remote sensing

MDPI

Article

The VLab Framework: An Orchestrator Component to Support Data to Knowledge Transition

Mattia Santoro [1,*], Paolo Mazzetti [1] and Stefano Nativi [2]

[1] Institute of Atmospheric Pollution Research—National Research Council of Italy (CNR-IIA), 50019 Sesto Fiorentino, Italy; paolo.mazzetti@cnr.it

[2] Joint Research Centre, European Commission, 21027 Ispra, Italy; Stefano.NATIVI@ec.europa.eu

* Correspondence: mattia.santoro@cnr.it

Received: 5 May 2020; Accepted: 29 May 2020; Published: 2 June 2020

Abstract: Over the last decades, to better proceed towards global and local policy goals, there was an increasing demand for the scientific community to support decision-makers with the best available knowledge. Scientific modeling is key to enable the transition from data to knowledge, often requiring to process big datasets through complex physical or empirical (learning-based AI) models. Although cloud technologies provide valuable solutions for addressing several of the Big Earth Data challenges, model sharing is still a complex task. The usual approach of sharing models as services requires maintaining a scalable infrastructure which is often a very high barrier for potential model providers. This paper describes the Virtual Earth Laboratory (VLab), a software framework orchestrating data and model access to implement scientific processes for knowledge generation. The VLab lowers the entry barriers for both developers and users. It adopts mature containerization technologies to access models as source code and to rebuild the required software environment to run them on any supported cloud. This makes VLab fitting in the multi-cloud landscape, which is going to characterize the Big Earth Data analytics domain in the next years. The VLab functionalities are accessible through APIs, enabling developers to create new applications tailored to end-users.

Keywords: virtual earth laboratory; orchestration; data to knowledge; environmental modeling; interoperability; geospatial technology

1. Introduction

Humankind is called to face a set of global challenges that are crucial for the sustainability of our planet and for the future development of human society. Therefore, international organizations defined a list of policy goals to be achieved in a defined time framework. The United Nations (UN) defined 17 Sustainable Development Goals (SDGs) [1] along with an implementation agenda supported by other relevant international initiatives and programmes, including the Conference of Parties 2015 on Climate (COP21) [2] and the Sendai Framework for Disasters Risk Reduction [3].

To measure these policy achievements, specific targets have been recognized to be accomplished through an informed decision-making process. This process aims at evaluating the outcomes of policy implementation actions with respect to the fulfillment of the targets. The entire society is asked to contribute to this process; in particular, the scientific community is asked to provide decision-makers with the necessary knowledge for a science-informed action-taking.

We live in the era of Big Data [4] and, in particular, Earth Observation (EO) generates tens of terabytes daily. The observation of our planet is implemented by using remote sensing (e.g., sensors flying on satellites, drones, airplanes, etc.), in-situ measurements (e.g., meteo stations, crowdsourcing, etc.), and socio-economic statistical data, (e.g., social networks and official statistical data). This data deluge offers a great opportunity for extracting the required knowledge from the large amount of data

produced every day, including the recognition of changes over time. The recent new spring of AI has provided a valuable instrument to analyze a large amount of Earth observations and generate insights. In such an innovative context, a new scientific discipline was introduced to understand the impacts and interrelationships between humans as a society and natural Earth system processes: the Big Earth Data science [5].

Some recent research and innovation projects, funded by the European Commission—H2020 ECOPOTENTIAL [6,7], H2020 ERA-PLANET [8,9]—recognized a general process that can be used for providing decision-makers with the knowledge required for science-informed action. The process, illustrated in and described in [10], consists of adopting a step-by-step approach for implementing a data-to-knowledge transition. *Observed data* (i.e., EO and socio-economic data) is processed to generate a set of *Essential Variables* (EVs) [11–14], characterizing the relevant Earth subsystems which are then further elaborated to produce one or more *Indicators* that summarize the state of investigated *Phenomenon* characterizing our planet. The carried-out *Indicators* summarize the knowledge provided to decision-makers; they can finally be used for comparisons against one or more policy targets, making it possible to assess and evaluate the fulfillment of a specific policy goal.

For *EVs* and *Indicators* generation, it is possible to adopt different procedures and methods, as shown by the scheme depicted in Figure 1. An information framework is required, stemming from the Big Earth Data science [5], to allows us to run and compare diverse scientific models in order to recognize the best actionable knowledge. This Big Earth Data science framework applies a (multi-organizational) collaborative approach implementing a flexible and open digital platform that deals with interoperability challenges [15–17].

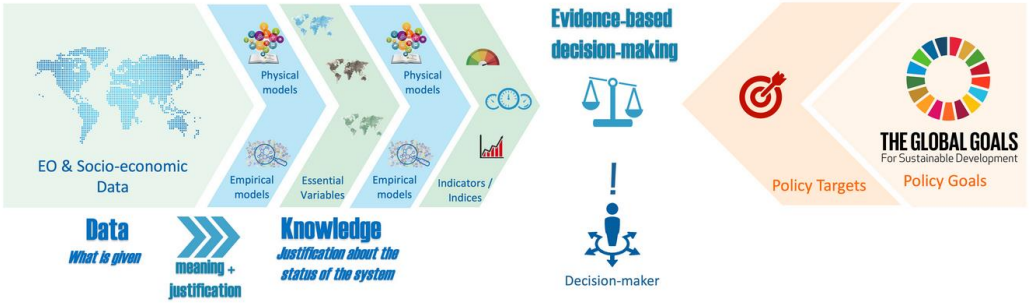

Figure 1. Science-informed decision-making process.

The next section discusses the main steps and approaches characterizing the development process to support decision-makers. The challenges to be addressed by the digital framework are described in Section 3. An implementation of such a digital framework, i.e., VLab, is introduced in Section 4. VLab framework experimentation is discussed in Section 5. Finally, Section 6 discusses the presented framework, and Section 7 covers conclusions and future work.

