Multi-Cloud Deployment and Execution of Earth Observation Services

João Pedro Martins Serras
Instituto Superior Técnico, Universidade de Lisboa

***Abstract*—At this day and age, with the progress in engineering, spatial data is being harvested at a high speed and volume with several formats from a continuously collection of satellites. For scientists to study and utilize this data it first requires some transformations from Earth Observation (EO) tools.**

However, these tools can be resource-heavy, restricting the use to a small and highly skilled community. With the arise of Cloud Computing, scientists do not need to worry with the cost of managing hardware. Furthermore, Cloud Computing releases the scientist from the burden of understanding the infrastructure of their equipment. As a result, Cloud Computing became the natural solution.

The procedures of deploying the EO tools on Cloud Computing environments emerges with some critical challenges. It requires to package these tools, provision as well as configure a cloud instance, deploy and expose them to the web context. Therefore, we aim to automate each of these steps by exploring the already existing technologies.

Our focus will begin with the creation of a high degree continuous delivery pipeline. This will be the main base functional aspect of this thesis. In addition, the monetary cost and how to maintain it to a minimum will still be taken into consideration. With this in mind, we aim to utilize an already existing work, titled Workflow Engine for Earth Observation Services to help with the validation of inputs/outputs when deploying the EO tools to the cloud, avoiding unnecessary runtime errors.

***Index Terms*—Earth Observation, WPS, Workflow, Composition, Validation, Translation.**

I. Introduction

At the present time, the European Union began an Earth Observation (EO) Program entitled Copernicus (former GMES). The purpose of Copernicus is to gather and monitor data gathered from a collection of satellites. This data is then used to produce products using EO tools. This data retrieved by the satellites are in the order of petabytes per year, hence the transformation of this data is resource intensive. Building and maintaining complex infrastructures are not feasible anymore. It requires a large monetary investment and a deep knowledge about the infrastructures, making cloud computing a much more appealing solution. [1].

Cloud Computing is a shared pool of configurable computer system resources and higher-level services that are delivered on demand to external customers over the Internet. It brings high availability and increased flexibility by adjusting to the customers requirements, without regard to where the services are hosted or how they are delivered [2].

Therefore, the European Commission has decided to invest in the development of Cloud Computing platforms specialized for the storage and exploitation of the Copernicus data, Copernicus Data and Information Access Services (DIAS). It eases the procedure of accessing to Copernicus data and information from the Copernicus services by providing data and information access alongside computing resources.

Exporting EO tool to the cloud environment rises some challenges.

- Each tool has a set of dependencies that are required to ensure it is the correct execution. Executing these tools on the cloud oblige these dependencies to be available in the environment. Thereby, a new problem arises. How do we package the EO tools?
- Succeeding this phase, the application needs to be deployed, transferred to the DIAS, and then executed. However, how do we deploy the tools over the different DIAS?

It is important to have in mind, the target audience of these tools are scientists that do not possess enough knowledge in cloud computing. The learning process can take quite some time and does not mean that the user will be proficient when using cloud technology. This can also be translated into unnecessary user errors. With all these points, we can state that our main goal is concerned with the usability. Therefore, we aimed to ease the accomplishment of the execution of a desired tool by its users in the cloud environment through automation.

It is important to denote, the data transforming tools need to be exposed to the users in such way that the user does not need to worry about the cloud management. Consequently, the ideal approach is to expose this tool through the web.

As stated before, our main concern is usability. Provide a system that enables the users to execute different EO tools over any cloud provider, without worrying about the problems that come with the deployment of these tools to the cloud. For this reason, we want to automate every step of the solution for the problem mentioned above in 1.1 Problem Definition.

In the light of this, it is possible to observe that it fits into the DevOps context. The process of developing software and release it to production can take several hours, if not days, and in the end, it can reveal to be unstable, when each step is done manually. DevOps practices aims to shorten the time between committing a change to a system and the change being placed into normal production [3]. This is done through the implementation of an automated Continuous Delivery (CD) pipeline, providing a high degree of automation [4]. It can be characterized in three stages:

- The first stage is triggered by the commit of the newly developed code. While in our case we don't develop new features to the same software, we can translate this phase to the adaptation of the EO tools to the web context;
- The second stage is responsible for the building of the application. The application binary package is created with all its dependencies. This stage perfectly matches the application problem challenge;
- The Production stage is responsible for the deployment of the application to production. In our case, this is translated to the provisioning of a new cloud instance to deploy the EO tool.

Consequently, DevOps practices depend in various tools to aid each step. Each of this stage corresponds to a sub problem mention in the section above. This thesis will rely in DevOps practices to produce the bigger picture of register an EO tool and deploy it to the cloud.

The Open Geospatial Consortium (OGC) is an international consortium comprised by several organizations with the objective of sharing, discover and reuse information as well as services related to the diverse EO fields. Every year, the OGC publish Engineering reports. These reports contain the activities that greatly contribute to the EO community. As already mentioned, our solution comprises in two separate features, the application packaging and the workflow execution using cloud computing. Although both display an impact in the EO community, the execution of workflows using cloud environments was the one with the biggest impact, fitting every category presented by the OGC to participate in the Engineering reports. As a result, our solution has been tested and verified and demonstrated in OGC Innovation Program (Testbed-15). The technical approach and the summary of the solution is included in the OGC testbed-15 Engineering report for the entire EO community to benefit from.

To accomplish our solution of providing an automated pipeline for both EO Tool Packaging and Cloud Deployment through the use of the cloud, we will begin by exploring the diverse existent tools for each step of the pipelines in chapter 2. Subsequently, we describe how we utilized them to support us during the automation process in both pipelines in chapter 3. Following the presentation of our solution, we proceed to validate it through usability tests in chapter 4. Ultimately in chapter 5, we present the conclusion of the thesis as well as the future work to be developed.

