Czech technical university in Prague
Faculty of Civil engineering

Master's thesis

Prague 2018

Bc. Adam Laža

Czech technical university in Prague
Faculty of Civil engineering
Study programme Geodesy and cartography
Geomatics

Master's thesis

Process isolation in PyWPS framework

Izolace procesů ve frameworku PyWPS

Supervisor: Ing. Martin Landa, Ph.D.
Department of geomatics

Prague 2018 Bc. Adam Laža

Abstract

Upravit abstract, aby odpovidal skutec strukture

This master thesis is dedicated to isolation of PyWPS processes as one of the OGC WPS implementation. OGC WPS is Web Processing Service Standard defined by Open Geospatial Consorcium. The practical part contains an introductory research where various solutions how to reach the process isolation are considered and described. Based on the research the Docker technology has been chosen for the implementation of process isolation. In the theoretical part Docker technology is described as well as the OGC WPS standard and its PyWPS implementation written in Python.

Keywords: OGC WPS, PyWPS, Docker container, Python, process izolation, Web Processing Service.

Abstrakt

Překlad WPS - Webová Procesingová??? Služba, OGC??

Tato diplomová práce se věnuje možnostem izolace procesů v rámci frameworku PyWPS jako jedné z implementací OGC WPS. Webová Procesingová Služba je standard vydaný a dále rozšiřovaný Open Geospatial Consorciem. Praktická část obsahuje úvodní rešerši, ve které jsou popsány různé možnosti, jak izolace jednotlivých procesů dosáhnout. Na základě rešerše byla pro implementaci vybraná technologie Docker. V teoretické části je popsána jak technologie Docker, tak OGC WPS standard a jeho implementace PyWPS napsaná v jazyce Python.

Klíčová slova: OGC WPS, PyWPS, Docker kontejner, Python, izolace procesu, Webová Procesingová Služba.

Declaration of authorship I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged. Formulations and ideas taken from other sources are cited as such

In Prague ....

(author sign)

Acknowledgement

Podekovani

Contents

Introduction

There are data all around us. As the society is becoming more and more digitalized the amount of the data is getting bigger and bigger. A lot of enterprises, institutions and organizations realize that these data hide a huge potential they can profit from. However the data themself in their raw form are not usually sufficient to make an conclusion from them. More often the data need to be processed and used as an inputs data for some kind of analises. With the increasing number of gathered data a manual processing is almost unconceivable. Data are processed in an automatized way.

Therefore, in order to be able to process the data independently of the type of acquisition, format or platform, it is necessary to define standards. Regarding spatial data, these standards are made by the Open Geospatial Consortium. Besides quite famous and used standards as WMS and WFS there also exists the WPS standard. The WPS standard defines an interface that facilitates the publishing of geospatial processes. It also provides rules how inputs and outputs are handled. The WPS is only a standard and there are several implementations. This work is primarily focused on the *PyWPS* framework.

The main topic of this thesis is process isolation. Process is just some geospatial operation which has its defined inputs and outputs and which is deployed on the server. Server is able to execute multiple processes at the same time. This thesis deals with the isolation of individual processes especially for security and performance reasons. With every process fully isolated so they cannot interact with each other the higher security level is assured.

The thesis is composed of several parts. The introductory research discusses the current state of the PyWPS and the other projects that implement the WPS standard, namely *ZOO-Project* and *52°North*. Then the introducing research offers possible soloutions to achieve process isolation. Various projects and technologies are described and finally the Docker has been selected as the technology we try to implement in the practical part. Docker has been selected as one of the most used technology for containerization. It puts every process into seperate container so the isolation is ensured. Moreover Docker provides mechanism to pause, stop and start a container so it looks like a possible solution for the future WPS 2.0.0 standard implementation which requires this funcionality. Using Docker it also opens new posibilities like being able to deploy running job to cloud.

The technological background is covered in the second part. There is the WPS standard described, especially its operations - *GetCapabilities*, *DesribeProcess* and *Execute* - and inputs and outputs strucutres. There are also *PyPWS* and *Docker* described.

Last part consists of the implementation description.

Doplnit uvod o implementaci

I have choosen this topic to get in touch with another OGC standard. I also appreciate I can dive more into Docker technology as it is a leader in containerization at the world.

Part I

Introductory research

1 Current state

2 Process isolation in PyWPS

2.1 Asynchonous requests

Right now in PyWPS 4.0 version a PyWPS server instance is able to run multiple concurrent processes in parallel. The server is configured for maximal amounts of concurrently running processes at the same time and for maximal amount of waiting processes in a queue, to later start their execution once new slots are available. If the new Execute request is received and the maximal amount is exceeded, the request is rejected and user is informed in response (see Lst. 1).

Listing 1: Resource exceeded exception

```
<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1"
    version="1.0.0">
    <ows:Exception exceptionCode="ServerBusy">
        <ows:ExceptionText>
            Maximum number of parallel running processes reached.
                Please try later.
        </ows:ExceptionText>
    </ows:Exception>
</ows:ExceptionReport>
```

To facilitate the management of concurrent processes, process metadata are stored into a local database. This database is used for logging , saving waiting Execute requests in the queue and invoking them later on. This database will also enable the implementation of pausing, releasing and deleting running process. These features will allow PyWPS to comply with WPS version 2.0.0.

2.2 Current state

At the beginning of every process execution its own temporary directory *workdir* is created. During the execution temporary files and continous outputs are stored in this folder. After successful execution final outputs are moved to *outputs* directory.

Both directories *outputs* and *workdir* are configurable and user can change path to them.