2. Developing Applications for Policy Making

According to the normative decision theory [18], a decision-making process includes the following steps: (a) formulating a policy-related question requiring a choice; (b) identifying the available choice options; (c) deriving the prospects for each option; (d) evaluating preferences over prospects; (e) select the most preferable, i.e., the "best" choice. In step (c), the use of the best available scientific knowledge makes a science-informed decision-making process. In the global change domain, a science-informed decision-making process is asked to implement such a stepwise methodology in keeping with three important principles:

a. To adopt a policy-driven approach formulating questions pertaining to policy-related objectives—see in particular steps (a) and (e);

b. To implement the steps through a co-design and collaborative approach—in particular, to identify the relevant outcome aspects to describe in step (c);
c. To leverage a multi-organizational (scalable) infrastructure, i.e., a multi-platform—see in particular, steps (b), (c), and (d)—to have access to the best available knowledge (data and models) and capabilities (storage, computing) wherever it is offered.

While the "client" is a policy-maker or a policy decision-maker, the participation of the stakeholders (i.e., information providers and/or users) in designing a science-informed process is essential to make it effective and usable by the end-user [19]. Besides, considering the complexity of policy-support applications, engaging disparate systems and software tools is highly recommendable, and it also improves the process of sustainability.

In the case of a science-informed application for decision-makers, the co-design should first define the reference framework of values, usually expressed by a shared policy goal (e.g., a Sustainable Development Goal or European policy, like the Common Agricultural Policy–CAP), and then recognize an official measure of preference expressing the preference about the state of affairs, for example, a simple or composite indicator, like an SDG or CAP indicator. Once the measure of preference is identified, it is necessary to understand the digital resources (i.e., data, information, models, and tools) that are necessary to implement it in a collaborative way. Finally, the needed capacity infrastructure(s) and software platform(s) must be identified in order to provide a scalable and replicable solution. The approaches and the main procedural steps are shown in Figure 2.

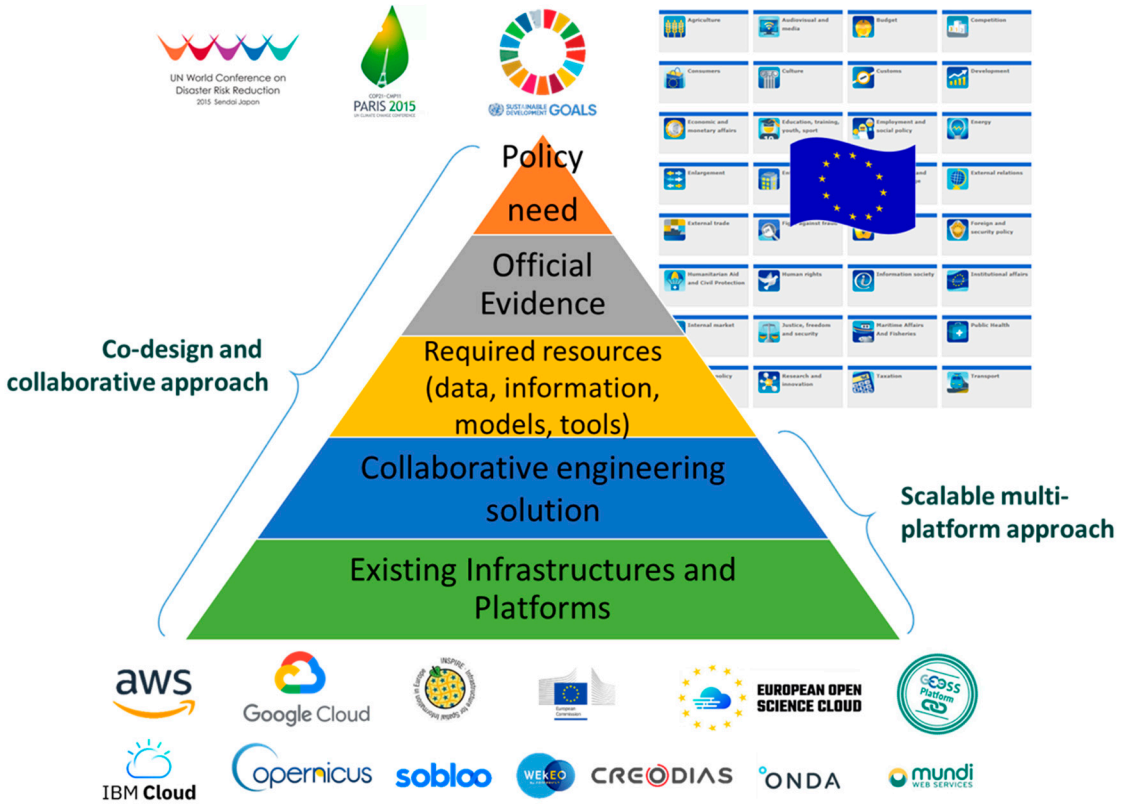

Figure 2. Approaches and components for designing and implementing applications for policy-makers.

To facilitate the co-design and co-creation of a science-informed process to support policy-makers, a Big Earth Data science framework is needed for the publication, sharing, and execution of scientific business processes, in a multiplatform environment. In this context, a business process is "*a defined set of business activities that represent the steps required to achieve a business objective*" [20]. While we define a *scientific* business process as *a business process whose objective is the delivering of science-based*

information, i.e., the related (business) activities must be scientifically sound. Commonly, scientific business process activities include the access to scientific data through authoritative and trusted sources, and the execution of scientific models based on acknowledged scientific theories.

3. Orchestration Framework and its Challenges: the VLab Implementation

Figures 1 and 2 show a digital framework for the co-design and collaborative development of a science-informed process to support decision making. A key component of such a digital framework is a software orchestrator. This must implement, in an automatic way, the configuration, coordination, and management of online-accessible systems and technologies. Virtual Earth Laboratory (VLab) is a technology that implements such an orchestrator, enabling the implementation of the co-design and co-development digital framework. VLab aims to minimize the interoperability requirements for sharing and reusing existing scientific models and data that are going to contribute to the implementation of the knowledge generation process, depicted in Figure 1. Working on a cloud environment; VLab supports two macro-scenarios:

- **Scientific resource publishing and sharing on-line**: a scientific resource provider is the owner of a scientific resource (e.g., a dataset or a model) that has already been methodically validated (to some degree) and that she/he would like to publish and share on-line. Through VLab, the resource provider can publish the resource, making it privately or publicly available to existing or new scientific processes.
- **Application development**: a software developer needs to realize an application making use of online available processes and resources, which were already validated from a methodological point of view. Through the VLab APIs, the software developer can programmatically access the online available resources and execute the processes.