II. Background

In this chapter we enumerate the different existing technologies and tools, that help us achieve our goal of providing automation in both EO tool Packaging and Cloud Deployment pipelines.

A. EO Tool Packaging

Each EO tool has its own requirements that need to be satised for their correct execution. As mentioned in [5] [6], these requirements can range from specify the location of information, work areas of the disk that will be used to process conguration files, install libraries and other applications. As a result, it is necessary to package them together and transfer them to the cloud environment. The packaging can be achieved either through Package Management Systems or Containers.

Package Management Systems (PMS) came forth with the urge to automate the process of setting up the environment properly to run the applications by bundling all this related information together into files. The most utilized PMS are Red Hat Package Manager (RPM) and Advanced Packaging Tools (APT). Containerization is a virtualization technique that was developed for the purpose of isolating process resources in the absence of any hardware requirements [7]. It provides an isolated environment specic for the application it encapsulates. This environment solely includes the essential for the application to execute. [8]. Linux Containers (LXC[1]) and Docker are the most utilized technologies that takes advantage of Containerization.

We decided to use Docker as the main tool for the EO Tool Packaging. Docker is available to all operating systems. Contrary to Docker, LXC are only available to Linux distributions. To encapsulate an application Docker, uses Dockerfiles. Dockerfiles consist in a single file with the description of every instruction to be performed during the encapsulation [9]. Dockerfiles present a more friendly user syntactic, simpler for automation. Equally to LXC, RPM and APT only function on Linux distributions, where APT only works in Debian. Both RPM and APT require the use of a file to specify all the dependencies. However, they present higher complexity. RPM require the use of eight sections and APT the use of four different files. Additionally, RPM requires one file for each hardware configuration and does not install the requirements, needs to be done manually.

B. Cloud Deployment

With the EO tool packed it is required to provision a cloud environment. In other words, a Virtual Machine (VM) Instance must be launched with the desired hardware requirements capable of supporting the application. This can be done with Conguration and Management Tool and Server Provisioning Tool.

Configuration and Management Tools (CMTs) are tools utilized to automate the management of IT Infrastructures. They provide modications to the infrastructure conguration as well as provision [10]. New CMT are arriving the market, being Chef[2], Ansible[3] and Kubernetes the most dominant.

Server Provisioning Tools (SPT) main purpose is provision. The term modication in SPT, revolve around deployments of entirely new servers [11]. They do not alter existent server congurations to accommodate new software releases. They only require the description of the end state of the server, it does not require the description of each step to achieve this

[1]https://linuxcontainers.org/
[2]https://www.chef.io
[3]https://www.ansible.com

end state. At the present, Terraform[4] and Opensource-Heat are the most popular SPT.

We will build a solution upon Ansible. Ansible and OpenStack-Heatuse YAML files to specify the desired configuration of the servers, easing the automation process. However, Ansible possess a wide variety of cloud providers in contrast with OpenStackHeat, who only utilize Openstack framework and AWS. The same justication can be utilized for Kubernetes. Kubernetes either uses Kubernetes or Kubespray for provisioning. Kuberspray however uses Terraform. Kops only has compatibility with AWS and Google Cloud Engines (GCP). In relation to Terraform, to have full access to its API it is required the paid version. Therefore, Ansible has the same advantage in comparison with Kubernetes Kubespray. At last, with respect to Chef, it utilizes a Client-Server architecture, requiring extra steps. Moreover, regarding provisioning, Chef utilizes drivers developed externally. As a result, Chef is lacking compared to Ansible.

C. Web Processing Services

As stated before, in the Introduction, there is a necessity to export EO tools to the web context. The Open Geospatial Consortium (OGC) developed an Interface Standard, Web Processing Services (WPS). WPS is a web service that enables the execution of computing processes and retrieval of metadata describing their purpose and functionality [12]. These processes are characterized by receiving inputs, executing algorithms and delivering outputs. The input and output can be Literals, encodes atomic data (scalars, linear units ...), ComplexData, does not describe a particular structure. Passed values must match with the given format. BoundingBoxData are coordinates in the form of an array that represents an area. To enable monitor, execution of processes and retrieval of data, WPS offers the core functions through HTTP POST and GET request. GetCapabilities to enable the retrieval of information about which operations the server. DescribeProcess to obtain a brief description of the algorithm it runS. Execute to request the application to execute.

Web Processing Service Transactional (WPS-T) came to light as the answer to package WPS processes and deploy them in WPS servers at runtime [13]. To achieve this, WPS-T make use of the core WPS interface and proposes and extension to this. The extension consists in adding two new operations, deploy and undeploy process, both requested via HTTP-POST through new endpoints. As explained in the WPS section, the GetCapabilities provides information about which operations are supported by the server, it must suffered modications in order to accommodate the new Operations.

Overall, in the interest of offering EO tool via WPS, we are going to utilize of templates for the transformation of an EO tool into a WPS. The source code, types of inputs/outputs and metadata will be the major attributes specied in the templates. In spite of dynamically register the EO tools into the WPS server, we deploy the WPS specied with already the predened EO tools. WPS-T is still in development and only start to arise recently. Under these circumstances, WPS-T may still be prone to unexpected errors. With the deployment to the cloud, unforeseen faults will result in a monetary loss.

In summary, we will add a new stage responsible for the encapsulation of the EO tool into a WPS standard using templates, in our CD pipeline after committing the code.