Listing 2: pywps.cfg - mode parameter

```
[processing]
mode=multiprocessing
```

Current version of PyWPS offers two solutions for running parallel processes:

- Multiprocessing
- Job Scheduler Extension[1]

If the execute request is sent asynchronously the type of process constructor is choosen depending on configuration parameter *mode* in section *processing* which is by default *multiprocessing* or can be changed to *scheduler*.

Listing 3: processing.__init__.py

```
def Process(process, wps_request, wps_response):
    """
    Factory method (looking like a class) to return the
    configured processing class.

    :return: instance of :class:`pywps.processing.Processing`
    """
    mode = config.get_config_value("processing", "mode")
    LOGGER.info("Processing mode: %s", mode)
    if mode == SCHEDULER:
        process = Scheduler(process, wps_request, wps_response)
    else:
        process = MultiProcessing(process, wps_request,
            wps_response)
    return process
```

[1] Job Scheduler Extension is currently only in develop branch of PyWPS.

Multiprocessing By default for processes running in the background, the Python multiprocessing module is used – this makes it possible to use PyWPS on the Windows operating system too.

Job Scheduler Extension PyWPS scheduler extension offers possibilities to execute asynchronous processes out of the WPS server machine. This extension enables to delegate execution of processes to a scheduler system like *Slurm*, *Grid Engine* and *TORQUE* from Adaptive Computing. These schedular systems are usually located at *High Performance Compute (HPC)* centers.

Figure 1: Grid Engine

Figure 2: Slurm

Figure 3: TORQUE

The PyWPS scheduler extension uses the Python *dill* library to dump and load the processing job to/from filesystem. The batch script executed on the scheduler system calls the PyWPS *joblauncher* script with the dumped job status and executes the job (no WPS service running on scheduler). The job status is updated on the filesystem. Both the PyWPS service and the joblauncher script use the same PyWPS configuration. The scheduler assumes that the PyWPS server has a shared filesystem with the scheduler system so that XML status documents and WPS outputs can be found at the same file location. The interaction diagram how the communication between PyWPS and the scheduler works is displayed at Fig. 4.

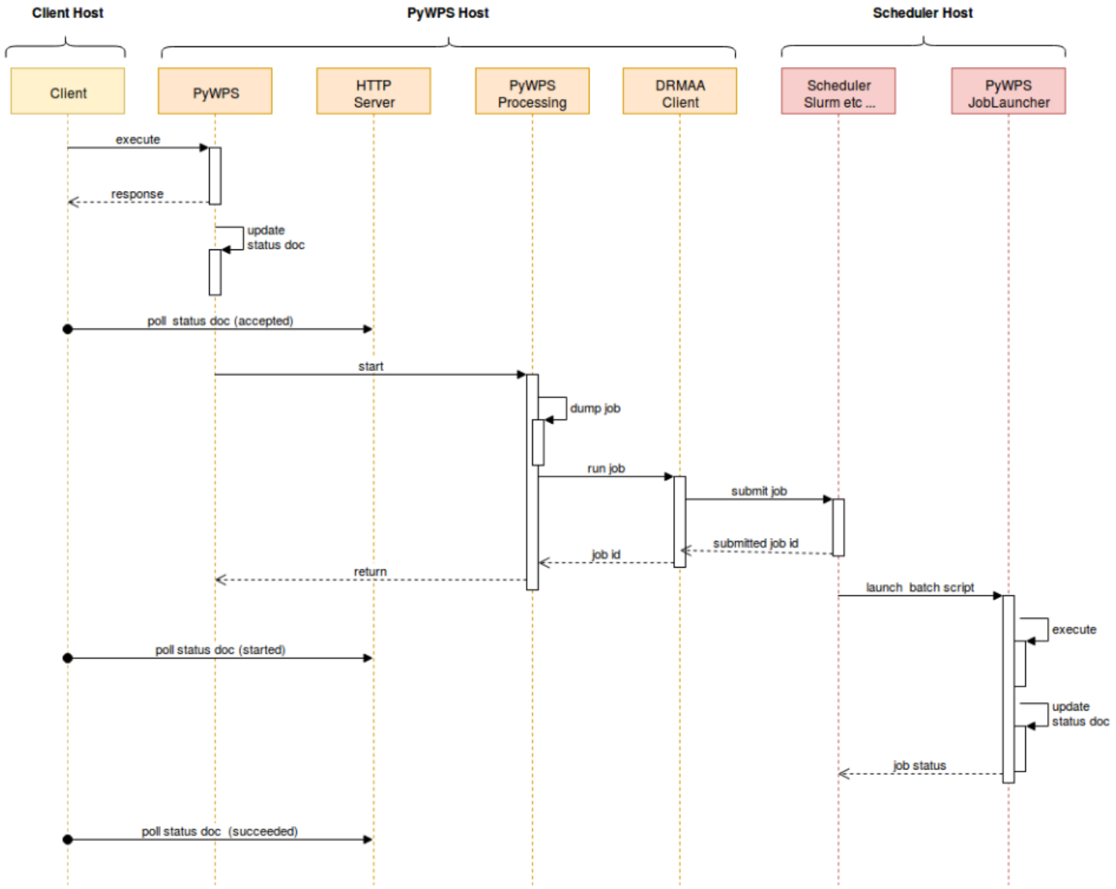

Figure 4: Communication between PyWPS and scheduler, source: [10]