By supporting the introduced macro-scenarios, VLab provides a software environment that aims at lowering users' entry barriers. Users interact with VLab, directly or indirectly, playing different roles:

- **Model Providers**: they act as a resource provider;
- **Application Developers**: they are the intermediate user who develops desktop or mobile applications for end-users, accessing the scientific resources and processes made available by VLab.
- **End-users**: they interact indirectly with VLab through an application created by an Application Developer, examples of end-users are: policy-makers, decision-makers, and citizens.

To facilitate user activities, VLab must address several barriers stemming from the complexities characterizing the resources domain and the computing environment to be utilized:

- **Geospatial Resources Harmonization**: geospatial resources have their own specificities related to their spatial and temporal components. Traditionally, several modeling and standard protocols exist for geospatial resources sharing, e.g., metadata and data models, service interfaces, and APIs. They addressed the need of handling a great variety of resources. This originates at a certain complexity level for working with geospatial information requiring specific expertise. Often, domain scientists (including modelers) but also application developers, do not have this expertise. Typically, a domain expert or application developer works with a limited set of protocols belonging to its realm, i.e., metadata and data formats, coordinate reference systems, service interfaces, APIs.
- **Software Environment Harmonization**: research modelers commonly work within their preferred software environments or simulation frameworks to ensure backward compatibility, access to customized libraries, performance achievements, personal digital skills. Often, it is difficult and/or time-consuming to try pushing a change in the utilized software environment or simulation framework.
- **Multiplatform Support for Scalability**: more and more often, a scientific model requires big data (i.e., big size datasets and/or a large number of small but heterogeneous datasets) and

heavy computing capabilities. Besides, even if a single run does not require such capabilities, the opportunity to invoke multiple parallel runs would require that. The utilization of a specific HPC or cloud infrastructure would not be a wise solution for several reasons, including the need to support different cost schemes (e.g., availability of cloud credits for users, or free credits for research/education projects), different capabilities (e.g., type of virtual machines available in a cloud), and different privacy and property right agreements.

These Big Data and analytics barriers stress the importance for VLab and similar solutions to manage diversity in a flexible and evolvable way. The main challenges include:

- **Large Data Variety**: Earth observation and Earth science products are extremely heterogenous. They differ in terms of metadata and data format, coordinate reference system, resolution. They also differ by the data model and semantic content, and they are typically served through systems adopting different service or programming interfaces. A tool like VLab must be able to access as many datasets as possible implementing the necessary interoperability agreements without imposing constraints to providers (as much as possible).
- **Large Model Variety**: Earth science scientific models are developed in many different programming environments (e.g., Python, Java, R) or simulation frameworks (e.g., NetLogo, Simulink). A tool like the VLab must be able to run models without imposing constraints on providers (as much as possible).
- **Big Data Volume**: Earth observation and Earth science products are now available in a great amount. Thanks to high-resolution sensors and new platforms, innovative solutions for sensor networks (IoT), social networks, passive crowdsourcing (e.g., from mobile phones), a lot of data is available to deliver relevant insights for decision-makers. A tool like VLab must be able to make this large number of datasets available to running models.
- **Fast Model Processing**: Scientific models encompass a wide scale of complexity. They range from simple models generating indicators to very complex simulation models. A tool like VLab must be able to run all (or at least a wide subset of them) with good performances allocating the correct resources.
- **Multiplatform**: Many different solutions for executing software code are currently available. Their services differ in terms of technical capabilities, resource availability, costs, and privacy/property rights conditions. A tool like VLab must be able to execute models on different platforms and, when possible, choose the right platform considering many aspects including user preference, cost, resource availability (e.g., minimization of data transfer), and privacy constraint.

As far as *Data Variety*, VLab relies on a brokering approach [21,22] to enable the discovery and use of available Earth observation and Earth science products (Section 5.1). In order to support multiple programming languages and environments for models (*Model Variety*), VLab relies on containerization (or container-based virtualization) [23]. Model developers provide in the VLab convention files (see Section 4.5) the container image to be utilized for running the model; this guarantees that not only the utilized language is supported, but also that the image incorporates all the required libraries. The use of containers is further clarified in Sections 4.3 and 4.4. *Big Data Volume* and *Fast Model Processing* challenges are addressed by means of cloud technologies; in fact, VLab is natively designed to run in cloud environments, which provide the necessary scalability as far as computing and storage resources are concerned. Finally, to support the use of multiple cloud platforms (*Multiplatform*), the VLab architecture identifies which cloud functionalities are needed for VLab execution (see Section 4.3) and how these are orchestrated to implement the VLab business logic (see Section 4.2). This allows us to support new cloud platforms by identifying the appropriate cloud platform-specific services and implementing the appropriate component(s) to connect to such services.

4. VLab Architecture

VLab architecture is defined applying the viewpoint modeling approach, i.e., information model (information view); logical functional components (computational view); deployment configuration (engineering view); utilized technologies (technology view).

4.1. VLab Information Model

4.1.1. Data to Knowledge for Policy Ontology

The reference context for the VLab data model is provided by the Data to Knowledge for Policy (D2K4P) ontology, defined in the GEOEssential Project of the ERA-PLANET [9]. This abstract ontology defines the major concepts and their relations to support the transition from Data to Knowledge for the assessment of policy goals; i.e., the D2K4P provides a knowledge representation for relating the different information artifacts (dataset, models, etc.). Figure 3 depicts the D2K4P, the main concepts are reported below:

- **Data/Observation**: a physical parameter which can be directly observed with proper instruments;
- **Essential Variable**: a physical parameter which is necessary to describe the Earth system status;
- **Indicator**: a derived parameter summarizing the status of the system under consideration (it can be an Indicator or, more generally, an Index);
- **Policy Goal**: the desired outcome or what is to be achieved by implementing a policy
- **EV Generation Model**: a science-based process to generate Essential Variables from Observables (it includes both physical and statistical (including machine learning and AI) models since both have scientific soundness if correctly adopted.)
- **Indicator Generation Model**: a process to evaluate and summarize Essential Variables representing the system status as an Indicator (or an Index).