1) Workflow Engine for Earth Observation Services: Workow Engine for Earth Observation Services (WEFEOS) allows the creation, execution and storage of workows. A workflow is a chain of WPS linked together. This allows the redirection of the output of a WPS as an input to other WPS and so forth. What make WEFEOS unique is his validation. There is no other workflow management system that provides validation. WEFEOS validation can be split into four unique levels. Each level is solving a sub problem of the whole validation on its own [14].

In the rst level, WEFEOS is responsible to check if the described process of a WPS process specication complies with the standard. In the second level all data types and restrictions ought to be respected.Once third level is reached, the validation starts to resolve around the workow creation. Internal inputs compatibility and minimum/maximum occurrences are examined. To not miss any workow requirements, for example the missing data, the workow specication is inspected and veried against ia set of rules on how the workflow must be specified. In the nal level, the only missing piece comes down to runtime validation. Being able to validate the output originated by the execution of a WPS before being sent to the next WPS in the chain.

With special attention to minimizing the costs, fault prevention becomes vital. The main source of errors came from the linkage of WPSs, creating a workow chain. Henceforward, we will use WEFEOS to prevent them.

III. Implementation

In the wake of defining the technologies required for the EO Tool Packaging and Cloud deployment automation, we will now explain our solution.

A. System

We will be using as the base for this thesis WEFEOS. WEFEOS system is comprised by WPS-V-WEB and WPS-V. WPS-V-WEB is responsible for the workflow management in the web browser. It has available a dashboard with a menu. In here, the users are able to compose, download and load workflows. During the composition of the workflows, it alerts the users of the compatibility issues between WPS processes. Nevertheless, WPS-V-WEB also provides a console to display the errors of the input data specified. To detect these errors, it is required a syntactic validation available with the use of WPS-V. Furthermore, WPS-V provides dynamic validation for the output generated by each WPS process.

Multi-Cloud Deployment and Execution of Earth Observation Service (MCDEEOS) is the module responsible for providing the EO Tool Packaging. It is through this module

[4]https://www.terraform.io

that the EO tool is wrapped into a WPS, dockerized and stored in a repository for later usage. In addition, MCDEEOS is also responsible for forwarding and preparing, if necessary, the workflow to a workflow engine for its execution, as well as to provide deployment of the WPS's to the cloud, validation of the outputs by using WPS-V-WEB and their termination. To offer these functionalities to the users we developed the MCDEEOS-WEB. MCDEEOS-WEB is responsible for providing a web interface for the users to specify all the information necessary to perform both application package and workflow execution on the cloud.

B. MCDEEOS

We will proceed to explain how we were able to accomplish our goals of packaging an EO tool and deploy a WPS for the workflow execution in our core back-end package MCDEEOS.

1) EO Tool Packaging: The first stage corresponds to the wrapping of an EO tool into a WPS. In other words, we need to develop the standard interface of a WPS for retrieving metadata about the numerous processes it contains and their execution through the methods described in chapter 2.3.

To wrap an EO tool into a WPS, we utilize a python WPS implementation named PyWPS. Through this we avoid developing every REST endpoint from scratch. The result from the execution of every WPS methods is an XML file containing every detail about the output of the method executed. It is crucial that these XML files are well formed and according to the standard given that WEFEOS use them to perform its syntactic validation. Therefore, the most viable choice is to utilize PyWPS. To use PyWPS as the base for turning the EO tool into a WPS, we first analyzed which files it required to change in order to accommodate for each tool. As a result, we identified a bare minimum of two files, since each WPS has at least one process. In other words, we need one file for each process and another for the main file. This file can be seen as five pieces. The first corresponds to the imports, where the required libraries are specified. The second is the name of the class which corresponds to the name of the process. The third is the use of a list containing functions specifying each data type for all inputs and outputs. Fourth is the specification of the metadata about the process. Lastly, it is required a handler function for specifying what is the result obtained from the execution of the process. To replicate such file, tailored for each process of the EO tool, we use a pre-defined template. This template uses variables, that we can change when the users request an EO tool wrapping. As a result, we are going to need variables for the imports, class name, inputs and outputs. Additionally, more variables are used inside the handler. To execute the EO tool we are going to use a library that enables us to call external commands. Therefore, we need a variable for the command line and the parameters, i.e, the inputs. It is required a variable to move the outputs file to the folder that is available for the user to access via REST call, since the outputs of the EO tool might be located outside the WPS folder. Additionally, it is also required the specification of the location of the outputs in the handler.

The second file that we need to change is the main file. It is in the main file that all the endpoints are specified. However, we only need to change a code line of the file. This file contains a processed list where the processes objects are instantiated. Therefore, to instantiate the processes we use a template with a simple variable.

With the EO tool wrapping finished, we move to the next phase of the pipeline, the encapsulation process. For this, we opted to utilize Docker since it is the most flexible technology when compared with the others. It is important to mention the EO tool is required to be available at the Dockerhub for us to have access to it. As a result, we use the Docker images containing the EO tool as the base image for the encapsulation of the WPS. Thus, the Dockerfile for the image building is going to differ from each EO tool. To create this Dockerfile we used once more a template. In here, we use the variable image to provide the EO tool Docker image along commands to create and transfer the WPS files to it and install all its dependencies.

During the development of our system we discovered that WEFEOS was designed with the objective of execution workflows locally. This rose a problem in which WPS's must be already running in order to validate their syntactics. To solve this problem, when the users perform the EO Tool Packaging in our system, we do an initial deploy to obtain XML file with the full specification of the WPS. Every time WPS-V requires to validate the syntactic of a WPS, it will query our system to obtain the designated XML file. As a result, a new stage arose in the pipeline after the EO tool wrapping.