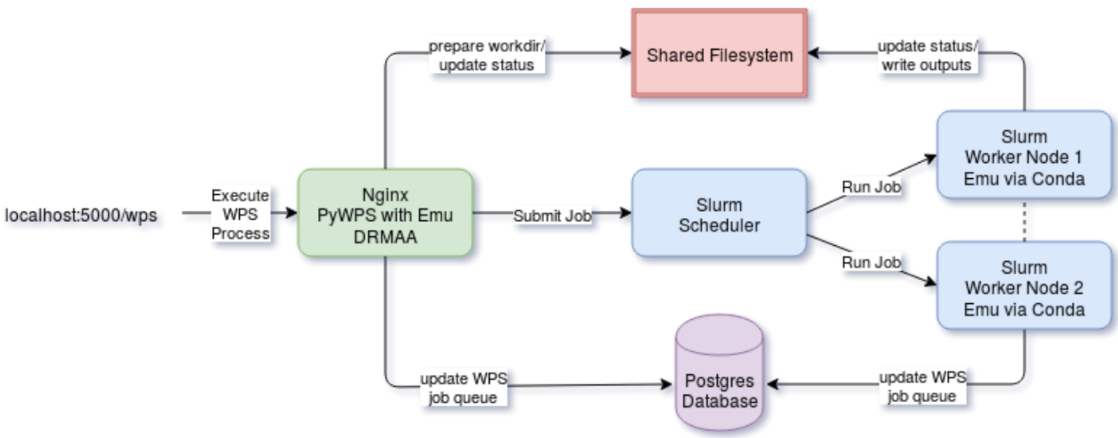

Figure 5: Example of PyWPS scheduler extension usage with Slurm, source: [10]

2.3 Possible solutions

In previous section there were described two mechanisms for running parallel processes. Nevertheless in case of Python module *Multiprocessing* the processes are not really isolated. They run concurrently but they can share resources and there are even methods like *Pipe()* that enables comunication between processes.

On the otherhand *Job Scheduler Extension* is dependent on *dill* library as well as on some external scheduler systems like *Slurm*, *Grid Engine* or *TORQUE*.

In this section there are described some other solutions. Some of them were suggested by PyPWS developers with encouragment to make a feasible study. Some of them were discovered during research on the internet forums like StackOverflow, some of them were referenced in documentation of other projects. During the research two requirements were considered.

- Solution provides mechanism for full isolation. This is a must-have requirements.
- Solution provides mechanism for start/pause/stop process execution. This is a nice-to-have requirements as this funcionality will be required to comply WPS 2.0.0 standard.

Finnaly these solutions were considered:

- Celery
- Docker
- psutil
- SandboxedPython
- VM

2.3.1 Celery

Celery is a task queue system written in Python. It helps distribute work across threads and even machines. Basic term is *task*. Task is an unit of work and it is an input into the task queue. The task queue is constantly monitored for new work to perform.

To communicate between client and workers Celery uses a *broker*. The communication is via messages. To initiate a task the client adds a message to the queue and the broker then delivers the message to a worker. Multiple workers and brokers can be added so there is assured high availability and horizontal scaling.

Celery provide worker remote control client in class *celery.app.control.Control(app=None)*. The class offers these functions:

- **revoke** - Tell all (or specific) workers to revoke a task by id. If a task is revoked, the workers will ignore the task and not execute it after all.
- **shutdown** - Shutdown worker(s).
- **terminate** - Tell all (or specific) workers to terminate a task by id.

2.3.2 Docker

Docker is one of the most used technology regarding containerization. This technology is described in depth in later chapter.

2.3.3 psutil

psutil is Python library for process and system management. It handles system monitoring, limiting process resources and the management of running processes. Its implementation is based on UNIX command line tools. psutil offers functions applied to these sections:

- CPU - functions for CPU statistics such as CPU utilization percentage, frequency and others.
- Memory - functions for system memory usage and swap memory statistics.

- Disks - functions for disk statistics such as disk usage or disk IO operations counter.
- Network - functions for network IO operations or network connection statistics.
- Sensors - functions for statistics about fans, battery or hardware temperature.
- Others - functions for boot time and users statistics.
- Processes - functions will be described in detail later.

Processes - Class *psutil.Process(pid=None)* represents an OS process with given pid. The class is bound with a process via its PID. The *Process* class offers these methods for starting/pausing:

- suspend() - Method suspends process using SIGSTOP signal.
- resume() - Method resumes process using SIGCONT signal.
- terminate() - Method terminates process using SIGTERM signal.
- kill() - Method kills process using SIGKILL signal.

2.3.4 Sandboxed Python

The general goal of sandbox is to run applications securely inside isolated environment they cannot escape from and so affect other parts of the system. Developers use them to run untrusted code inside. It is quite difficult to develope fully sandboxed solution due to Python complexity. The basic problem is that Python introspection allows several ways to excape out of the sandbox. True security requires an overall design with many security considerations included. Some of the projects that can run Python code in sandbox are:

- PyPy
- Jython

PyPy PyPy is Python interpreter written in RPython that implements full Python language and very closely emulates the behavior of CPython. PyPy offers fully portable sandboxing feature similar to OS-level sandboxing (e. g. SECCOMP). It is not sandboxing at the Python language level so it does not put any restriction on any Python funcionality.

Untrusted Python code that is intended to be sandboxed is launched in a subprocess, that is a special sandboxed version of PyPy. All its inputs/outputs are not directly performed but are serialized to a stdin/stdout pipe. The outer process reads the pipe and afterwards decides which commands are allowed.

Jython Jython is Python language interpreter for Java. Java offers strong sandboxing mechanisms. The security facility in Java that supports sandboxing is the *java.lang.SecurityManager*. By default, Java runs without a SecurityManager.

pysandbox A prove, that it is very difficult to develop some kind of sandbox with all security holes considered, could be a project call *pysandbox*[2]. After working on it for 3 years, during which the project was used on various production servers by other developers, its author declared that the project is broken by design. In his post to the python-dev mailing list [11] the author explained that with every vulnerability founded it became more difficult to actually write a real code:

"To protect the untrusted namespace, pysandbox installs a lot of different protections. Because of all these protections, it becomes hard to write Python code. Basic features like "del dict[key]" are denied. Passing an object to a sandbox is not possible to sandbox, pysandbox is unable to proxify arbitary objects.

