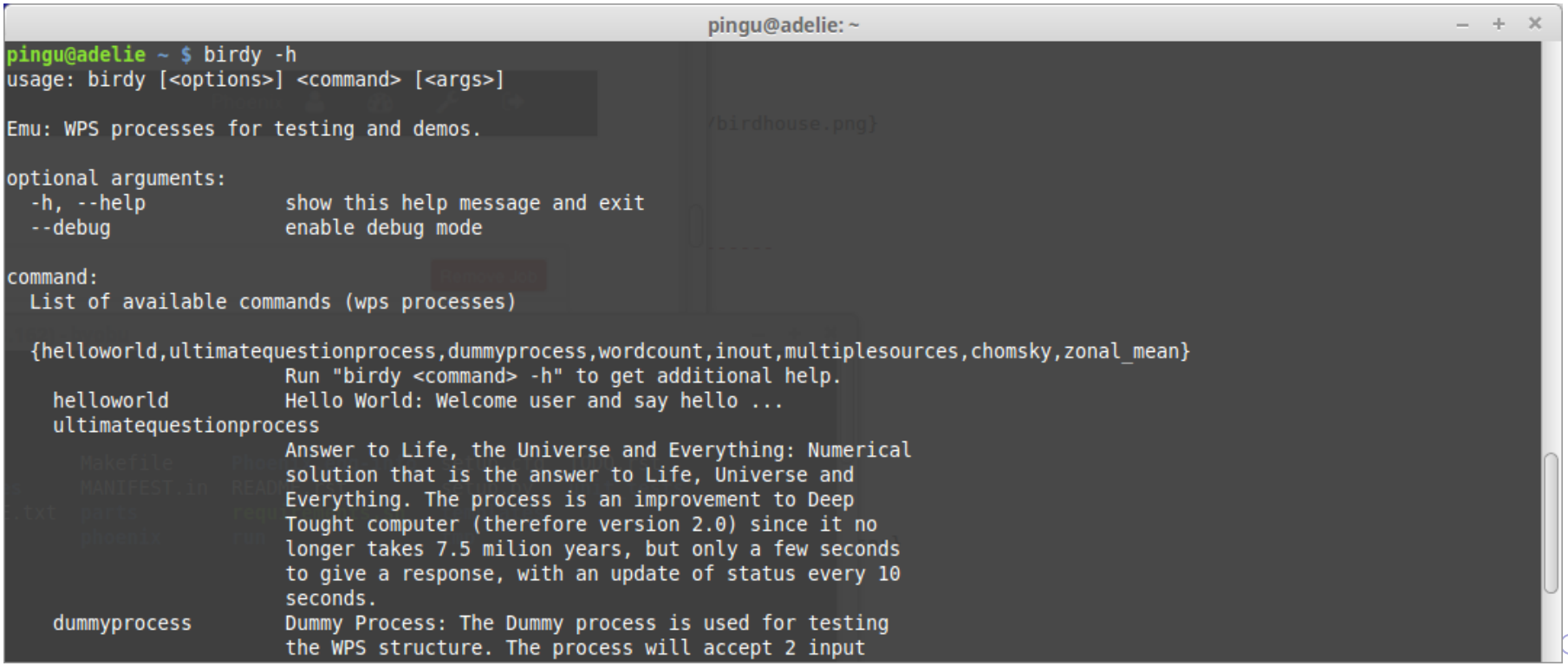


Phoenix web-based WPS client



Birdy command line WPS client

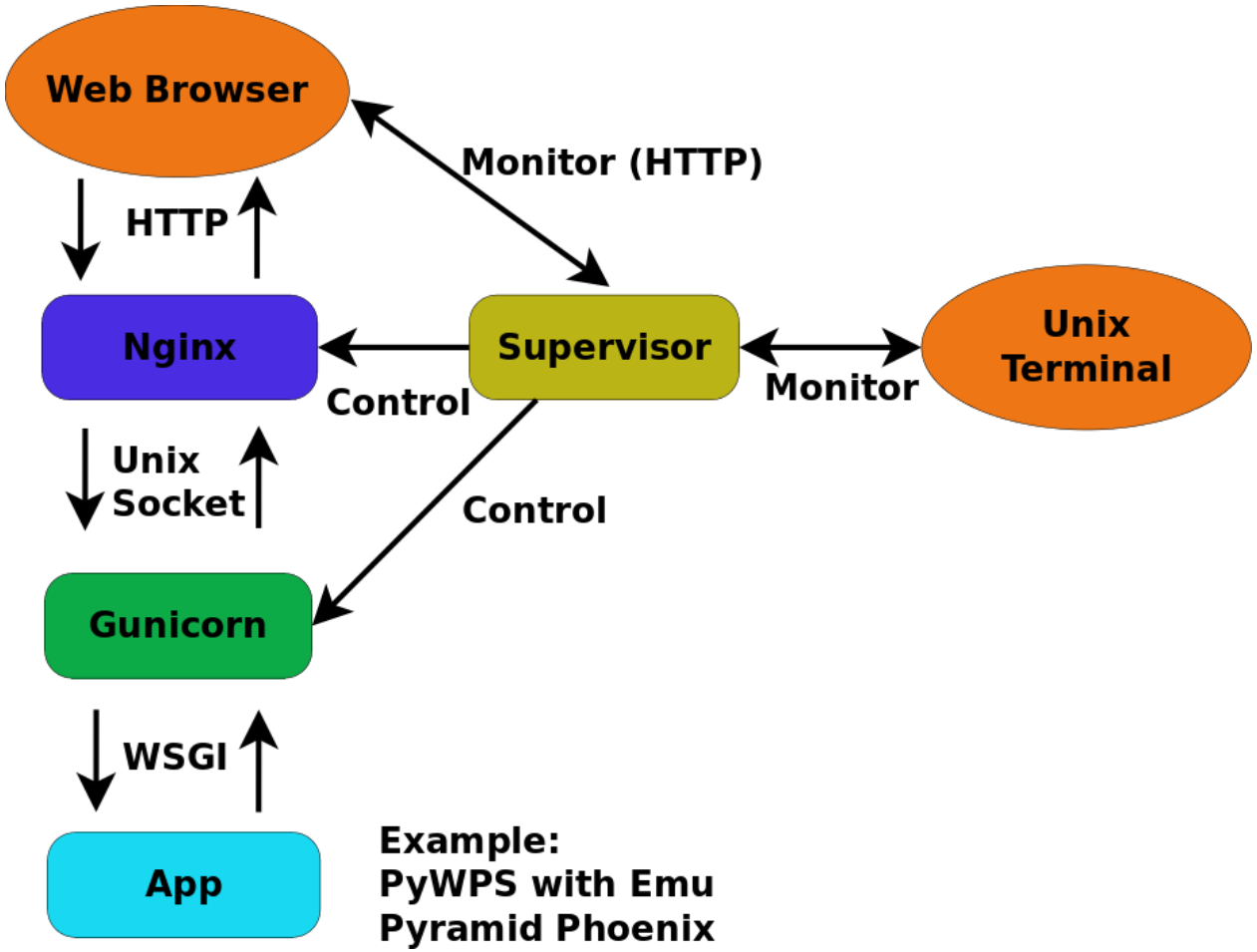
```
>> conda install -c birdhouse birdhouse-birdy
>> export WPS_SERVICS=http://localhost:8094/wps
>> birdy -h
```