The process to the storage of the WPS image was rather straightforward. We choose to use Dockerhub as the repository. As result, Docker already provides all the commands to store and manage images in the Dockerhub through the REST API.

2) Workflow Execution in Cloud: With the goal to deploy WPS in the cloud in order to execute them in a chain, our priority was to begin with a simple WPS deploy. To achieve we used Ansible. Ansible utilizes YAML templates to execute their modules. For each stage of the pipeline templates were created. We are going to present the challenges that arose during the development of each stage.

Considering Copernicus DIAS cloud providers had not fully developed their API in time, we search for another viable cloud provider that has access to Copernicus Sentinel data. Through this search, we found AWS meet this criterion. We will further describe, step by step, each task composing the Ansible playbook to achieve provisioning and configuration in AWS.

It is important to bear in mind, our system provision instance in the users AWS account. Therefore, users are forced to provide AWS secret and access keys. Along with the keys, the region needs to be specified by the users where the provision is going to occur.

To enable the provisioning in AWS some configurations must be done prior to the provision task. To have access to the cloud instance, it is required the use of an SSH key. To accomplish this Ansible has an EC2 module that enables this

creation. Upon creating the key, it is required to save it for later use.

Even though we run tasks multiple times, Ansible has the ability to detect if the criterion of the tasks have been met, skipping the task if they were. Every cloud instance must have an IP associated in order to be connected to a computer network. The IP of a cloud instance is compelled in three pieces. The first identifies the Virtual Private Cloud (VPC) the second the Subnet inside the VPC and the third the machine. Therefore, it is required to create a VPC, along with a VPC Subnet in the given region. The assignment of the last bits of the IP used to identify the machine is done dynamically by AWS. Although the VPC is already specified, the VPC is still a private network not ready for use. For the execution of WPS they require access to the Internet. Therefore, it is essential to create an Internet Gateway and associate it to the VPC along with setting up a route table to enable the connection of the Subnet to the Internet. Moreover, for users to access the WPS through web browser, it is required to create a Security Group that allows HTTP traffic.

At long last, with the configurations done we only require one more piece of information before specifying the provisioning task, the image id. Since the image ids changes from region to region, we used an official module that enables us to perform a query to obtain information about images in AWS, where ids are specified in. With every preparation done, we can now compel it to specify every parameter in the provisioning task. At the end of provisioning, we must store it along with the key location and user in the Ansible Inventory to have access to it afterwards.

With the cloud provision accomplished, it is now required to devise a template to perform configurations. While the syntactic validation is performed locally, the dynamic validation is performed on the cloud instance instead. By performing the validation of the outputs generated by the WPS execution on the cloud instance we avoid the need to transfer these outputs to the local machine. As a result, we encapsulated WPS-V in a Docker image. Despise WPS-V being in the same cloud along with the WPSs, Docker containers are isolated from each other, the data still needs to be transferred from one container to the other. To avoid this, we created a volume, mount it to the outputs folder of the WPSs and attach it to the WPS-V container. Thus, data is shared between containers, allowing the direct access by the WPS-V.

With the use of the Docker images in the cloud instance, they are not available to the outside world. They are running in a local network created by Docker. To solve this, we used Nginx that provides proxy redirection. This allows the requests performed to the cloud instance to be redirected to the Docker containers. To enable the proxy redirection, we need to provide a configuration file. To differentiate the path of the validator from the WPS we added a tag to the location path. As a result, since the WPS name change from WPS to WPS, we use a template to create this configuration file. To speed up the process, we created our own image when provisioning. This image contains the Docker, Nginx and the WPS-V container, since they are required in every cloud instance. With this approach, we avoid wasting time in the configuration along with pulling the WPS-V Docker image and instead we are only required to send the Nginx configuration file, execute a task to order the WPS-V container to start and restart the Nginx.

Until this point, we have been using the default disk space specified by the different cloud providers. However, this disk space may not be enough for the execution of the WPS. Considering the disk space required changes from WPS to WPS, the best approach is to ask the users how much space it is required. However, we provide two different choices when the users specify the disk size. The first choice consists in using the maximum value possible from the instance type used to provision the cloud instance. While the second choice consists in using the value the user specified.

Our system is not bound to AWS as the only cloud provider. We design our system with the desire to enable the addition of cloud providers in the future. As a result, to verify the difficulty of augmenting the system with diverse cloud providers, we decided to add Microsoft Azure to our system.

To increase our system with Microsoft Azure, we had to develop new playbooks for Ansible. The main differences from AWS were in the introduction of a network interface, which replaces the AWS gateway and the need for configuring routing table, and the creation of a public dynamic IP, which AWS assigns automatically. Overall, the implantation of Microsoft Azure was achieved in two days. However, more time was spent due to the previous versions of Ansible 2.8.0 having some bugs disabling the provision of instances and the learning how to obtain the crucial keys to enable the provisioning through Ansible.

Once the cloud deployment was accomplished, we aimed to automate the execution of a chain of WPSs, a workflow. To achieve this, we used Workflow Engines. The users begin by specifying a workflow with the WPSs they want to chain along the WPSs URL, and the inputs required. At the end, the users submit the workflow.

When a workflow execution is triggered, we load the workflow engine with the workflow and search for the first WPS in the chain. Once we have this information, we begin by initiating the cloud deployment pipeline. Subsequently, we notify the workflow engine to start the workflow execution. Thus, the workflow engine performs an execution request to the WPS deployed. At the end of the execution, the control is passed to our system to send a request to the WPS-V, running in the same cloud instance, to validate the outputs acquired. Since this is the first WPS in the chain, it is not possible to terminate it. The outputs obtain from its execution are going to be utilized as inputs for the next WPS. Subsequently, we repeat the same process for the next WPS. Following the execution, we do not require the previous WPS to be running, since we are not going to utilize its outputs. Thus, we terminate its cloud instance. The last WPS is left running for the user to have access to the results.