*For something more complex than evaluating "1+(2*3)", pysandbox cannot be used in practice, because of all these protections. Individual protections cannot be disabled, all protections are required to get a secure sandbox."*

2.3.5 VM

[2]https://github.com/vstinner/pysandbox

Part II

Technological background

3 Web Processing Service

3.1 History

First mention of the Web Processing Service was in October 2004. Back then it was named Geoprocessing Service [1]. The specification was first implemented as a prototype in 2004 by Agriculture and Agri-Food Canada (AAFC). In its further development during a Geoprocessing Services Interoperability Experiment [2] the name was changed to "Web Processing Service" to avoid the acronym GPS, since this would have caused confusion with the conventional use of this acronym for Global Positioning System [4]. The first version of WPS was released in September 2005 [3]. The experiment demonstrated that various clients could easily access and bind to services which were set up according the WPS Implementation specification.

Currently two major versions of WPS Standard exist. The WPS version 1.0.0 is currently used mostly. If not explicitly said this thesis is dedicated to the version 1.0.0. The WPS version 2.0.0 was released in 2015 [5].

3.2 OGC

D

oplnit kratky odstavec o OGC

3.3 Web Processing Service

The OGC Web -Processing Service (WPS) Interface Standard defines a standardized interface that facilitates the publishing of geospatil processes. Also provides rules how to standardize requests and responses for geospatial processing services.

Process means any operation on spatial data from simple ones as maps overlay or buffering to highly complex as complicated global models. Any kind of GIS funcionality can be offered to clients across network with correctly configured WPS.

Publishing means creating human-readable metadata that allow user to discover and use service as well as making available machine-readable binding information.

Data can be both vector or raster data and can be delivered across the network or be available at the server.

The interface does not specify any specific processes that can be implemented by a WPS nor any specific data inputs or outputs. instead it specifies a generic mechanisms to describe any geospatial process and data required and produced by the process. The interface does not only provide mechanisms for calculation but also to identify required data, initiate the calculation and manage output data so clients can access it.

Web Processing Service as one of the OGC web services scpecifies three types of requests which can be requested by a client and performed by a WPS server. The implementation of these three requests is mandatory by by all servers:

- GetCapabilities
- DescribeProcess
- Execute

GetCapabilities - The request returns to client a Capabilities document that describes the abilities of the specific server implementation. It also returns the name and abstract of each of the processes that can be run on a WPS instance.

DescribeProcess - The request returns details about the processes offered by a WPS instance. Describes required inputs and produces outputs and their allowable formats.

Execute - The request allows a client to run a specified process with provided parameters and returns produced outputs.

These operations are very similar to other OGC Web Services such as WMS, WFS, and WCS. Common interface aspects are defined in the OpenGIS ® Web Services Common Implementation Specification [6]. As seen at class diagram at Fig. 6 the WPS interface class inherits the GetCapabilities operation from OGCWebService interface class. The operations Execute and DescribeProcess are specific for the WPS. The WPS operations are based on GET and POST requests.

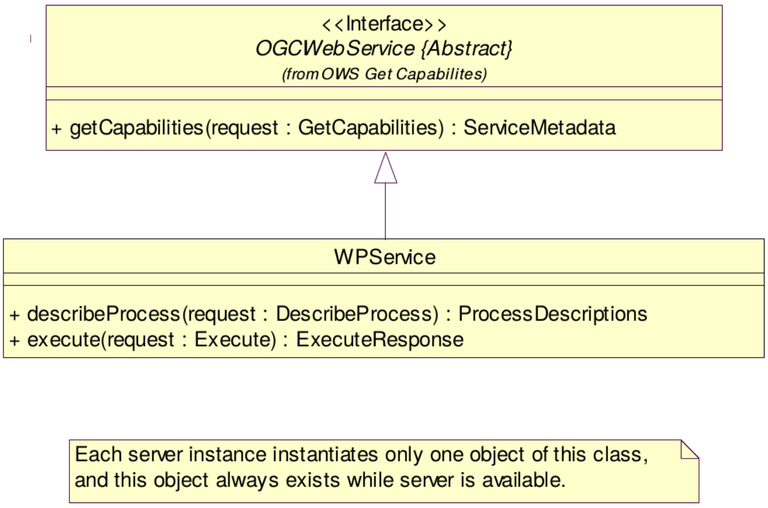

Figure 6: WPS interface UML description, source: [4]

Operation	Request encoding	
	Mandatory	Optional
GetCapabilities	KVP	XML
DescribeProcess	KVP	XML
Execute	XML	KVP

Table 1: Operations request encoding

The GetCapabilities and DescribeProcess shall use HTTP GET with KVP encoding and Execute operation shall use HTTP POST with XML encoding. Summarized in Table 1.

3.3.1 GetCapabilities

The GetCapabilities operation is mandatory. The operation allows clients to retrieve capabilities document (metadata) from a server. The response XML document contains service metadata about server and all implemented processes description.