Birdhouse WPS serivces

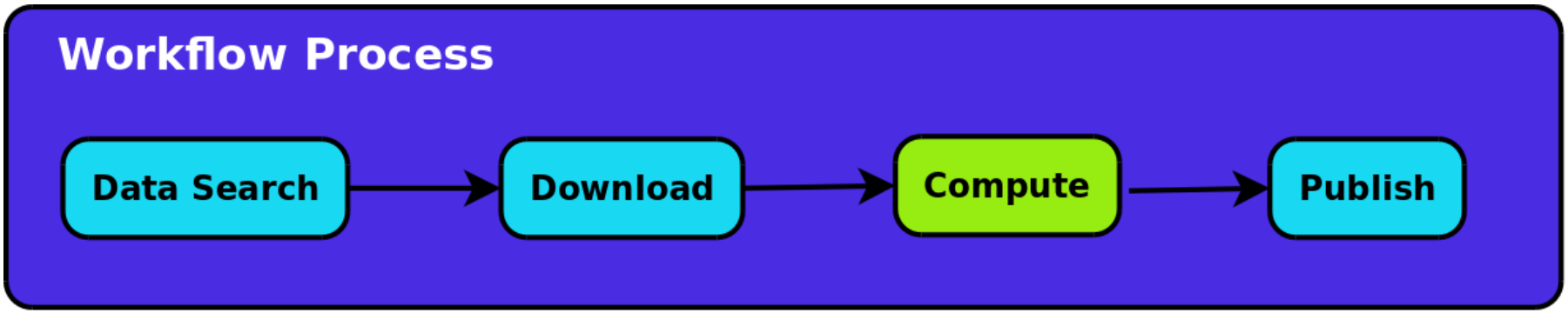
- Malleefowl (for internal use): download data with cache, workflows ...
- Emu: processes for testing and demo
- Hummingbird: processes for general tools used in the climate science community like cdo and cfchecker
- Flyingpigeon: processes for climate data, indices and extreme events

WSGI Application controlled with Supervisor



Workflow Process in Birdhouse

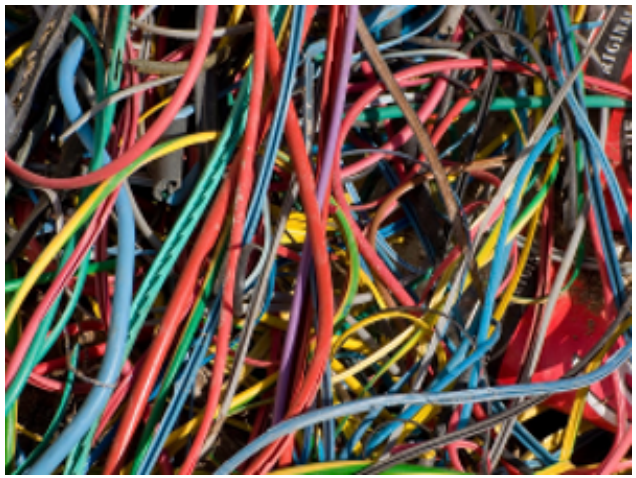
- the chain of WPS processes is controlled by a Workflow Engine (dispel4py)
- The Workflow Process itself is a WPS process
- data-search, download and publish are internal processes (provided by Malleefowl)
- Compute is a process choosen by the user ... for example cfchecker
- The Phoenix wizard is used to collect the parameters for the workflow process



Overview

Why Conda and Buildout?

- many components: WPS, WMS, web-server, solr, ...
- lots of dependencies: cdo, cfchecker, ocgis, numpy, R, ...
- many different kinds of config files need to be configured
- installation needs to be reproducible at different locations
- should work with different Linux distributions (Centos, Fedora, Debian, Ubuntu, ...)



conda package manager

- originally for python ... but has a general concept
- does not need admin rights
- manages dependencies

install from birdhouse channel

```
>> conda install -c birdhouse pywps cdo
```

create conda environment=emu

```
>> conda create -n emu -c birdhouse \
       python=2.7 cdo pywps
```