Considering this, we began by concentrating on developing

our own Workflow Engine in order to test the waters. Allowing us to improve the understanding of how to develop the orchestration as well as to pinpoint with more precision the location of errors along the process without worrying about the problems that may be caused by the Workflow Engine itself.

Afterwards, we augmented our set of workflow engines with Camunda. Camunda is the workflow engine currently being use in the EO community. Workflow engines might have unique characteristics that differentiates them from each other. Camunda has available a graphical user interface that enables the users to observe the execution in real time. Hereby, we provide a certain degree of flexibility by having multiple workflow engines available to the user when he desires to execute a workflow. To fully integrate Camunda in our system, it was required a week. However, the major difficulty was not in adapting it to our system but rather in learning how to utilize its API. Contrary to our workflow engine, Camunda utilizes BPMN. Since BPMN is a business process notation and JSON is an exchange data format, there is no direct relation between the two. Therefore, our first challenge was to translate the workflow specification in JSON to BPMN through its java API.

At the end of developing the workflow orchestration, we searched for more methods to decrease the deployment process time. From this, we emerged with Execution modes. Execution modes are nothing more than different methods to perform the deployment. In other words, it is stated how to group WPS into a single instance, if they were meant to be executed in the same cloud region. By grouping WPS together, we avoid losing time preparing another cloud instance when we can reutilize the previous.

Our system provides three different execution modes. The first consists in deploying a WPS per cloud instance. The second is the deploy and execute each WPS in the same cloud instance if they have the same hardware configuration. In other words, if there is a WPS with higher requirements, it will be deployed in a different cloud instance. The third option is to and execute each WPS in the same cloud instance, however the cloud instance will be provision with the higher hardware configuration required by the WPS. To augment our system with these execution modes, it was not required any major modifications. The only mandatory change was to the Nginx configuration template to accommodate more WPSs.

C. MCDEEOS-WEB

To enable the users to interact with the system, we design a clean and simple interface available through our front-end module, MCDEEOS-WEB. Through this, users can package EO tools, execute workflows into the cloud and observe all the available WPS in the system. The OGC, along with a few companies, offers information regarding WPSs already running in a server using catalogs. We enhanced our system with WPSs discovery from these catalogs along the registration, search and removal in a specific catalog of a WPS.

1) EO tool Packaging: To accomplish the EO tool Packaging, users are required to provide information regarding the EO tool. This information is compelled by two categories. The first category is related to the EO tool itself. In here, it is specified the RAM and CPU required, the Docker repository of the EO tool and the desired name for the WPS. The second category is the information regarding the EO tool processes, the different inputs and outputs types of these processes. WEFEOS-WEB provide a web-form for the user to describe all this information.

Even though the outputs have three possible data types, only ComplexData can be worked with. With the possibility of a WPS generating multiple LiteralOutputData, it is not feasible to detect and parse each LiteralOutputData. Moreover, through observing the WPSs available in the industry they all work through files, therefore only ComplexOutputData is generated. Along with the data types users must also specify the location where the outputs will be stored inside the docker containers. This allows our system to make them available in the future for the chaing process. Nevertheless, we also provide fields for metadata as the author, title, description, abstract and the most important the command line needed to execute the EO tool.

2) WPSs List: With several EO tools being packed into a WPS using our system, MCDEEOS-WEB can present all available WPS. Through the use of a table, it is displayed every WPS the users owns. One key component of our system is the ability to share WPS between users. For this reason, it crucial to also display the WPSs that belong to other users. This is done through the use of another identical table.In case a user required more information regarding a WPS, the name of the WPS in the table must be clicked, triggering an event to create a new table with the list of processes available in the WPS along with their identifiers, titles and description.

3) Workflow Execution: To develop MCDEEOS-WEB we decided to use the same programming language as WPS-V-WEB, allowing us to import WPS-V-WEB module. However, to import WPS-V-WEB, minor modifications were required. Workflow Execution section can be seen in two different parts, the first creation and validation of the workflow and the second the specification of cloud information. With the success in use WPS-V-WEB, its graphical user interface (GUI) remained the same. It has available a drag and drop dashboard with a menu for creating WPS, importing workflows and save workflows. When selecting a new WPS a new web-form is available with dropdown boxes to select which WPS and the process the users wants. With enough WPS, users are now able to chain then to create an actual workflow. At the end of chaining, users must specify every information required for the inputs of each WPS and its chaining.

Finally, with every important detail is specified, users can validate. WPS-V-WEB has available a console, in which error messages are displayed resulted from the validation. Once the workflow is successfully validated, the button to execute the workflow is enabled. However, users still need to provide critical information through dropdown menus. To execute a workflow, users must choose the Workflow Engine, provide the execution mode, select the cloud provider, the region for

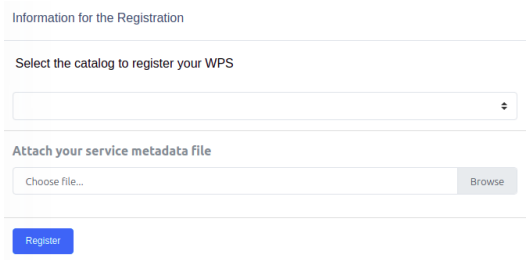

Fig. 1: Interface for the WPS catalog registration

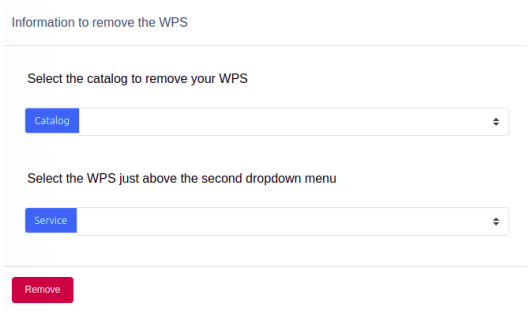

Fig. 2: Interface for the WPS catalog removal

each WPS specified in the workflow, the size mode for each instance and the necessary size.

