D2.8 Development environment design

WP2 – EO Platform

Deliverable Lead: ATOS FR

Dissemination Level: Public

Deliverable due date: 28/02/2017

Actual submission date: 31/05/2017

Version 1.1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687275

Document Control Page	
Title	D2.8 Development environment design
Creator	Fabien CASTEL (ATOS FR)
Description	This document describes and gives a preliminary design of the interactive development environment that the EO4wildlife platform aims to provide.
Publisher	EO4wildlife Consortium
Contributors	Fabien CASTEL (ATOS FR)
Creation date	12/05/2017
Type	Text
Language	en-GB
Rights	copyright “EO4wildlife Consortium”
Audience	☒ Public ☐ Confidential ☐ Classified
Status	☐ In Progress ☐ For Review ☐ For Approval ☒ Approved

Disclaimer

This deliverable is subject to final acceptance by the European Commission.

The results of this deliverable reflect only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Statement for open documents

Table of Contents

List of Figures

List of Tables

EO4wildlife Project Overview

EO4wildlife main objective is to bring large number of multidisciplinary scientists such as biologists, ecologists and ornithologists around the world to collaborate closely together while using European Sentinel Copernicus Earth Observation more heavily and efficiently.

In order to reach such important objective, an open service platform and interoperable toolbox will be designed and developed. It will offer high level services that can be accessed by scientists to perform their respective research. The platform front end will be easy-to-use, access and offer dedicated services that will enable them process their geospatial environmental stimulations using Sentinel Earth Observation data that are intelligently combined with other observation sources.

Specifically, the EO4wildlife platform will enable the integration of Sentinel data, ARGOS archive databases and real time thematic databank portals, including Wildlifetracking.org, Seabirdtracking.org, and other Earth Observation and MetOcean databases; locally or remotely, and simultaneously.

EO4wildlife research specialises in the intelligent management big data, processing, advanced analytics and a Knowledge Base for wildlife migratory behaviour and trends forecast. The research will lead to the development of web-enabled open services using OGC standards for sensor observation and measurements and data processing of heterogeneous geospatial observation data and uncertainties.

EO4wildlife will design, implement and validate various scenarios based on real operational use case requirements in the field of wildlife migrations, habitats and behaviour. These include:

- Management tools for regulatory authorities to achieve real-time advanced decision-making on the protection of protect seabird species;
- Enhancing scientific knowledge of pelagic fish migrations routes, reproduction and feeding behaviours for better species management;
- Enable researchers better understand the movement behaviour of sea turtle populations; and
- Setting up tools to assist marine protected areas and management.

Abbreviations and Glossary

A common glossary of terms for all EO4wildlife deliverables, as well as a list of abbreviations, can be found in the public document “EO4wildlife Glossary” available at EO4wildlife.eu.

Executive Summary

This document provides a description of the interactive and remote development environment that the EO4wildlife platform provides to its users.

In a first section the functional needs are summed up. In the next sections, more technical aspects are addressed. The second section describes the technical architecture of the Notebook technologies in general and Jupyter Notebook in particular. The third section address the multi-user aspect and how the solution provided will handle it. The fourth section gives an insight on how the Notebook technology will be deployed on the EO4wildlife platform, and lastly the fifth section describes some features that will enrich the Notebook solution we provide.

Task 2.6 (Platform Development Environment) was supposed to start in January, but WP2 needed first the definition of the use cases and the requirements, and focused in the first version of the architecture. The beginning of this task was postponed to the second semester of the year, and as a result, D2.8 initially scheduled at M14, was delayed to M17 (May 2017).

1 Functional needs

1.1 Low level interaction with the platform

The EO4widlife main feature is to allow user executing processing workflow on a remote platform. Workflows are not strictly speaking black boxes as the platform will eventually provide the capacity to create them by combining unitary processing elements. However the unitary elements themselves are black boxes: pieces of code packaged and integrated on the platform that cannot be explored.