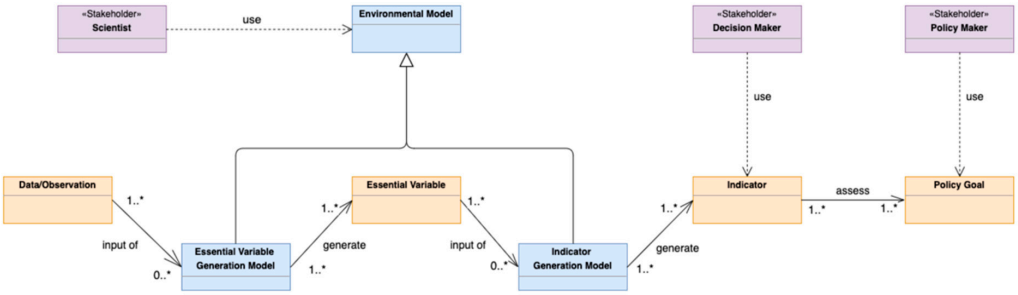

Figure 3. Data To Knowledge for Policy Conceptual Model.

Figure 3 captures possible entry-points for different stakeholder types:

- **Decision Makers** are typically interested in the generated Indicators in order to evaluate possible policy choices to be adopted;
- **Scientists** are interested in developing and/or searching for Environmental Models to study how to model Earth system for the generation of Essential Variables and/or Indicators.

4.1.2. Model Execution Information Model

The VLab information model stems from the D2K4P ontology in Figure 3, adapting it to support the model execution according to the GEO Model Web framework [15]. The main concepts of VLab execution data model are depicted in Figure 4. Resources are represented using different colors to indicate their relevant properties.

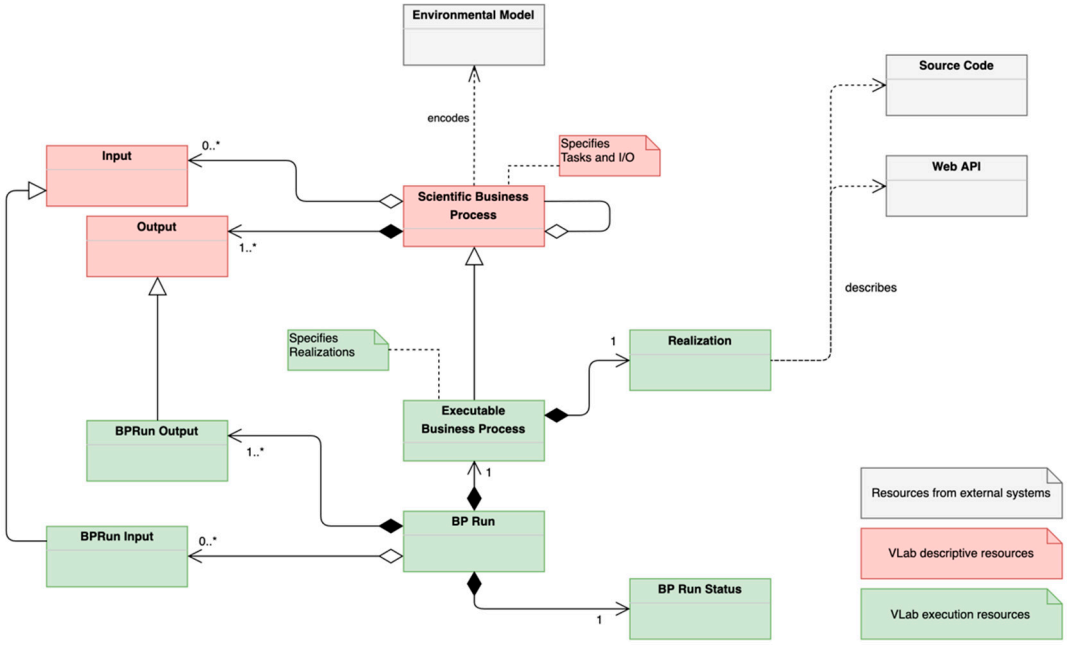

Figure 4. VLab Information Model.

White classes represent the resources that are not part of the VLab system such as an *Environmental Model*, a *Source Code* repository, or *Web API*. These resources are described/referenced by VLab (environmental models) or utilized (source code and Web API) for model execution.

Red classes represent formalized descriptions of relevant resources for model execution. The *Scientific Business Process* concept here is characterized by the set of tasks it is composed of and by its *Input*(s) and *Output*(s). It is worth to note that at this level (execution), the distinction between *Data/Observation* and *Essential Variable* is no longer relevant; in fact, the data model is simply capturing the fact that a *Scientific Business Process* needs a number of *Input*(s) and *Output*(s). Instead, at this level, it is important to capture execution-related information about these objects, such as type (if a single dataset or array of datasets), obligation (if mandatory or optional), and other information detailed in Section 4.5.

Finally, green classes represent the resources required for actual model execution. The *Executable Business Process* class represents an executable version of a *Scientific Business Process*; i.e., a *Scientific Business Process* associated with a *Realization*. The *Realization* class describes an external resource which realizes (implements) a *Scientific Business Process*. *Realizations* can reference a *Source Code* (e.g., hosted on a code-sharing repository such as GitHub [24]) or a *Web API* that can be used to execute the model. The *BP Run* class represents an actual execution of an *Executable Business Process*. Each execution is characterized by (i) the *Executable Business Process* being executed, (ii) the *BP Run Status*, (iii) the utilized *BPRun Input*(s), and (iv) the generated *BPRun Output*(s).