AcceptVersion vs version, AcceptFormats vs format

GetCapabilities request

Name	Optionality and use	Definition and format
service=WPS	Mandatory	Service type identifier text
request=GetCapabilities	Mandatory	Operation name text
AcceptVersion=1.0.0	Optional	Specification version
Sections=All	Optional	Comma-separated unordered list of sections
updateSequence=XXX	Optional	Service metadata document version
AcceptFormats=text/xml	Optional	Comma-separated prioritized sequence of response formats

Table 2: GetCapabilities operation request URL parameters, source: [6]

Request parameters

- *service* - Mandatory parameter, WPS is only possible value.
- *request* - Mandatory parameter, GetCapabilities is only possible value.
- *version* - Optional parameter, version number. Three non-negative integers separated by decimal point. Servers and their clients should support at least one defined version.
- *sections* - Optional parameter that contains a list of section names. Possible values are: *ServiceIdentification, ServiceProvider, OperationsMetadata, Contents, All.*
- *updateSequence* - Optional parameter for maintaining the consistency of a client cache of the contents of a service metadata document. The parameter value can be an integer, a timestamp, or any other number or string.
- *updateSequence* - Optional parameter for maintaining the consistency of a client cache of the contents of a service metadata document. The parameter value can be an integer, a timestamp, or any other number or string.
- *format* - Optional parameter that defines response format.

The GetCapabilities operation can be requested with parameters from table 2. A corresponding request URL looks like: `http://localhost:5000/wps?service=WPS&request=GetCapabilities&AcceptVersion=1.0.0&Section=ServiceIdentification,OperationsMetadata&updateSequence=XXX&AcceptFormats=text/xml`

GetCapabilities response

Normal response When GetCapabilities operation requested a client retrieve service metadata document that contains sections specified in *sections* parameter. If the parameter value is *All* or is not specified all sections retrieved.

- *ServiceIdentification* - Server metadata.
- *ServiceProvider* - Server operating organization metadata.
- *OperationsMetadata* - Metadata about operations implemented by the WPS server, including URLs to request them.
- *ProcessOfferings* - List of processes with name and brief description implemented by the WPS server.

In addition to sections each GetCapabilities response should contains:

- *version* - Specification version for GetCapabilities operation.
- *updateSequence* - Server metadata document version, value is increased whenever any change is made in complete service metadata document.

GetCapabilities exceptions In case that WPS server encounters an error a client retrieve an exception report message with one of there exception code:

- *MissingParameterValue* - GetCapabilities request does not contain a required parameter value.
- *InvalidParameterValue* - GetCapabilities request contains an invalid parameter value.

- *VersionNegotiation* - Any version from AcceptVersions parameter list does not match any version supported by the WPS server.
- *InvalidUpdateSequence* - Value of updateSequence parameter is greater than current value of service metadata updateSequence number.
- *NoApplicableCode* - Other exceptions.

3.3.2 DescribeProcess

The DescribeProcess operation is mandatory. The operation allows clients to retrieve a detailed description about one or more processes implemented by a WPS server. The detailed information describe both required inputs and produced outputs and allowed format.

DescribeProcess request

Request parameters

- *service* - Mandatory parameter, WPS is only possible value.
- *request* - Mandatory parameter, DescribeProcess is only possible value.
- *version* - Mandatory parameter, version number. Three non-negative integers separated by decimal point. Servers and their clients should support at least one defined version.
- *Identifier* - Optional parameter, list of process names separated by comma. Another possible value is *all*.

The DescribeProcess operation can be requested with parameters from table 3. A corresponding request URL looks like: `http://localhost:5000/wps?request=DescribeProcess&service=WPS&identifier=all&version=1.0.0`

DescribeProcess response

Name	Optionality	Definition and format
service=WPS	Mandatory	Service type identifier text
request=DescribeProcess	Mandatory	Operation name text
version=1.0.0	Mandatory	WPS specification version
Identifier=buffer	Optional	List of one or more process identifiers, separated by commas

Table 3: DescribeProcess operation request URL parameters, source: [6]

Normal response Normal response to DescribeProcess request contains or more process descriptions for requested process identifiers in *ProcessDescriptions* structure. Each process description contains detailed information about process in *ProcessDescription* including process inputs and outputs description. Number of inputs or outputs is not limited. Three types of input or outpus exist:

Doplnit popisy dat

- *LiteralData* -
- *ComplexData* -
- *BoundingBoxData* -

Name	Optionality	Definition and format
ProcessDescription	Mandatory	Full description of process including inputs/outputs
service=WPS	Mandatory	Service type identifier text
version=1.0.0	Mandatory	Operation specification version
lang	Mandatory	Language identifier

Table 4: Parts of ProcessDescriptions data structure, source: [4]

DescribeProcess exceptions In case that WPS server encounters an error a client retrieve an exception report message with one of there exception code:

Name	Optionality	Definition and format
Identifier	Mandatory	Process identigier
Title	Mandatory	Process title
Abstract	Optional	Brief description
Metadata	Optional	Reference to more metadata about this process
Profile	Optional	Profile to which the WPS process complies
processVersion	Mandatory	Release version of process
WSDL	Optional	Location of a WSDL document that describes this process
DataInputs	Optional	List of the required and optional inputs
ProcessOutputs	Mandatory	List of the required and optional outputs
storeSupported	Optional	Complex data outputs can be stored by WPS server
statusSupported	Optional	Execute response can be returned quickly with status information

Table 5: Parts of ProcessDescription data structure, source: [4]

- *MissingParameterValue* - GetCapabilities request does not contain a required parameter value.
- *InvalidParameterValue* - GetCapabilities request contains an invalid parameter value.
- *NoApplicableCode* - Other exceptions.

3.3.3 Execute

The Execute operation is mandatory. The operation allows clients to run a specified process implemented by a server. Inputs can be included directly in the request body or be referenced as web accesible resource. The outputs are returned in XML response document, either directly embedded within the response document or stored as resource accesible by returned URL.

Ussualy the response document is returned right after the process execution is completed. However it is possible to get response document right after sending request. In this case, returned response document contains a URL link from which the final response document can be retrieved after completed process execution. Client can request execution status update to find out the amount of processing remaining if the execution is not completed. Shown at Fig. 7.