The development environment goal is to give the users the capacity to work at an even lower level. It allows users to develop their own code and execute it on the platform with full access to all the platform resources.

The idea here is not to replace the processing units currently running on the platform by custom code, but instead to provide the users with an environment in which they can prototype and develop their processing service before submitting them as fully packaged units.

1.2 Functional needs

1.2.1 Available data

Code developed in the development environment should have full access to the platform resources:

- Data from the user workspace and the common data folder
- Data from the catalogue

1.2.2 Supported languages

The development environment should not be strictly constrained to a unique development language. Current processing services are implemented in R and in Python, so at the minimum the environment should support these languages.

1.2.3 Data visualization

The development environment should be able to provide its users with proper feedback.

It should be possible to visualize images.

Specific means to handle geographic data is really interesting in the EO4wildlife context, in particular being able to put georeferenced data on top of a geographic map layer.

1.2.4 Security

The development environment allows users to run any code on the platform. However users should not be able to overload the platform or to corrupt his integrity. The access to critical and private data on the platform should be properly restricted and boundaries have to be established to limit the amount of resources available for a user.

2 Notebook technology

2.1 General architecture concepts

A Notebook is a remote development environment available for the users from their browser that allows writing code and executing it on a server machine.

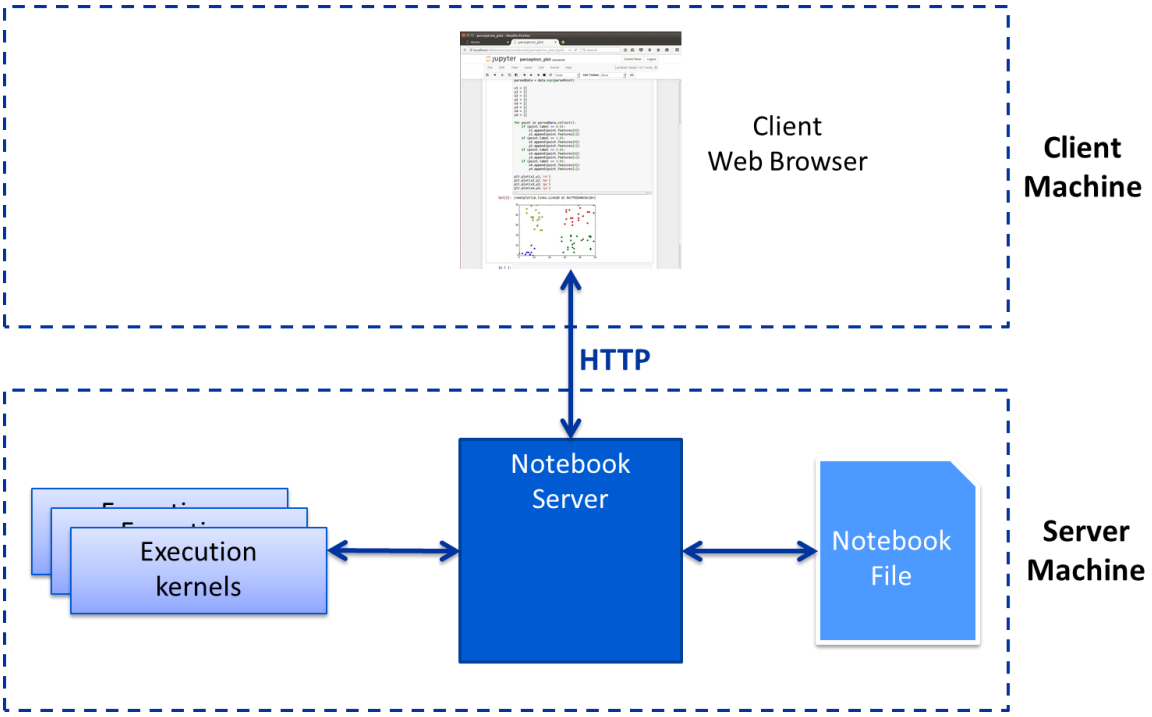

Figure 1: Notebook environment architecture

Users write Notebook *documents:* HTML files containing code, textual information and additional metadata (execution language, version…). Notebook writers can mix in their documents presentation information as they would do in a wiki media, executable code and visualization of the results of this code.