These last two classes extend *Input* and *Output* classes by providing *Realization*-specific information. In particular, they provide additional information about how a specific implementation (described by a *Realization*) expects to be provided with inputs and will provide outputs, e.g., in which folder a specific model implementation expects to find input data.

4.2. VLab Components

Figure 5 depicts the main VLab components grouped in packages according to their high-level functionalities. The following sections describe the functionalities of the components in each package.

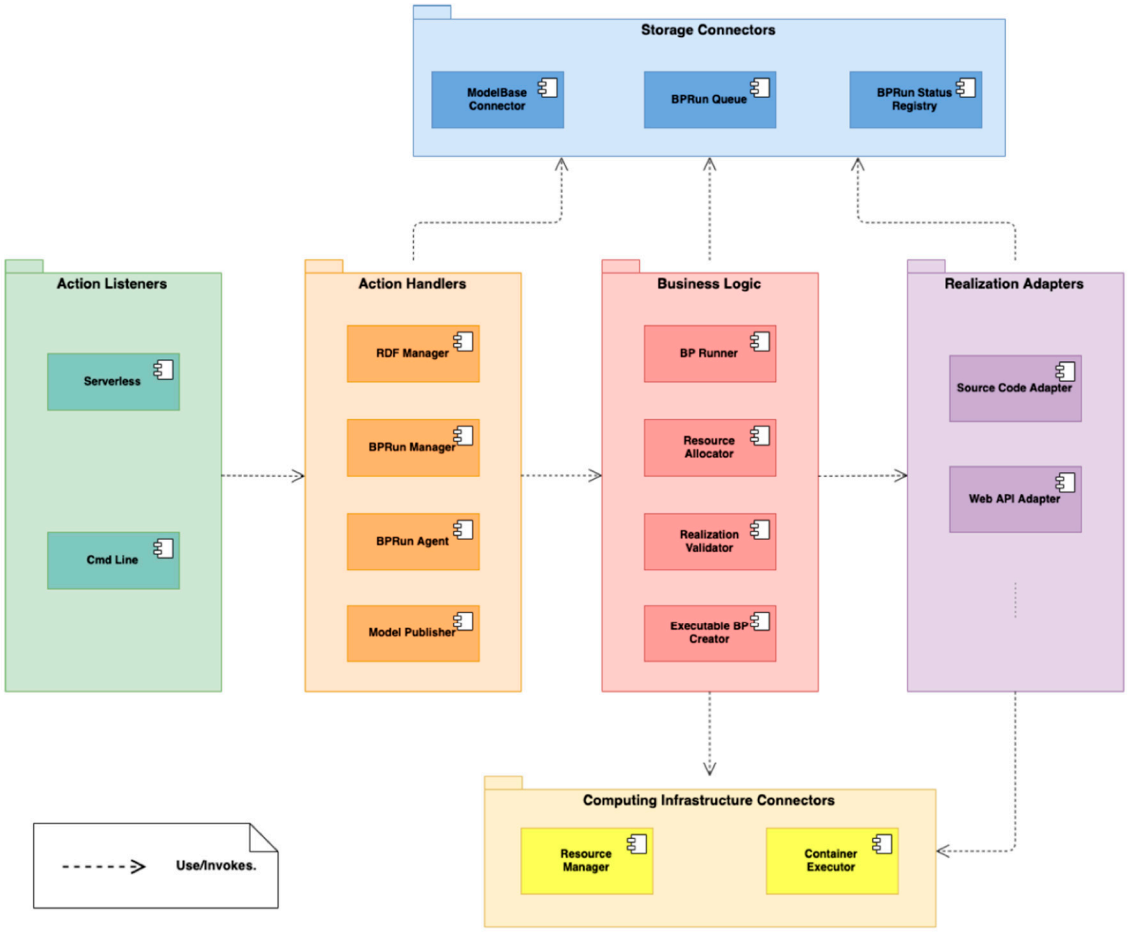

Figure 5. VLab Software Component Diagram.

4.2.1. Storage Connectors Package

Storage Connectors components are used to read/write VLab Information Model objects to persistent storage services. Each component provides different implementations to support different storage services according to the underlying cloud platform and/or deployment configuration. *BPRun Status Registry* is in charge of serializing/deserializing *BPRun Status* objects and read/write them to the configured storage system. The *BPRun Queue* provides a simple FIFO (First-In/First-Out) queue interface for *BPRun* objects. Finally, the *ModelBase Connector* implements read/write functionalities of *Scientific Business Process* instances to persistent storage.

4.2.2. Action Listeners Package

This package contains components that listen for action requests and propagate them to the proper component in the Action Handler package. The *Serverless* component implements the required interfaces for receiving actions from the cloud Serverless service (see Section 4.3). The *Cmd Line* component allows running VLab actions from the command line.

4.2.3. Action Handler Package

These components implement the proper orchestration of *Storage Connectors* and *Business Logic* components to fulfill the requested action.

The *RDF Manager* is executed to get/add a concept instance from/to the RDF triple-store. When invoked, this component parses the received inputs. If the parsing fails due to a bad request, an error

message is returned. If the parsing succeeds, a proper message is created and submitted to the *RDF Connector*.

The *BPRun Manager* is invoked when a new run is requested; it creates a new *BPRun* based on the user's request (a list of *Inputs* and the identifier of an *Executable Business Process*) and requests to add it to the runs' queue by invoking the *BPRun Queue* component. This component is also in charge of fetching the status of a given *BPRun*, reading the identifier of the *BPRun*, and requesting its status to *BPRun Status Registry*.

The *BPRun Agent* is in charge of trying to execute the next run queue. When triggered, this component reads the next *BPRun* to be executed from the *BPRun Queue* and invokes the *BP Runner* for its execution. If the *BP Runner* succeeds and triggers the execution, the *BPRun Agent* removes the *BPRun* from the runs' queue; otherwise, a resource augmentation request is requested to the *Resource Allocator*.

The *Model Publisher* component is invoked to create a new *Executable Business Process* from a *Realization*. First, it utilizes the *Realization Validator* to validate the provided *Realization*. If the validation fails, a message is created reporting the validation exception. If the validation succeeds, the *Model Publisher* requests the *Executable BP Creator* to generate an *Executable Business Process* associated with the given *Realization* and returns a success message.