conda recipe

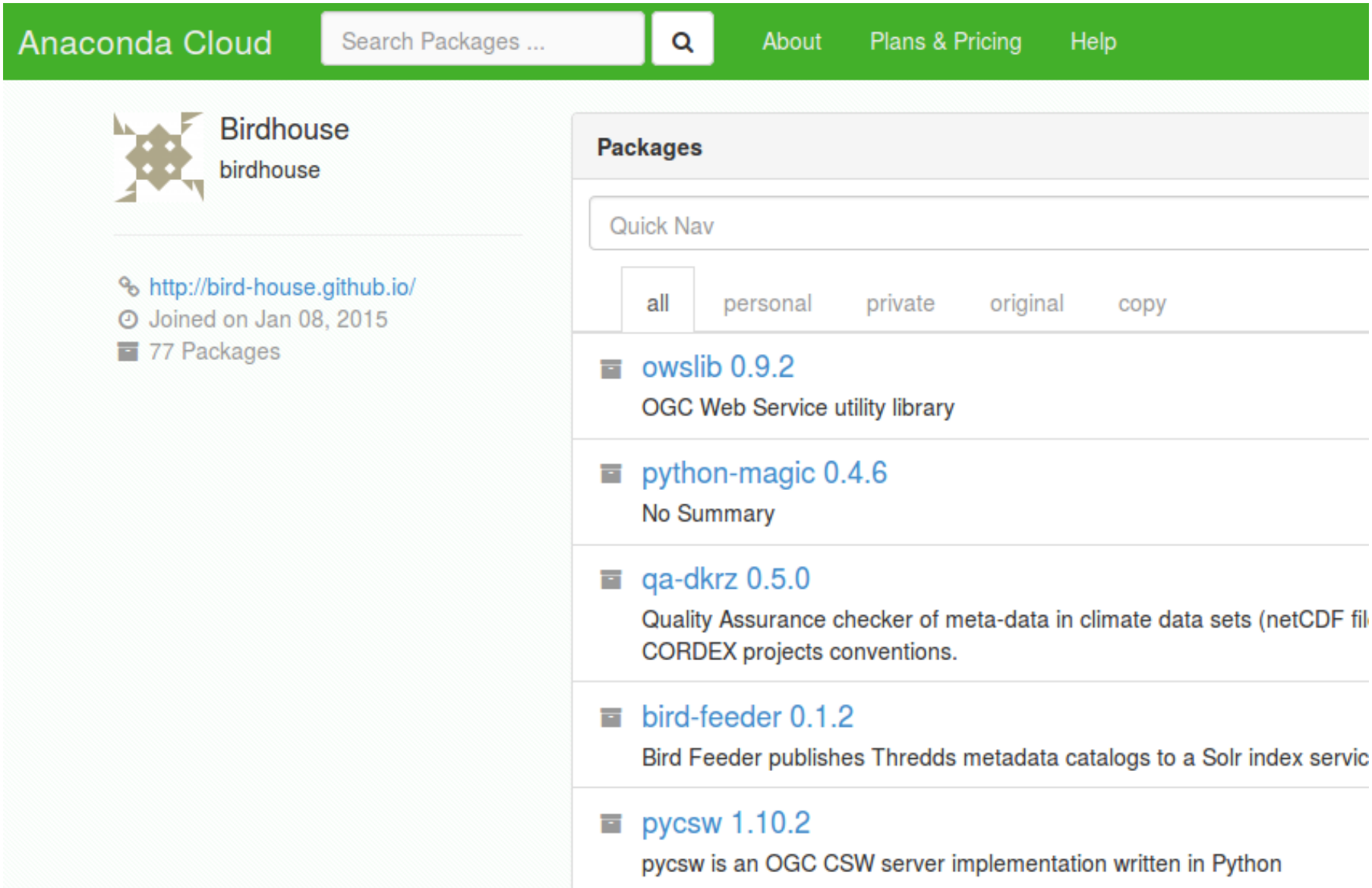
meta.yaml

```
1 package:
2   name: owslib
3   version: !!str 0.9.2
4 source:
5   url:
6 requirements:
7   run:
8     python
9     lxml
```

build conda package

```
>> ls
meta.yaml build.sh
>> conda build .
```

Anaconda Cloud



Buildout

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

Buildout configuration

buildout.cfg

```
1  [buildout]
2  parts = conda pywps
3
4  [settings]
5  hostname = localhost
6
7  [conda]
8  recipe = birdhousebuilder.recipe.conda
9  pkgs = ipython cdo
10
11 [pywps]
12 recipe = birdhousebuilder.recipe.pywps
13 title = Emu WPS
14 hostname = ${settings:hostname}
```

Buildout Recipe

birdhousebuilder.recipe.pywps

```
1  from birdhousebuilder.recipe import conda,
       ↪ supervisor, nginx
2
3  class PyWpsRecipe(object):
4      def __init__(self, buildout, name, options):
5          # set default options
6
7      def install(self):
8          # install pywps with nginx, gunicorn and
               ↪ supervisor
9          # generate config files from templates
               ↪ according the options
10
11     def update(self):
12         # update configuration
```