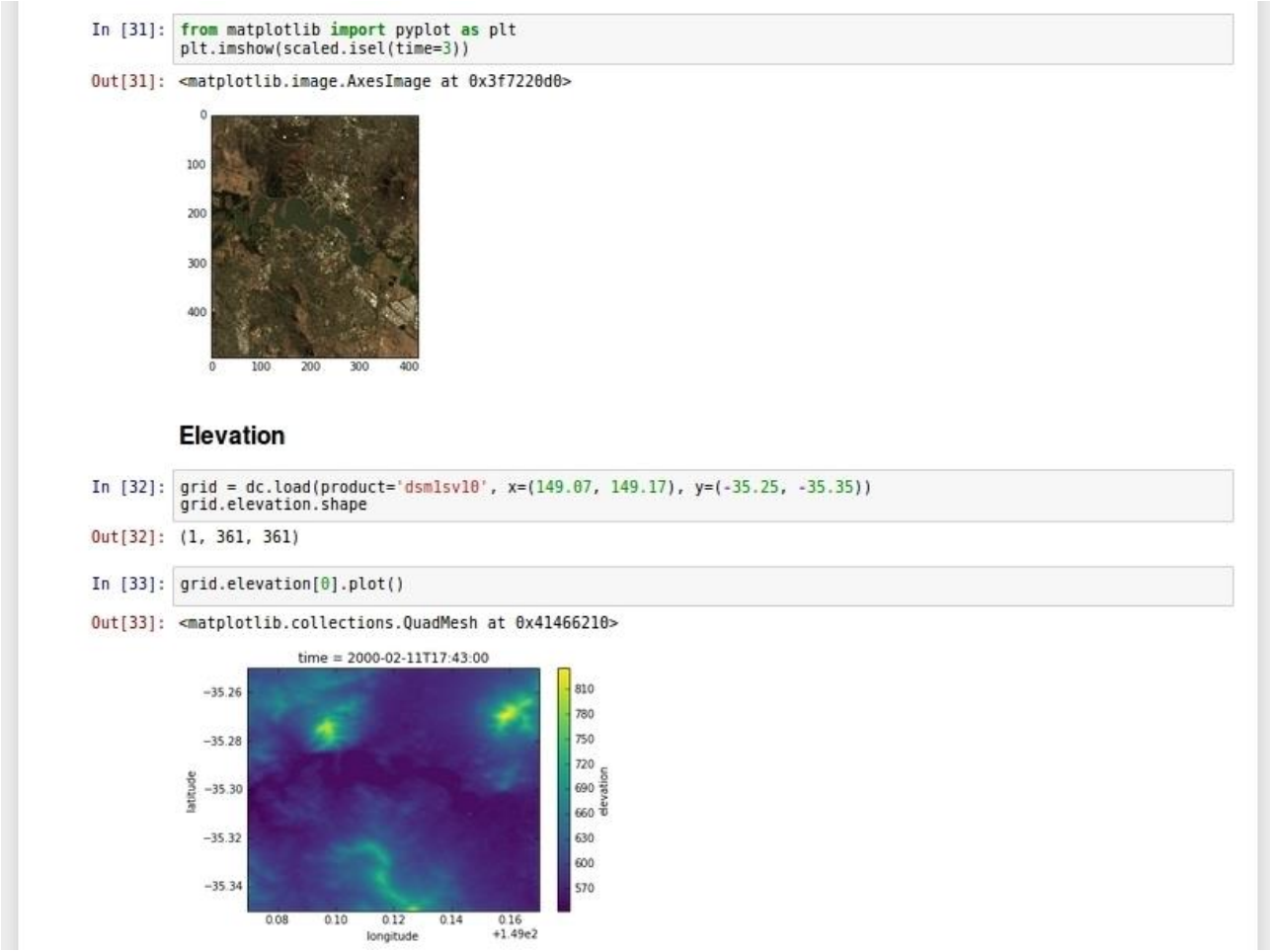

Figure 2: Notebook mixing rich text, code and result visualization

The Notebook documents files are stored on the Notebook server machine, but they can also be downloaded by the user to be kept locally or shared with other users.