4.2.4. Business Logic Package

The Business Logic package provides the components which execute the actual business logic to implement the core functionalities of the VLab (run a model, validate a realization, etc.).

The *BP Runner* component encapsulates the business logic to trigger the actual execution of a *BPRun*. First, this component requests the proper *Realization Adapter* to the *Realization Adapters* package; then the *BP Runner* executes the needed initialization steps on the *Realization Adapter* (e.g., sets the inputs defined in the *BPRun*), invokes the *Realization Adapter* method to reserve the resources needed for the execution and, if the reservation process succeeds, triggers the *Realization Adapter* execution. If the *Realization Adapter* fails in reserving the resources, the *BP Runner* execution exits providing the proper message to the caller.

The *Resource Allocator* is in charge of implementing the request of additional resources to the underlying infrastructure (e.g., more computing units) for the execution of a *BPRun*. When a resource augmentation is requested for a *BPRun*, the proper *Realization Adapter* is retrieved; then, the description of the necessary resources for the model execution is obtained from the *Realization Adapter*; finally, an augmentation request for the underlying infrastructure is created and submitted through the *Resource Manager*.

4.2.5. Realization Adapters Package

All components in this package implement the same interface which can be used to (i) request if the component supports a Realization, (ii) set the list of Inputs for the model execution, (iii) describe the necessary resources for the model execution, (iv) execute the model, and (v) clean temporary resources associated with an execution.

Each component (*Realization Adapter*) is associated with a *Realization* type (e.g., Web API, Git repository, etc.). The common interface is implemented by each adapter by interacting with the resource described by the *Realization* it is associated with, e.g., a Git Source Code Adapter is available to connect to a Git repository.

The *Web API Adapter* implements the required mediation functionalities in order to create and submit a proper execution request to the API URL publishing the model processing service. The service executes the computation and returns the result to the *Web API Adapter*. This component is defined in the VLab architecture for completeness, but it was not implemented yet.

The execution of models published on source code sharing repositories is supported by the *Source Code Adapter*. This component is detailed in Figure 6. It implements the common interface by delegating to two components: *Repository Connector* and *Source Code Executor*.

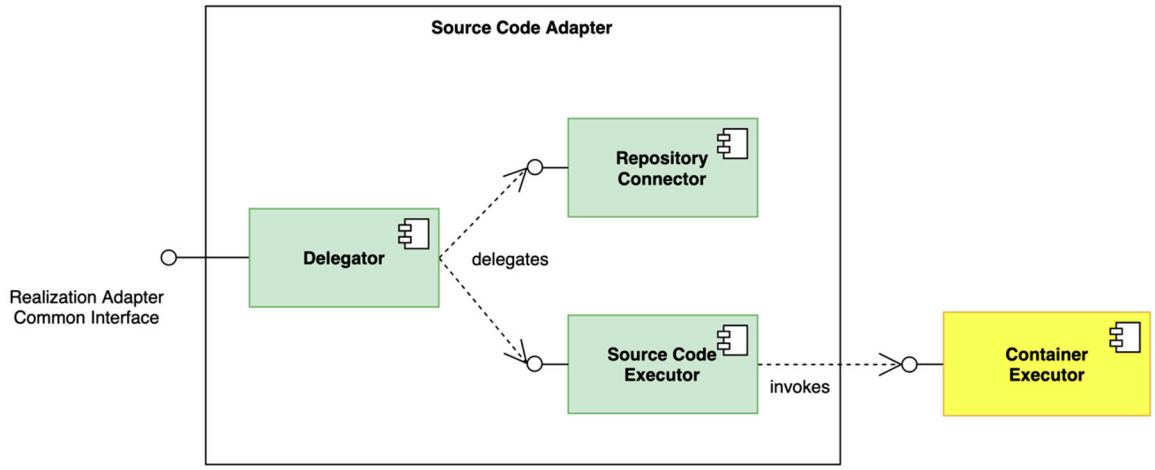

Figure 6. The Source Code Adapter.

The *Repository Connector* is in charge of connecting to the code-sharing repository, fetching its content to the local environment and validating it. Repository content is valid if the *VLab Conventions* files are present. In fact, to be able to execute arbitrary models coded in arbitrary programming languages, the VLab needs some information about the model implementation (e.g., the input set, which scripts to execute, etc.). This information is provided by the *VLab Conventions* files, which are detailed in Section 4.5.

The information provided by the convention files is utilized by the *Source Code Executor*. This defines the actions required to execute the code and requests the *Container Executor* to perform such actions. In brief, the approach is to create a containerized application to be executed on computing infrastructure that supports the containerization technology (details about the utilized containerization technology are provided in Section 4.4). The executor defines the following actions:

- Launch a containerized application based on the container image from the convention files;
- Copies the last version of the source code to the running container;
- Ingests the provided inputs to the running container;
- Executes the source code on the running container;
- Saves the generated output.

4.2.6. Computing Infrastructure Connectors Package

This package provides the components to interact with the computing infrastructure provided by the underlying cloud platform. Both the *Resource Manager* and the *Container Executor* work as clients to the computing infrastructure, implementing the client-side of the cloud platform APIs.

The *Resource Manager* is in charge of requesting/dismissing computing nodes. The *Container Executor* implements the APIs required to manage the life-cycle of a containerized application on the computing infrastructure.

4.3. VLab Deployment Configuration

Figure 7 depicts how VLab can be deployed on a cloud infrastructure. Different cloud services, i.e., services provided by the underlying cloud infrastructure, are needed either to ensure the system scalability/reliability or to provide infrastructure functionalities required by VLab. In the following section, the utilized cloud services are briefly introduced; then their use in VLab deployment is described.