Install Birdhouse Component with Buildout

- for convienience there is a Makefile to call the buildout commands
- all Birdhouse components (WPS, Phoenix) are installed in the same way

First installation

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

Update configuration like hostname, port

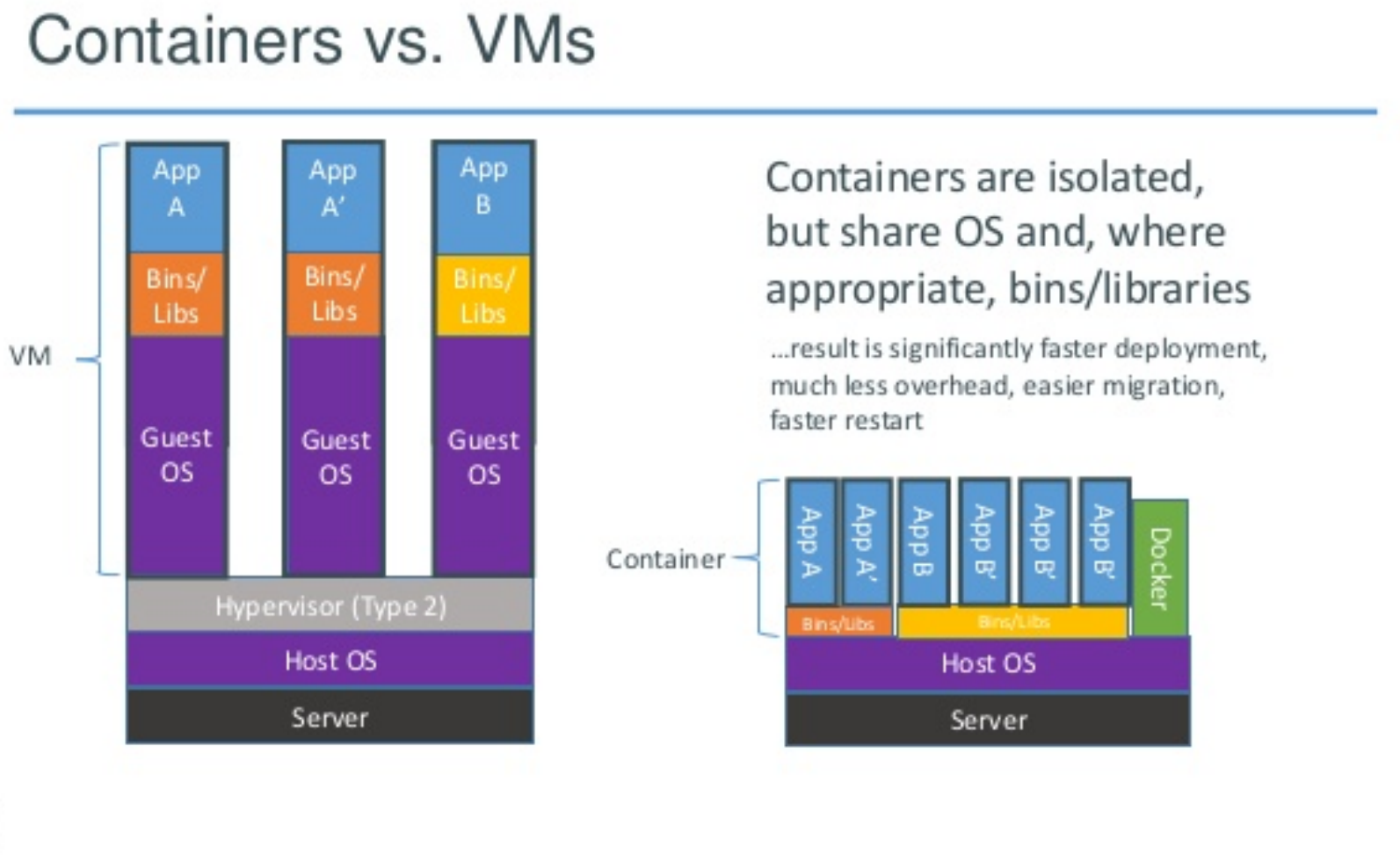
```
>> vim custom.cfg
>> make update
>> make restart
```

Overview

What is Docker?