D. WPS Catalog Registration

For the users to accomplish the registration, a catalog must be selected for the service registration in the dropdown menu. Subsequently, the users only require uploading a Geojson with every crucial detail of the service for the registration, fig:Interface for the WPS catalog registration.

E. WPS Catalog Removal

To remove a service from a catalog, it is presented two dropdown menus, fig:Interface for the WPS catalog removal. The first is used to specify the catalog. The second is to specify the service identifier. It is important to mention users can only remove services they registered.

F. WPS Catalog Discovery

In case a user wants to search for a service, we provide a service discovery. In other words, the user provides a part of an identifier name with the catalog id and our system will present, through the use of the, every process the service containing the identifier has.

G. WPSs Catalog List

In the same vein as the subsection:WPSs List, our system also displays the WPS present in every catalog through the use of tables with the addition of a collum containing the Endpoints. Despite presenting every WPS available in every catalogue, to facilitate the navigation, our system provides a dropdown menu for the users to select the catalogue they want to see.

Validation

In this chapter we aim to validate our solution. Revise the set goals, define the provide the description of the steps required for the accomplishment of the EO Tool Packaging and Workflow Execution in the cloud environment, list of possible errors to keep track, how the execution of the validation will take place, the results obtained along with an analysis. It is important to mention our system is unique, the only comparison is to do the process manually. However, it requires days to not only learn but also execute both EO Tool Packaging and Workflow Execution. The complexity of them, is far greater than using our system. Therefore, it will not be taking into consideration.

IV. Goals

To perform a thorough validating for our system, it is first required to define means of comparison between the two approaches. In other words, we set goals based on the desired functionalities by the EO community. As mention above, we want to provide cloud computing as a solution while providing the finest user experience, without placing extra burden on the user. As a result, we can translate this into the following usability requirements:

- The end users learn with ease how to use the system;
- The end users use with ease the system;
- The system is able to prevent and recover from errors.

V. EO tool Packaging

The validation of our system is done in two parts. The first part corresponds to the EO Tool Packaging.

The procedure of wrapping an EO tool generally has an impact in the performance of the application when not performed case by case. Traditionally the addition of a new layer may involve transformations for the inputs as well as modifications to adapt the code. However, in our situation the new layer aims to provide descriptive information regarding the EO tool itself through HTTP. The inputs do not require any transformations. The new layer simple calls the EO tool with the inputs sent. Even though it does not interfere with the performance itself, by performing the EO Tool Packaging through our system, errors resulting from the EO tool are not recorded.

With our system, all these phases are automated. However, the users ought to provide information regarding the EO tool. One possible limitation of using our system to provide automation is the specification of the outputs in terms of system expressiveness. However, in the EO field the EO tools only produces ComplexData types. Thus, we do not loose expressivity. The validation of our system is measured through the filling of the information required for the EO Tool Packaging.

A. System Steps

To use our system to perform an EO Tool Packaging, it is required to provide the information requested in our web-form. This is translated in the following steps:

- Specify the name and hardware requirements for the application;

- Specify the Docker repository for the base image containing the EO tool;
- Specify the input and output formats;
- Specify the command line to execute the EO tool;
- Specify the location of the outputs.
- Select the submit button.

B. Errors

The automation of the EO Tool Packaging requires user input to be accomplished successful. However, errors may still occur in the event of the users providing either wrong information or malformed images. Considering that our system is not able to pinpoint which error of the previous errors took place, it is still possible for the users to verify the information provided since it is not erased from the web-form. This allows to further restrict the possibilities of where the error happen and to free the user from filling all the information again. Upon correcting the errors, the users must submit again to start the process from scratch. The potential errors when using our system can be reduced to the following list:

- Misinformation during the specification of the inputs and outputs data;
- Missing information during the specification of the inputs and outputs data;
- Wrong Docker repository;
- Malformed base application Docker image provided.

C. Execution

With the clarification on the steps required along with the list of possible errors, we are now going to unfold how the execution of the validation of the EO Tool Packaging is going to be accomplished.

In the first place, it is important to have in mind scientists are inexperienced EO developers with the minimum knowledge in cloud computing. Their focus is on the analysis of the results obtained using EO tools. Given that, our target audience, when developing our system, is the scientific community. Therefore, our validation is performed with scientists as the users. Each user is going to utilize their own EO tool. Prior to start, it is required to learn how to perform each step. Each user is going to search for information regarding the use of our system. This information is available through a guide. We measure the time spent in the whole process of learning. However, it is normal to still search information during the execution of the task. We are going to have into account this time and add it in the end. At the end of measuring the learning time.

At the same time, we also want to examine if our system is faster to use. We will measure the time spent by each user using our system to perform the EO Tool Packaging. It is vital to perceive we are concerned with usability and not performance. Thus, during the automation phase, the time the system requires to perform the pipeline is not going to be considered.

During the execution of the task, it is common for the user to make mistakes. Through the list of possible errors defined above, we will track how many the user made during the execution of each step.