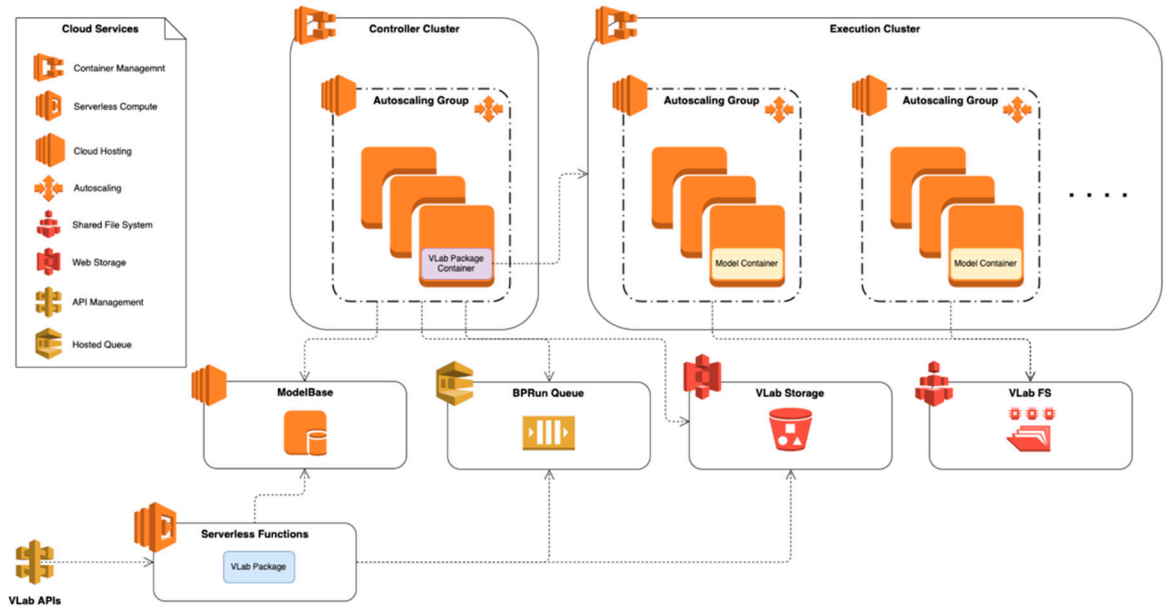

Figure 7. VLab Cloud Deployment.

4.3.1. Cloud Services

Cloud Hosting

A service that allows us to run different virtual computing environments, known as instances. It is possible to run instances with various configurations of CPU, memory, network, etc. This service provides also a set of pre-configured instances with various combinations of operating systems and installed software.

Auto Scaling

A service that manages a cluster of instances (auto-scaling group) by automatically adding or removing instances according to a set of scaling rules.

Container Management

A container management service that allows us to create/run/stop containerized applications on a cluster by utilizing APIs or SDKs.

Shared File System

A shared file system service. This can be used to create a file system and mount it on different instances at the same time. This way, applications running on an instance have access to the shared file system as if it were a normal local drive.

Web Storage

This service provides the possibility to store and retrieve a large amount of data in a web-accessible storage system. It requires us to use its own APIs for storing and retrieving data.

Hosted Queue

This is a service that provides a cloud-based hosting of message queues. It is possible to create a queue and interact with it from different distributed components.

Serverless Compute

A serverless compute service. Such a service allows us to define a function in one of its supported programming languages (Java, Python, etc.). Each function can be configured with different triggers for its execution (events, periodic tasks, etc.). When execution is triggered, the function code is executed on cloud-hosted capacities without requesting the provision or management of servers.

API Management

A service that facilitates the creation and publication of Web APIs. It allows us to define an API and, for each path and HTTP method, the API Management service allows us to define an action to be performed, e.g., trigger a Serverless function.

4.3.2. Deployment

The persistent storages utilized in the configuration depicted in are:

- *ModelBase*: a database where instances of *Scientific/Executable Business Process* are stored; this can be deployed on a dedicated compute node or can be provided as a service by the cloud platform; current deployment utilizes the dedicated single node option;
- *VLab Storage*: this is provided by the cloud Web Storage service and is used to store executed *BPRuns*, model execution outputs, and *BP Run Statuses*;
- *BPRun Queue*: the queue where the requested *BPRuns* are stored waiting for execution;
- *VLab FS*: a file system provided the cloud *Shared File System* service; this is utilized by the running models to read inputs and write temporary files and outputs.

The VLab software package is deployed both as the function code of the cloud *Serverless Compute* service and as a containerized application on the cloud *Container Management* service. Both deployments read/write from the same persistent storages, ensuring that executions are consistent (i.e., they access the same data).

The first deployment (serverless function) is utilized to implement the VLab APIs. This is possible by connecting the definitions of the VLab APIs in the cloud *API Management* service to the serverless function. This way, high scalability and reliability of VLab APIs is ensured by the undelaying cloud *Serverless Compute* service. Every incoming request is intercepted by the *API Management* service which handles it as far as HTTP level is concerned. Then the business logic is executed by the packaged VLab software, which is instantiated and executed on-the-fly by the *Serverless Compute* service.

The containerized deployment of VLab is utilized for the actual execution of the models. The *Container Management* service environment is configured with two clusters: Controller and Execution clusters. The first one is where the VLab containerized application is executed. The second one is where the model executions take place.

On the Controller cluster, the execution of the VLab software is configured to be regularly scheduled (e.g., once a minute), triggering the *BPRun Agent* via command-line. That is, the VLab pulls new *BPRuns* from the *BPRun Queue* every minute. When a new *BPRun* is found, the VLab removes the BPRun from the *BPRun Queue*, stores it to the *VLab Storage,* and ingests the model source code and the defined inputs to the *VLab FS*; then VLab runs the containerized model utilizing available resources (i.e., computing nodes) in the Execution cluster. When the model completes, its outputs are moved from the *VLab FS* and are stored in the *VLab Storage*. The *VLab Storage* is utilized also for storing and retrieving run statuses throughout all the execution time.