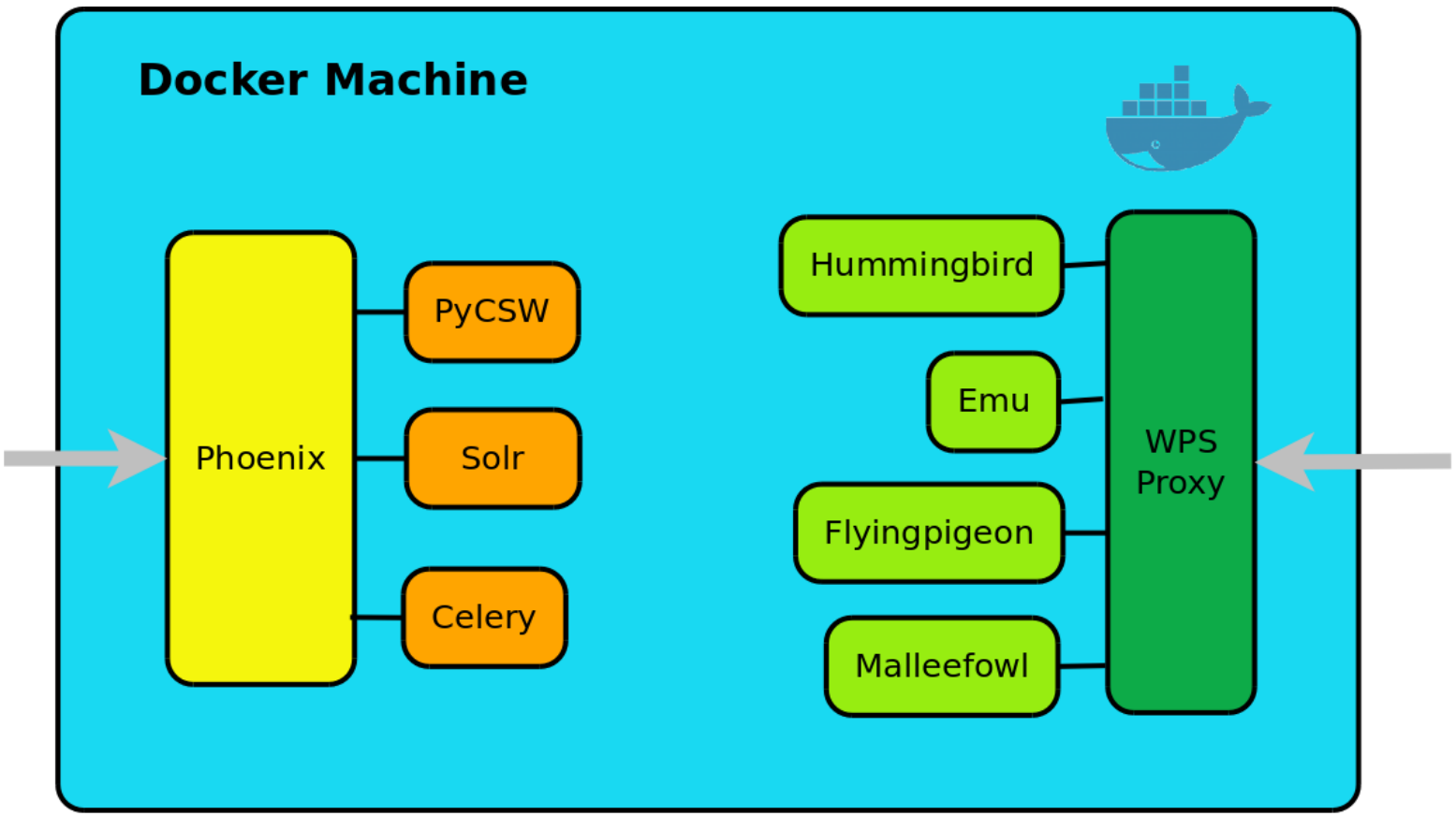
- a lightweight Virtual-Machine using Linux Containers
- isolated environment for your Linux installation
- runs on the same hardware as the host
- run latest Ubuntu on an older Centos
- rapid startup time
- only changed parts of docker image need to loaded on update