There are several obvious benefits to such an approach. First, users do not need to install anything in their local machine. They can run Python programmes without any local Python installation for instance.

Moreover, programmes executed in the Notebook environment can access data located on the Notebook server. There is no need to download locally all the data. When dealing with big amounts of data, it is possible to execute code that will filter, select or reduce them, and transfer only small amounts to the user machine for display.

2.2 Jupyter Notebook

Jupyter[1] is an open-source Notebook environment born from the refactoring of the IPython project to become usable with any language. Initially designed to execute Python code, it supports now more than 40 different languages thanks to a modular system of configurable kernels. The Jupyter project is very active and new features are regularly proposed by the community.

[1] http://jupyter.org

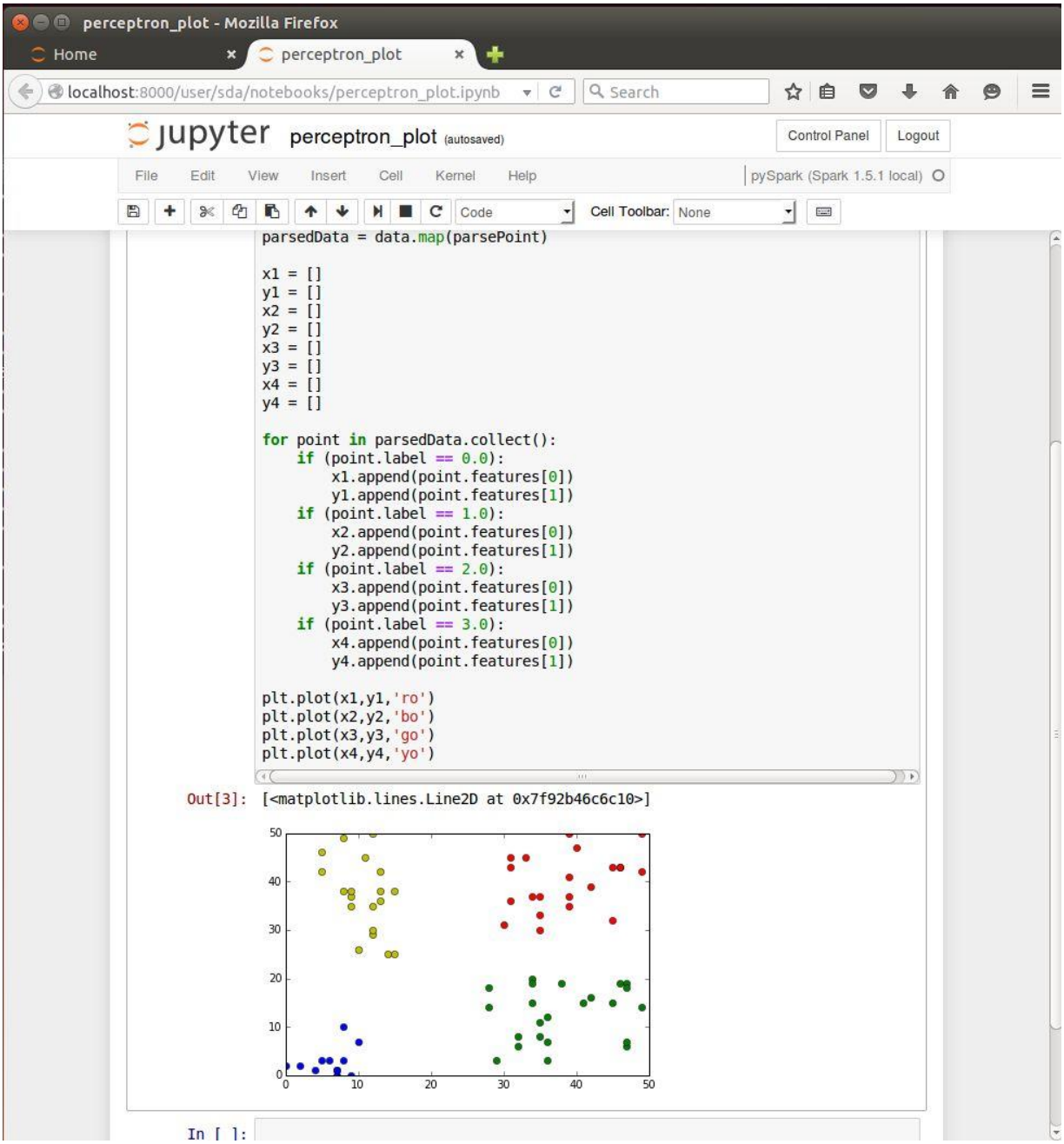

Figure 3: Jupyter Notebook screenshot

2.2.1 Notebook documents

A Jupyter Notebook document is a sequence of *cells*. A cell can contain code or rich text information encoded with markdown syntax. Each cell can be executed independently. An execution context is managed by the server to keep in memory all variables defined in a previously executed cell.

Some demonstration documents are made available by the Jupyter project team at https://try.jupyter.org.

2.2.2 Code execution kernel

The core Notebook server is responsible for routing the client request and managing the Notebook documents. The actual execution of the code is performed by components installed in addition to the core

server. The basic Jupyter installation provides a Python 2.7 kernel. In order to execute code from any other language, additional kernel should be installed and configured.

R and Python kernel are required as current processing services are implemented with these languages.