D. Results

The validationwas accomplished with three users, during the third week of October 2019. The users age ranged from 32 to 43. All the three users have a Computer Science degree, two of them are working now as Technical Manager, while the other user is working as a Project Manager. Although, the users have a background in Computer Science, none of the users had interacted with our system prior to the validation, they were inexperienced users.

Upon finishing the usability tests, both tasks of learning and executing were accomplished with success. We condensed the results obtained into two tables. The tab:EO Tool Packaging times recorded from the system validation contains the time measured for each user in seconds. While the tab:EO Tool Packaging errors recorded from the validation presents all the recorded.

Learning Phase	Execution Phase
471	445
432	451
352	542

TABLE I: EO Tool Packaging times recorded from the system validation

Mistakes	Number of occurrences
Misinformation during the specification of the inputs and outputs data	0
Missing information during the specification of the inputs and outputs data	1
Wrong Docker repository	0
Missing information during the specification of the inputs and outputs data	1
Malformed base application Docker image provided	0

TABLE II: EO Tool Packaging errors recorded from the validation

Our system learning curve is very low regarding the EO Tool Packaging. It averages 418 seconds. The reason behind this, is due to our system only requiring simple information about the EO tool. Therefore, our system, for the EO Tool Packaging simply describes the details required in each field, containing a more detailed description on the more complex fields. In addition, it also uses figures from the system itself to help in the visualization.

The use of straightforward information with the support of the easy reading documentation translated in faster times during the execution phase. It averages 479 seconds. It is important to mention during the execution phase a user forgot to

fill a crucial field regarding the ComplexDataOutput. However, our system was able to detect and notify the user straight away. It is important to have in mind, the complexity of the EO tool has an impact in both time and errors. The more complex the more prone and time consuming the procedure is.

Overall, based on the results obtained, it is possible to state our goals were fully achieved. We were able to build an easy to learn and use system regarding the EO Tool Packaging.

VI. Workflow Execution

Succeeding the validation of the EO tool Packaging befalls the Workflow Execution validation. With the EO tool packed we must now provision and configure the cloud instance to deploy and execute the WPS in the workflow chain. On a similar note to the EO Tool Packaging, our system also requires user inputs to fully automates these stages.

A. System Steps

Our system integrated WEFEOS to provide management and validation of workflows. Therefore, the users will utilize WEFEOS to create and validate their own workflows. Furthermore, it is required additional information regarding the cloud and instance. This can be translated in the subsequent steps:

- Specify the WPSs in the dashboard;
- Link all the WPS's in the dashboard;
- Specify the inputs for each WPS;
- Validate the workflow;
- Specify the workflow engine and execution mode;
- Specify the cloud provider and region for each WPS;
- Specify the hardware storage for each WPS;
- Select the submit button.

B. Errors

Using WEFEOS syntactic validation, we can prevent some of the manual errors such mismatching data types between inputs and outputs. However, it is still possible for the WPSs to produce an incorrect output. Regarding this occurrence, we can detect with the dynamic validation of WEFEOS, stop the whole workflow execution and notify the users. In addition, our system is not able to validate all user inputs. Information regarding hardware configuration specified during the EO Tool Packaging. The correction of these types of errors require to repackage the EO tool and start the process from scratch. All things considered, our system is only susceptible to misinformation regarding hardware configuration given by the users or the unexpected incorrect output obtained.

C. Execution

In the same vein as the EO Tool Packaging, the validation of the Workflow Execution process is performed by the same users. Additionally, we will also have a learning phase prior to the validation process. The learning phase will occur in the same manner as in the learning phase of the EO Tool Packaging. In the Workflow Execution validation, we are going to provide the WPS ourselves in order to chain them, releasing some extra burden on the users from creating a new EO tool and encapsulating them. The workflow for this validation is composed by two WPS's. The first WPS is responsible for the aggregation, sum, of the total rain in each time range. The second WPS is responsible for calculating the long-time averages of the aggregations, the arithmetic average. With usability as our focus, it is important to bear in mind we are not validating the execution of the WPS's in the workflow. What we are validating is if the users can build the workflow using our dashboard while expressing every single piece of data relevant for its execution, validation and deployment with ease. The comparison of the workflow execution in both cases is a problem more related with performance. In this field, our system is compelled have better results in the event of the WPS's not being executed in parallel. Additionally, the time to detect when the execution of a WPS ended is much higher when done manually. Our system gets notified as soon as the execution finished starting right away the deployment of the next WPS. Another major key point that directly has an impact in the performance is the fact that automation by itself is always faster than a person typing and doing task one by one. With the increase in WPS's, tasks and complexity the time spent in each step is significantly increase with the manual approach. Instead, we are only concerned with the users constructing the workflow in the dashboard, specify the inputs, validate them and state the workflow engine, execution mode, cloud provider, cloud region, size mode and the size required for the instance. Thus, we are going to measure the time spent in this process.

In the same vein as the EO Tool Packaging validation, we will also track the errors performed by the users in the cloud deployment using the list of possible errors described above.

D. Results

The validation of the execution of workflows in the cloud environment was accomplished in the same day as the validation of the EO Tool Packaging. As a result, usability tests were done for the same users, in the same week, utilizing the same environment and conditions.

Once the usability test for the Workflow Execution were finished, we grouped the results and displayed them into one table. In the same vein as the EO Tool Packaging, the tab:Workflow Execution times recorded from the system validation contains all the times measured in the validation test. During the execution phase of the Workflow Execution there was no occurrence of any mistakes using our system.

Learning Phase	Execution Phase
145	652
220	904
205	763

TABLE III: Workflow Execution times recorded from the system validation

In the same manner as the EO Tool Packaging documentation, the Workflow Execution guide is also simple to read.