Containers vs VMs (slide taken from docker)



Deploy Birdhouse with Docker

- each service is running in a Docker container
- docker containers can be *linked* to other containers
- only some containers (Phonix, WPS Proxy) need to be exposed to external use



Dockerfile

```
1  FROM ubuntu:14.04
2
3  # Add application sources
4  ADD . /opt/birdhouse
5
6  # cd into application
7  WORKDIR /opt/birdhouse
8
9  # Install system dependencies
10 RUN bash bootstrap.sh -i && bash requirements.sh
11
12 # Run install
13 RUN make clean install
14
15 # Volume for data, cache, logfiles, ...
16 VOLUME /data
17
18 # Ports used in birdhouse
19 EXPOSE 8090 8094
20
21 # Update config and start supervisor ...
22 CMD ["make", "start"]
```

Try a Docker ...

Images are available on DockerHub

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

Start a docker image with Emu WPS

```
>> docker run -it -p 8090:8090 -p 8094:8094 \
        --name=emu_wps birdhouse/emu
```

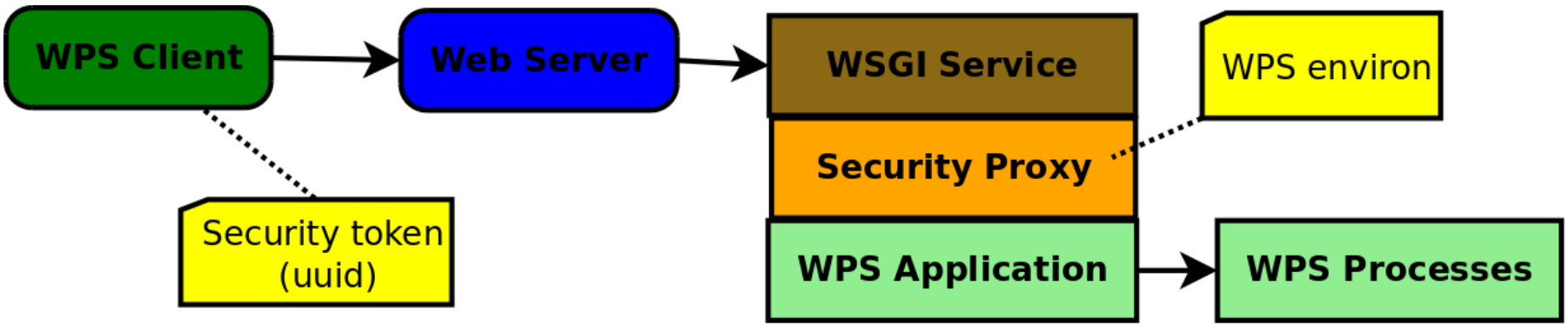
Run WPS GetCapabilities Request

```
http://localhost:8094/wps? \
   service=WPS&version=1.0.0&request=getcapabilities
```