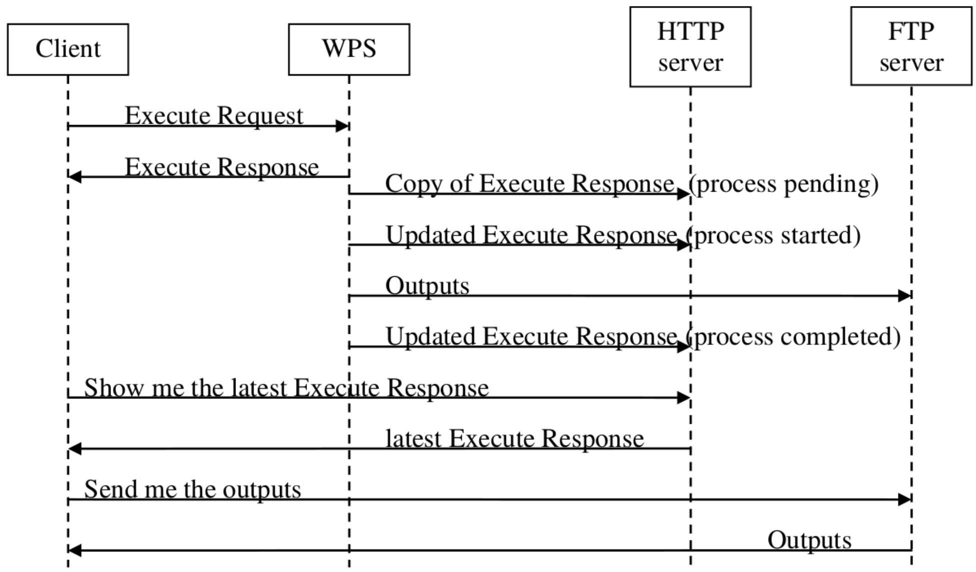

Figure 7: Activity diagram when client requests storage of results, source: [4]

Execute request

Execute response Ussualy the Execute operation response document is an XML document. Only exception is in case when a response form of *RawDataOutput* is requested, execution is successful and only one complex output is created, then directly the produced complex output is returned.

In ussual case response to Execute opetation is an ExecuteResponse XML document. The contents depend on ResponseForm request elements.

Name	Optionality	Definition and format
service	Mandatory	Service type identifier text
request	Mandatory	Operation name text
version	Mandatory	WPS specification version
Identifier	Mandatory	Process identifier
DataInputs	Optional	List of inputs provided to this process execution
ResponseForm	Optional	Response type definition
language	Optional	Language identifier

Table 6: Parts of Execute operation request, source: [4]

Name	Optionality	Definition and format
service	Mandatory	Service type identifier text
version	Mandatory	WPS specification version
language	Mandatory	Language identifier
statusLocation	Optional	Reference to location where current ExecuteResponse document is stored
serviceInstance	Mandatory	Reference to location where current ExecuteResponse document is stored
Process	Mandatory	Process description
Status	Mandatory	Execution status of the process
DataInputs	Optional	List of inputs provided to this process execution
OutputDefinitions	Optional	List of definitions of outputs desired from executing this process
ProcessOutputs	Optional	List of values of outputs from process execution

Table 7: Parts of ExecuteResponse data structure, source: [4]

3.4 WPS implementations

The OGS WPS is just interface standard that provides rules for standardizing requests and response. It also defines how clients can request the execution of defined processes and how the outputs are handled. There are several open-source projects that implement this standard across the platforms or programming languages.

- *PyWPS* - Python implementation. This thesis is dedicated to this implementation.
- *Zoo Project* - WPS implementation written in C, Python and JavaScript.
- *WPS.NET* - WPS implementation on .NET platform.
- *52°North WPS* - Java implementation.
- *deegree* - Java implementation of many OGC standards including WPS.
- *WPSint* - Java Spring implementation.

4 PyWPS

Doplnit a prepsat

PyWPS is a server side implementation of the OGC WPS standard in the Python programming language. The first version of PyWPS started in 2006 as a student project. In 2007 PyWPS 2.0.0 was released supporting WPS 0.4.0. Next year in 2008 PyWPS 3.0.0 was released with support for WPS 1.0.0. It was possible to run multiple WPS instances with one PyWPS istallation. This version had simple code structure and contained examples of processes. The newest version is PyWPS 4.0.0 from September 2017.[9].

PyWPS itself is just interface implementation. It is not an analytic tool or engine so it does not perform any kind of geospatil calculation nor provide any processing funcionality. PyWPS handles inputs, process execution and produces outputs but it is up to user (typically developer or scientist) to provide own code that is deployed on the PyWPS server instance and the server afterwards gets input data, evaluates them and calls the execute method.

Figure 8: PyWPS project logo

4.1 PyWPS 4.0

PyWPS-4 is the most current version of PyWPS. Rewritting from scratch involved this major changes:

- It is written in *Python 3* with backwards support for Python 2.7.
- It utilizes native Python bindings to existing projects (GRASS GIS).
- New popular formats like *GeoJSON*, *KML* or *TopoJSON* are reflected and their support is provided.
- PyWPS project has changed the licence from *GNU/GPL* to *MIT*.
- PyWPS 4.0 is implemented using the *Flask* framework.
- A C-based XML parser *Lxml* is used to handle XML files.
- *OWSLib* structures are used for some data types.

4.2 PyWPS-demo

Doplnit info o demu a jeho pouziti

PyWPS-demo is a small side project distributed with PyWPS. It is a simple demo instance of PyWPS server running on Flask with several demo processes.

5 Docker

Containerization is a lightweight alternative to full machine virtualization. It involves encapsulating an application into a container with its own operating environment. It helps to run containerized application on any physical machine without any worries about dependencies. The origin of containerization lies in the *LinuX Containers LXC* format. Containerization works only in Linux environments and can run only Linux applications.