By using real images of the system with described details on what to do and what each field requires, it was possible for the users to read it with ease and fully understand how the system works. The learning phase per person average 190 seconds.

In addition to the meticulous and simple documentation, the system provides the best tools to enable the users to provide every crucial information in a swift and secure manner. The use of a dashboard, to construct the workflow, supported by a validator along with dropdown menus, to specify the additional information concerning the cloud deployment, assists the users by minimizing possible mistakes. It also important to emphasize that no errors were seen during the usability tests. Moreover, by utilizing a few clicks the users can provide every information without mistakes. Not only does it reduce the complexity of executing a workflow through a cloud environment but also enables the users to spend less time using the system. As result, based on the values obtained in the tab:Workflow Execution times recorded from the system validation, the average time to complete this procedure is 773 seconds. It is important to mention, the longer the workflow, the more time is required.

We are able to observe every goal was achieved in our system. An accessible system, easy to learn and use regarding the workflow execution in the cloud. While at the same time being able to minimize the possible mistakes that users are susceptible when doing the procedure manually.

VII. Conclusion

To summarize, the procedure of wrapping EO tools into WPSs and deploying them to the cloud platform is time consuming, repetitive and requires a certain amount of knowledge when done manually. In the end, it places unnecessary burden in the user that can translate into avoidable mistakes, becoming cost heavy. Therefore, we aimed to develop continuous delivery pipelines that ease these burdens from the user.

We transformed the EO tools on a widely used standard, WPS, using templates. Subsequently, we used Docker to perform the encapsulation. We opted to use Dockerhub to store our images. We utilized WEFEOS for the data validation, preventing resource waste from errors. Using Ansible, we were able to create playbook to automate the provision and configuration of cloud instances. Additionally, we implemented a WPS management system from several catalogs to enable the future use of WPS already running in servers.

With the solution described and supported by the validation results, we can state that all proposed objectives of providing a high degree of automation were accomplished with success. Moreover, with the contribution to the OG engineering report our system definitely has a positive impact in the EO community.

VIII. Future Work

Our system is bound to the dashboard provided by the WPS-V-WEB. As a result, our system only supports sequential workflow executions. Further developing the dashboard and the system to provide parallel execution brings numerous advantages. The execution of workflows where the execution time is a constrain greatly benefit from using parallelism. Moreover, by introducing parallelism to the system, it enables the possibility to divide the execution of a WPS processing a dataset between several cloud instances while executing them at the same time.

Throughout the development of our system, we found the monetary costs are affected by numerous variables when deploying in the cloud. One of these variables is the hardware configuration, RAM and CPU. Choosing the right hardware configuration might reveal to be a slightly tricky. Either the resources are not enough, or too many resources were selected and not fully utilized. Additionally, the monetary cost for the hardware configuration varies from cloud provider to cloud provider and region to region. Another key factor that has an impact in the monetary cost is the outbound and inbound data transfer, additional charges depending on the data size and transmission time. With all these variables available, users might feel overwhelmed, resulting in not taking the best approaches. Therefore, it is important to enhance the system with a machine learning module. By taking into consideration these variables.

References

[1] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds," *CCGrid 2010 - 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing*, pp. 513–517, 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5493446

[2] R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, pp. 599–616, 2009. [Online]. Available: www.elsevier.com/locate/fgcs

[3] L. Zhu, L. Bass, and G. Champlin-Scharff, "DevOps and Its Practices," *IEEE Software*, vol. 33, no. 3, pp. 32–34, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7458765

[4] J. Wettinger, "Gathering Solutions and Providing APIs for their Orchestration to Implement Continuous Software Delivery," Master's thesis, Universität Stuttgart, 2017. [Online]. Available: https://elib.uni-stuttgart.de/handle/11682/9110

[5] D. Norman Lee Faus, "Packaging an Application," Red Hat Inc, Tech. Rep. US 9323519B2, 2016. [Online]. Available: https://patents.google.com/patent/US9323519B2/en

[6] E. C. Bailey, *Maximum RPM Taking the RPM Package Manager to the Lim-it*. Red Hat, Inc., 2000.

[7] M. J. Scheepers, "Virtualization and Containerization of Application Infrastructure: A Comparison," Tech. Rep., 2014. [Online]. Available: http://mmc.geofisica.unam.mx/acl/Herramientas/MaquinasVirtuales/VirtualizacionEnLinuxCon-Containers/539ae779eb69a.pdf

[8] J. Turnbull and C. Pahl, "Containerization and the PaaS Cloud," *Published by the IEEE Computer Society*, vol. 7, no. 11, pp. 24–31, 2015. [Online]. Available: https://www.computer.org/csdl/magazine/cd/2015/03/mcd2015030024/13rRUyoPSR7

[9] J. Turnbull, *The Docker Book*. Turnbull Press, 2014.

[10] C. Lueninghoener, "Getting Started with Configuration Management," p. 17, 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.193.9608&rep=rep1&type=pdf

[11] Y. Brikman, *Terraform Up and Running Writing Infrastructure as Code*. O'Reilly Media, 2017.

[12] Matthias Mueller and Benjamin Pross, "OGC WPS 2.0.2 Interface Standard," Open Geospatial Consortium, Tech. Rep. 14-065r2, 2015.

[13] B. Schaeffer, "Towards a Transactional Web Processing Service (WPS-T)," 2013.

[14] D. Rafael Ferreira Lopes, "Workflow Engine for Earth Observation Services," Master's thesis, Instituto Superior Técnico, 2018. [Online]. Available: https://fenix.tecnico.ulisboa.pt/downloadFile/563345090416386/79018-Diogo-Ferreira-Thesis.pdf