When installing a language specific kernel, it is possible to install additional libraries so that the users can use them natively in their Notebook.

Additional libraries pointed out by the partners are listed in the following table.

Name	Coding Language	Language Version	Additional libraries
Python	Python	2.7	GDAL, netCDF4, NumPy, SciPy, pandas
IRKernel	R	3.3	mapdata, maptools, adehabitatLT, adehabitatHR , sp, rgdal, rgeos, raster, geosphere, maps, rts, zoo

Table 1: Jupyter main kernels

The following table lists (not fully exhaustively) all the other available execution kernels for Jupyter Notebook. Integrating a kernel to the Jupyter core module, while not being a huge work, still require time and use some resources on the platform, so useless kernels should not be installed.

At the moment, only the need for a Spark kernel is pointed out by the partners. It will be installed in addition to the Python and the R kernels on the platform.

Name	Coding Language	To be installed in EO4wildlife platform?	Additional libraries
ICSharp	C#	No	
IErlang	Erlang	No	
IForth	Forth	No	
IFSharp	F#	No	
IGo	Go	No	
IHaskell	Haskell	No	
IJavascript	Javascript	No	
IJulia	Julia	No	
IOCaml	Ocaml	No	
IOctave	Octave	No	
IPerl	Perl	No	
IRuby	Ruby	No	
IScala	Scala	No	
IScilab	Scilab	No	
ISpark	Scala with Spark Integration	No	
jupyter-c-kernel	C	No	

Name	Coding Language	To be installed in EO4wildlife platform?	Additional libraries
jupyter-kernel-jsr223	Java	No	
jupyter-nodejs	node.js	No	
Jupyter-PHP	PHP	No	
MATLAB Kernel	Matlab	No	
Prolog	Prolog	No	
Pyspark	Python with Spark integration	Yes	GDAL, netCDF4, NumPy, SciPy, pandas
SageMath	SageMath	No	
SPARQL	SPARQL primitives in Python	No	

Table 2: Jupyter additional kernels

3 Multi-user aspect

3.1 JupyterHub

3.1.1 Functional features

The JupyterHub[2] project is a Jupyter sub-project adding a multi-user layer to the Jupyter core server with an authentication system that allows using it efficiently in a business environment.