Docker is not the only one technology for containerization. Other alternatives exist, it is *Kubernets*, *CoreOS rkt*, *Open Contrainer Initiative (OCI)*, *Canonical's LXD*, *Apache Mesos and Mesosphere* and many others. However Docker is a leader on the field of contanerization and with most public traction is de facto considered as a container standard. That's why the Docker was choosen for this thesis as a container technology. So from this point on any term *container* refers to Docker container.

Figure 9: Kubernetes

Figure 10: CoreOS rkt

Figure 11: Canonical's LXD

Figure 12: Apache mesos

Figure 13: Docker logo

Docker is a Linux container technology that allows package and ship applications and everything it needs to execute into a standard format, and run them on any infractructure.

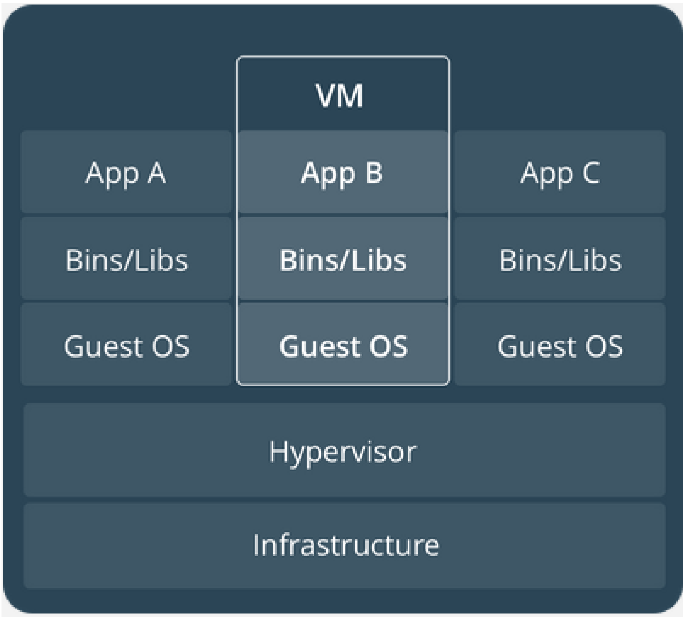

Figure 14: Virtual machine architecture, source [7]

Docker container vs. Virtual machine Both virtual machines and docker containers are two ways how to deploy multiple, isolated applications on a single platform. They both offer a way to isolate an application and its dependencies into a self-contained unit that can run anywhere. They both offer some kind of virtualization. They differ in architecture, see Fig. 14, 15.

Let's start with virtual machine (Fig. 14) and its layers description from bottom up:

- *Infrastructure* - It can be a PC, developer's laptop, a physical server in data-center but as well a virtual private server in the cloud as Microsoft Azure or Amazon EC2.
- *Host OS* - Host operating system. In case of native hypervisor this layer is missing. In case of hosted hypervisor it is probably some distibution of Linux, Windows or MacOS.
- *Hypervisor* - Also called virtual machine monitor (VMM). It allows to host several different virtual machines on a single hardware. There are two types of hypervisors:
  - Type 1 - Also called *bare metal* or *native*. This type is run on the host's hardware to control it as well as manage the virtual machines on it. It is much faster and more efficient. This type hypervisors are KVM, Hyper-V or HyperKit.
  - Type 2 - So called *embedded* or *hosted* hypervisors. There hypervisors are run on a host OS as a software. They are slower and less efficient on the other hand they are much easier to set up. It includes VirtualBox or VMWare Workstation.
- *Guest OS* - Guest operating system. Each VM require own guest operating system which is controlled by hypervisor. Each guest OS needs its own CPU and memory resources and starts on hundreds megabytes in size.
- *Bins/Libs* - Each guest OS needs various binaries and libraries for running the application. It can be *python-dev* or *default-jdk* packages as well as personal packages to run the application.

- *Application* - The application source code that is desired to be run isolated. Therefore each application or each version of application has to be run inside of its own guest OS with own copy of bins and libs.

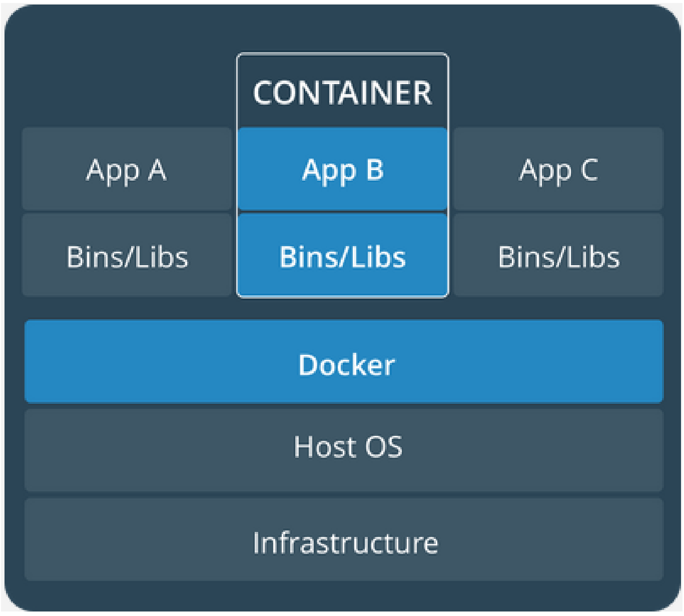

Figure 15: Containers architecture, source [7]

Now, what is different regarding containers (Fig. 15)

- *Infrastructure* - PC, laptop, physical or virtual server.

- *Host OS with container support* - Any OS capable of run Docker. All major distributions of Linux are supported and there are ways to run Docker even on MacOs and Windows too.