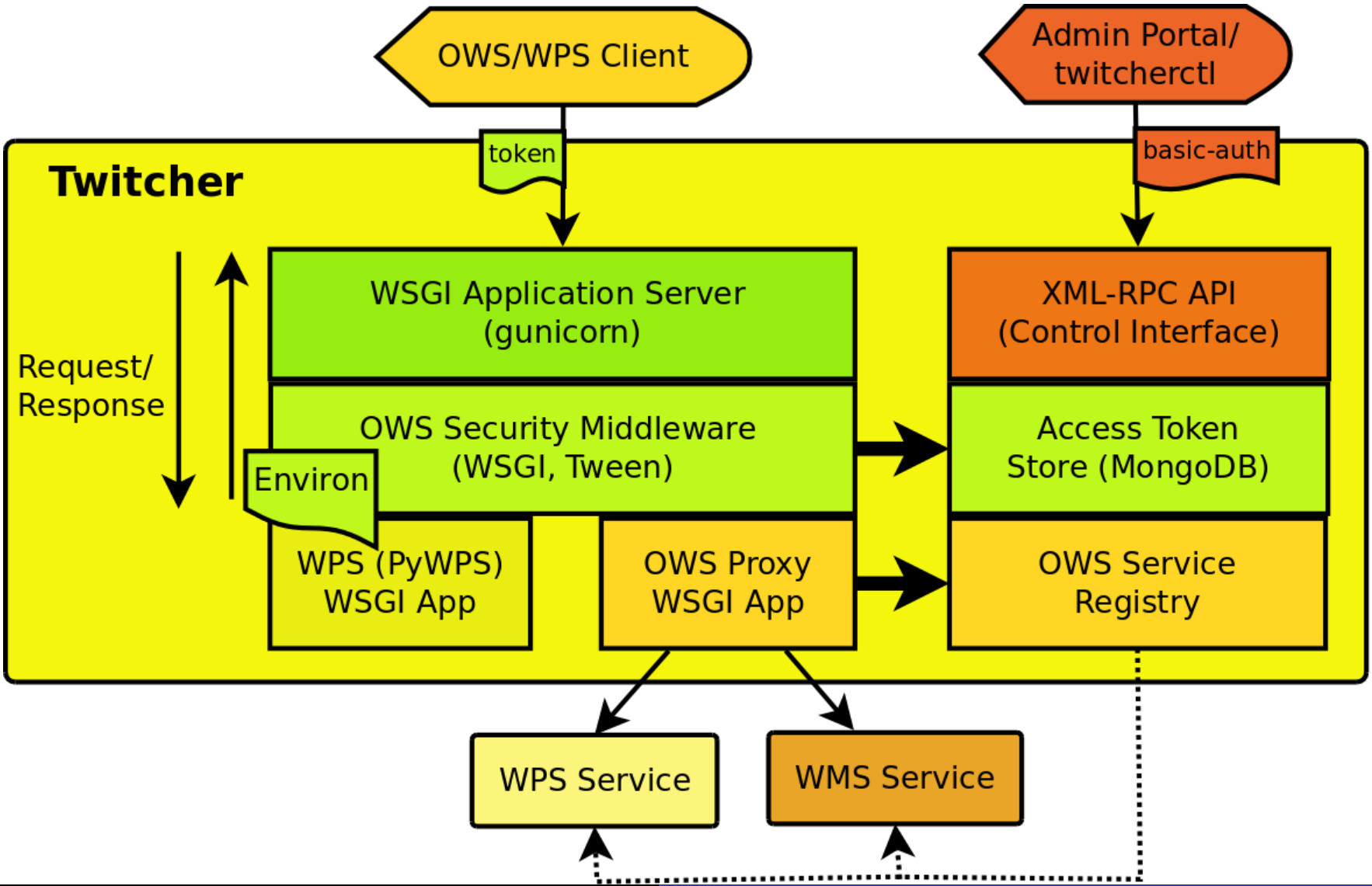
Overview

WPS Security Proxy

- using string token (uuid) as part of URL or in request header to protect WPS execute access
- X509 certificates to access (remote) data from ESGF are provided by proxy (using environ)
- implemented as WSGI application layer



Twitcher OWS Security Proxy (prototype)



Best Practises to make WPS interchangeable

- support of complete WPS protocol (literal type, complex types, ...)
- separation of WPS definition and functional code (provided as Python library)
- convenience and integration code provided as library (e.a. Malleefowl provides functions used by WPS processes)
- using WPS profiles (common WPS process definitions)
- use self-describing possibilities of WPS
- parameters relevant for the process should be part of the process definition

Overview

Further Reading I

Birdhouse

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

Buildout

http://www.buildout.org/

Anaconda

https://www.continuum.io/why-anaconda

Evaluation of WPS Frameworks

http://www.slideshare.net/mepa1363/foss4g-ebrahim

Web Processing Service

http://www.slideshare.net/GasperiJerome/20130530-web-processing-service-cct-cloud-toulouse-29423710

The End