3.1.2 Architecture

JupyterHub is composed of:

- A Hub component, managing the multi-users connections, with their specific access rights and password.
- A frontend proxy, routing request to the Hub and creating Jupyter server instances specific for each users. The proxy is then in charge of transferring the incoming requests to the corresponding Jupyter server.
- Several standard Jupyter servers.

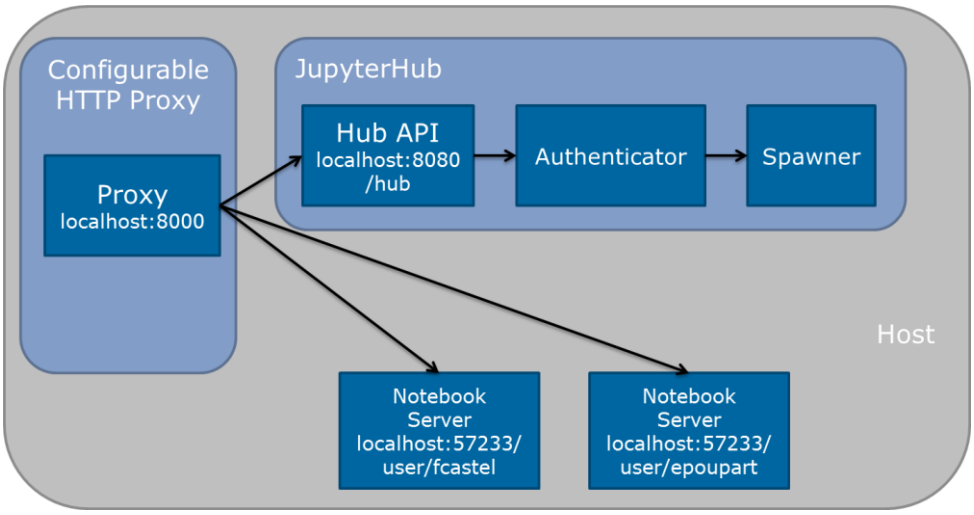

Figure 4: JupyterHub global architecture

3.1.3 Jupyter server spawning

The action of creating a Jupyter server specific for a user is called *spawning*. In the Hub, a specific module, the Spawner, handles this action. Several implementations exist for this module according to the strategy chosen to spawn the Notebook servers: local or remote spawning based on cluster management software (Torque, PBS...), on Docker, on Kubernetes…

The spawner implementation installed on the EO4wildlife platform is based on Kubernetes[3] (for obvious reason, as the platform itself is based on Kubernetes).

[2] https://jupyterhub.readthedocs.io/en/latest/
[3] https://github.com/jupyterhub/kubespawner

3.1.4 User authentication

A specific module handles the user management on the hub. As for the spawning, several implementation of this module exist: based on the UNIX users, on the OAuth protocol or on LDAP.

The user base in the platform is managed in an LDAP server, so we opt here for the LDAP authenticator[4] to directly connect the JupyterHub user base to the global platform user base.

3.1.5 Workspace management

A specific Jupyter instance is created every time a user launches the Notebook feature in the platform. Thus, by default the workspace of this instance is empty. Different solutions exist to enable a persistent workspace.

On the platform, instance spawning is based on Kubernetes, and thus by extension on Docker containerization. Each newly created instance is a new Docker container running on the Kubernetes platform. Kubernetes/Docker allows configuring *volumes*, i.e. folders shared between containers, or between a container and the host system. The strategy to make the user workspace persistent in JupyterHub is to create a volume mapping the JupyterHub workspace folder with the user workspace folder currently used on the platform. As an additional benefit, users can directly access in Jupyter data stored in their workspace and in the common_data folder, and can also access and download from the EO4wildlife platform the Notebook document they created.