- *Docker engine* - Also called Docker daemon. It is a service that runs in the background on host operating system. It manages all interaction with containers.

- *Bins/Libs* - Binaries and libraries required by the application. They get built into special packages called *Docker images*. The Docker daemon runs those images.

- *Application* - Each application and its library dependencies get packed into the same Docker image. It is managed independently by the Docker daemon.

But the architecture is not the only one difference:

- Docker use Docker daemon to manage containers, hypervisor manages virtual machines.
- The Docker daemon communicates directly with host OS and manage resources for each container.
- VMs usually boot up in minute and more, containers start in seconds.
- Docker virtualizes operating systems, using VMs is hardware virtualization.
- VM and container vary in size. VMs start at hundreds of megabytes. Container can be smaller then one megabyte.
- Containers share the kernel altough they are isolated. VMs are monolitic and stand-alone.

Dockerfile Dockerfile is a core file that contains instruction to be performed when an image is built. It usually consists from commands to install packages, calls to other scripts, setting environmental variable, adding files or setting permissions. In Dockerfile there is also defined what image is to be used as base image for the build.

Dockerfile instructions

- *FROM* - The FROM instruction defines the base image for next instructions and initializes a new build stage. Every Dockerfile has to start with FROM command. The only exception is ARG command which can be before FROM command.
- *ARG* - The ARG instruction defines a variable that users can pass at build-time to the builder.
- *ENV <key>=<value>* - The ENV instruction sets the environment variables. It is key-par value.
- *LABEL* - The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. It can be anything from version number to description.

- *ADD <src> <dest>* - The ADD instruction copies files or directories from source and adds them at the destination path. It also unzip or untar files when added.
- *COPY <src> <dest>* - Similar to the ADD instruction it copies files or directories from source and adds them at the destination path. This command doesn't provide any kind of decompression.
- *RUN <command>* - The RUN instruction will execute any defined command and commit the results.
- *CMD ["executable","param1","param2"]* - The CMD instruction provides defaults for an executing container. It can include an executable. In case the executable is ommited the CMD instruction must be used together with the ENTRYPOINT instruction. There can be only one CMD instruction in Dockerfile. In case there is more CMD the last one will be used.
- *ENTRYPOINT* - The ENTRYPOINT defines a configuration of container that will run as executable.
- *WORKDIR /path/to/dir* - The WORKDIR instruction sets the working directory for any RUN, CMD, COPY and ADD instruction that follow in Dockerfile.
- *EXPOSE* - The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.
- *VOLUME* - The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

Except the FROM instruction, all the instructions can be defined from command line when starting docker container. There are more Dockerfile instructions however they are not relevant for this thesis as there are never used in practical part.

Listing 4: Dockerfile example

```
ARG VERSION=0.9.22
FROM phusion/baseimage:${VERSION}
LABEL maintainer="devs@pywps.example.net"

RUN apt-get update -y && apt-get install -y \
        git \
        python3 \
        python3-dev

RUN git clone https://github.com/geopython/pywps-flask.git

WORKDIR /pywps-flask
RUN pip3 install -r requirements.txt

RUN mkdir /etc/service/pywps4
COPY pywps4_service.sh /etc/service/pywps4/run
RUN chmod +x /etc/service/pywps4/run

EXPOSE 5000
ENTRYPOINT ["/usr/bin/python3", "demo.py","-a"]
```

Part III

Implementation

Popsat implementaci

Seznam použitých zkratek

HPC	High Performance Compute
KVP	Key Value Pair
OGC	Open Geospatial Consortium
PID	Process identifier
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	Virtual Machine Monitor
WPS	Web Processing Service
WMS	Web Map Service
WFS	Web Feature Service
WCS	Web Coverage Service
XML	eXtensible Markup Language

References

[1] Mark Reichardt *OGC Newsletter - October 2004, OGC document number 04-043* [online]. URL: <http://www.opengeospatial.org/pressroom/newsletters/200410>

[2] Sam Bacharach *OGC announces Web Processing Services Interoperability Experiment* [online]. URL: <http://www.opengeospatial.org/pressroom/pressreleases/414>

[3] Open Geospatial Consortium Inc. *OpenGIS ® Web Processing Service, OGC document number 05-007r4, ver. 0.4.0* [online]. URL: <https://portal.opengeospatial.org/files/?artifact_id=13149&version=1&format=doc>

[4] http://www.opengeospatial.org/pressroom/newsletters/200410

[5] Open Geospatial Consortium *OGC® WPS 2.0 Interface Standard Corrigendum 1, OGC document number 06-121r3* [online]. URL: <https://portal.opengeospatial.org/files/?artifact_id=13149&version=1&format=doc>

[6] Open Geospatial Consortium Inc. *OGC Web Services Common Specification, OGC document number 14-065* [online]. URL: <https://portal.opengeospatial.org/files/?artifact_id=20040>

[7] Docker *Docker documentation* [online]. URL: <https://docs.docker.com/>

[8] Jáchym Čepický, Luís Moreira de Sousa *New implementation of OGC Web Processing Service in Python programming language.* [online]. URL: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B7/927/2016/isprs-archives-XLI-B7-927-2016.pdf>

[9] Jorge de Jesus, Luca Casagrande, Jáchym Čepický *PyWPS a tutorial for beginners and developers* [online]. URL: <https://www.slideshare.net/JorgeMendesdeJesus/pywps-a-tutorial-for-beginners-and-developers>

[10] PyWPS developers *PyWPS documentation* [online]. URL: <http://pywps.readthedocs.io/>

[11] Victor Stinner *The pysandbox project is broken* [online]. URL: <https://lwn.net/Articles/574323/>

6 Seznam tabulek a obrázků

List of Tables

List of Figures