[4] https://github.com/yuvipanda/ldapauthenticator

4 Deployment on the platform

4.1 Front-end integration

Jupyter provides a web interface to open, edit and run Notebook documents. This interface is directly integrated to the EO4wildlife HMI (technically, as an Iframe component). A configuration of Jupyter has to be performed to allow embedding the Jupyter interface on the EO4wildlife HMI (See **7.1.1 HMI embedding)**.

4.2 User management

Jupyter is directly connected to the EO4wildlife LDAP user authentication base (see **3.1.4**). There is no need to manage any additional user base specific to Jupyter.

4.3 Security

4.3.1 Public Jupyter server

By default, a Jupyter server is executed locally and is only reachable locally (on the "localhost:8888" address). The server can be configured to be reachable from the outside through a public network interface, but security precautions have to be taken in this case.

First, Jupyter should be configured to be secured with a user/password access. This feature is natively present in JupyterHub so there is no additional work for the EO4wildlife configuration here.

Then, the SSL protocol should be used to encrypt the password between the client machine and the server. This configuration is done in JupyterHub.

Finally, the default configuration of Jupyter must be adapted to access outside connections (see **7.1.2 Running a public Jupyter server**)

4.3.2 Jupyter security model

The potential for malicious people to attempt to exploit the Notebook for nefarious purposes always exists. Jupyter provides a security model to prevent execution of untrusted code without explicit user input. The responsibility of handling the executed code so that the hosting platform remains stable and secured is then fully in charge of the hosting platform administrator.

The whole point of a Notebook is arbitrary code execution. There is no desire to limit what can be done with a Notebook, which would negatively impact its utility. Unlike other programs, a Jupyter notebook document includes output. Unlike other documents, that output exists in a context that can execute code (via Javascript).

The security problem that needs to be solved is that no code should be executed just because a user has opened a notebook that he/she did not write. Like any other program, once a user decides to execute code in a notebook, it is considered trusted, and this code should be allowed to do anything in its limited scope.

The Jupyter basic security model is the following:

- Untrusted Javascript is never executed
- HTML and Javascript in Markdown cells are never trusted
- Outputs generated by the user are trusted

- Any other HTML or Javascript (in Markdown cells, output generated by others) is never trusted
- The central question of trust is “Did the current user do this?”

4.3.3 Security and resource constraint in the EO4wildlife platform

As seen above, the idea is not to constrain the code that the user can write and execute. As beside philosophic considerations, there is no realistic means to automatically sort malicious code from normal code. However, a user written code should not jeopardize the whole platform stability. For instance, if a user executes a very resource consuming infinite loop, the platform should remain usable for other users.

To handle this issue, resources usable by a Notebook execution have to be restricted, i.e. a maximum CPU and memory usage threshold is set.

This feature is quite easy to implement with the JupyterHub configuration installed on the platform based on Kubernetes. When creating a Jupyter user specific container, an additional configuration is set to limit the resources that the container can use.

Moreover, Jupyter servers are very well isolated. As a specific server is created dynamically by JupyterHub for each user, if a problem occurs in the Jupyter container, it might crash and the user might restart his Notebook server, but there will be no impact on the containers of the other users.

5 Additional features

5.1 API to the data catalogue

The EO4wildlife platform is designed to provide an easy access to a wide range of georeferenced products through a data catalogue. Processing on the platform is based the OGC interface standard WCS to retrieve the data.

Notebook document written on the Platform will be able to access this data catalogue using the same WCS entry point.

API implementing the WCS interface standard (and other OGC standards) exists in Python (OWSLib[5]) or in Java (GeoAPI[6]). In R a proper API exists to run WPS process (geoknife[7]) but it does not cover all the other OGC standards such as WCS. However WCS services can still be accessed with low level interaction means (such as simple curl command).

Without a proper API, a workaround is still possible. Notebook document can access data on the user workspaces, thus as soon as extraction from the catalogue to the user workspaces is available it is possible to work with catalogue data on notebook documents.

5.2 API to the service catalogue

Processing services are launched from the Workflow panel. When trigging and execution from the frontend, a backend web service is called to launch the processing code execution on the execution engine.

Launching processing services directly from a notebook can be interesting in order to test workflows. An API can be developed to be able to call the backend processing execution web services from the notebook in the same way as they are called from the workflow panel on the frontend.

5.3 Spark integration

Spark is a development framework for intensive and distributed computing. Among the kernels available in Jupyter, some of them allow a native Spark integration.

5.3.1 Spark kernel

Spark integration in Jupyter makes available to Notebook writers a native support for Spark application development. No configuration for them is needed to create a Spark context and run their code in the underlying installed Spark cluster.

Kernels integrating Spark are available in Python and in Scala (see **1.2.2 Supported languages**).

5.3.2 Spark cluster deployment

Spark can be run in several modes:

- In local mode on a single machine. This can be useful to perform pseudo distribution on a machine with a multi core processor.
- In a real distributed mode on a multi-machine cluster.

[5] https://geopython.github.io/OWSLib/
[6] http://www.opengeospatial.org/standards/geoapi
[7] https://cran.r-project.org/web/packages/geoknife/vignettes/geoknife.html

Obviously, the main benefits of Spark are obtained when using a multi-machine cluster. Very resource consuming tasks can thus be distributed on several worker nodes to improve the global processing time.

A Spark cluster can eventually be installed on the EO4wildlife platform if the need to run such heavy processing is identified. In this case, the cluster should be usable through the Notebook feature but also from the processing services.

6 Conclusion

Integrating a Notebook server to the platform is a feature that allows free and low level interaction with the EO4wildlife platform.

Users obtain the possibility to access data on the platform through an interactive interface providing them with a development environment adapted for processing prototyping and development. Service providers can test their processing services on the platform before releasing them, and end-users can experiment custom workflows.

A multi-user Notebook environment can be installed on the EO4wildlife platform based on already used technology (Kubernetes, Docker, and LDAP).

7 Annex

7.1 Jupyter Configuration

7.1.1 HMI embedding

To embed the notebook on a website (e.g. in an Iframe) it is necessary to override the Content-Security-Policy to allow embedding. Assuming the website is at *https://mywebsite.example.com*, notebook can be embedded with the following configuration setting in `jupyter_notebook_config.py`:

```
c.NotebookApp.tornado_settings = {
    'headers': {
        'Content-Security-Policy': "frame-ancestors
'https://mywebsite.example.com' 'self' "
    }
}
```

Listing 1: Jupyter configuration, HMI embedding

When embedding the notebook in a website using an iframe, consider putting the notebook in single-tab mode. Since the notebook opens some links in new tabs by default, single-tab mode keeps the notebook from opening additional tabs. Adding the following to `~/.jupyter/custom/custom.js` will enable single-tab mode:

```
define(['base/js/namespace'], function(Jupyter){
    Jupyter._target = '_self';
});
```

Listing 2: Jupyter configuration, single-tab mode

7.1.2 Running a public Jupyter server

To access the notebook server remotely via a web browser, a public notebook server has to be run.

If one does not already exist, a configuration file has to be created for the notebook using the following command line:

```
$ jupyter notebook --generate-config
```

In the `~/.jupyter` directory, the notebook configuration file, `jupyter_notebook_config.py`, has to be edited. By default, the notebook config file has all fields commented out. The minimum set of configuration options that should be uncommented and edited in `jupyter_notebook_config.py` is the following:

```
# Set options for certfile, ip, password, and toggle off
# browser auto-opening
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/mycert.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/mykey.key'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

Listing 3: Jupyter configuration, public mode