The two clusters have different configurations. The Controller cluster requires few computing resources, in fact, the VLab package is quite lightweight and does not execute computationally intensive tasks. However, an Autoscaling Group is defined also for this cluster. This ensures that if a node fails, the system will keep working (reliability), and that in case of exceptionally high work-load the system will add the necessary capacity (scalability) to run multiple VLab package instances in parallel. The Execution Cluster is where heavy computation takes place. This is configured with a set of Autoscaling Groups. Each group is defined to run instances with different CPU and Memory configurations to be able to match the requirements of the model to be executed. In addition, each computing node that is executed in this cluster mounts the VLab shared file system during its start-up procedure. This way, models can access input data and write output data utilizing the usual programming language functionalities, i.e., models are not aware of being executed in the VLab. Besides, the use of a shared file system service allows the ingestion of very large data without having to manage the disk space provision on each computing node.

4.4. VLab Technology

The VLab software is mainly developed in Java [25], with the exception of few scripts in Python [26] and Javascript [27] utilized in the definition of serverless functions for very simple actions.

As far as cloud technologies, VLab was deployed on Amazon Web Services (AWS) [28] and OpenStack [29]/Kubernetes [30] environments. For each cloud service described in Section 4.3, Table 1 lists which cloud-specific technology was utilized to fulfill VLab deployment requirements for deploying VLab. In the utilized OpenStack deployment environment, not all required cloud services were available; however, it was possible to complete the deployment by utilizing the corresponding AWS services to provide the required functionalities. Ongoing experimentations are investigating possible solutions that implement the required cloud services in OpenStack environments.

The VLab provides RESTful APIs for the development of applications on top of it. The APIs are documented utilizing swagger technology; the APIs documentation is available online [31].

The containerization technology that is used is the widely-adopted Docker [32].

Table 1. Cloud Services utilized by VLab deployments for different cloud environments.

Cloud Service \ Cloud Environment	Amazon Web Services	OpenStack/Kubernetes
Cloud Hosting	Elastic Compute Cloud [33]	OpenStack Compute (nova) [34]
Auto Scaling	Auto Scaling Group [35]	OpenStack Heat [36]
Container Management	Elastic Container Service [37]	Kubernetes [30]
Shared File System	Elastic File System [38]	Network file System [39] + Kubernetes Persistent Volume [40]
Web Storage	Simple Storage Service [41]	MinIO [42]
Hosted Queue	Simple Queue Service [43]	KubeMQ [44]
Serverless Compute	Lambda [45]	-
API Management	API Gateway [46]	-

4.5. Implementation

4.5.1. Scientific/Executable Business Processes

Instances of the *Scientific Business Process* are represented utilizing an extended version of the BPMN (Business Process Model and Notation) standard [20]. The extension allows us to annotate Data objects of a BPMN document with a set of attributes required by VLab to characterize Inputs and Outputs; the main attributes are listed in Table 2.

Table 2. Attributes of VLab BPMN Extension for Input and Output Objects.

Attribute Name	Type	Allowed Values	Description
id	String	any	The identifier of the input
name	String	any	The name of the input (human-readable)
description	String	any	The description of the input (human-readable)
obligation	Boolean	true false	True if the input is mandatory, false otherwise
hasDefault	Boolean	true false	True if the input is mandatory, false otherwise
inputType	String	individual array	This property defines if this input is a single file (individual) or a list of files (array)
valueSchema	String	url bbox sat_product number_parameter string_parameter	This property defines the schema of the value, i.e., how the value must be interpreted by VLab. When the value schema is *url*, the value is expected to be valid 'ready to use' URL. When the value schema is *bbox*, the value is expected to be a bounding box in the form *west, south, east, north*. The schema *sat_product* is expected to be a valid identifier of a satellite product. Presently, supported satellite products include: Sentinel. The schema *number_parameter* is expected to be a number. The schema *string_parameter* is expected to be a string.

4.5.2. VLab Conventions

As introduced in Section 4.2, when executing models from source code, VLab requires some information about the source code execution. The required information includes:

- Environment to execute the model, this includes:
  - The Docker image must be used to run the model;
  - The resources required to execute the model (i.e., memory and/or cpu);
  - How to launch the model (i.e., the command to execute to trigger the model execution);
- Scripts required by the model; VLab will copy all required scripts to the execution environment;
- Realization-specific Input/Output description: information about where (i.e., in which folder) the model expects to find its input(s) and where it stores the output(s) of the computation; besides it is also possible to specify here default input values for the computation.

When porting an existing model to VLab, the above information is expected to be found in the source code repository for VLab to consume it.

4.5.3. ModelBase

Instances of *Scientific/Executable Business Process* are stored in a ModelBase. The current implementation utilizes an RDF triple-store for storing simple metadata about each *Scientific/Executable Business Process*, including title, abstract, developer, developer organization, and a link to the BPMN document. The choice of utilizing an RDF triple-store stems from the fact that in future developments, it might be necessary to utilize semantic web technologies for linking this ModelBase to other knowledge bodies, e.g., Essential Variables, Policy Indicators/Targets, etc.

4.5.4. Model Chaining

The VLab Data model is general enough to cope with model chaining (workflows) as well. In fact, a *Scientific Business Process* can be composed of other *Scientific Business Processes* and, in turn, an *Executable Business Process* can be composed of *Executable Business Processes*, generating a workflow. In this context, workflow is intended as "the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action according to a set of procedural rules" [47]. The present implementation of VLab does not include the possibility to execute such artifacts. This enhancement is part of scheduled next developments.

5. Experimentation

This section describes the results of VLab experimentation. In particular, Section 5.2 describes the experience and feedbacks from modelers who tried to publish their model on VLab; in Section 5.3, some web applications developed utilizing VLab are described; finally, Section 5.4 describes the results of testing the deployment of VLab on multiple clouds.

5.1. High-level Overview

Figure 8 depicts a high-level overview of the VLab experimentation framework. In a nutshell, VLab must be able to retrieve the model source code, transfer it to a computing platform, ingest the required input data, execute the model on the computing platform, and collect the generated output data. Therefore, VLab must be able to utilize different types of assets, in particular: data, source code and computing platforms. These are provided by external systems, which make their content available on the web. Data systems provide data discovery and access functionalities, including Earth observation (remote and in situ) and socio-economic data; source code sharing systems allow the retrieval of model source code; computing platforms provide the necessary functionalities to request and instantiate computational resources.

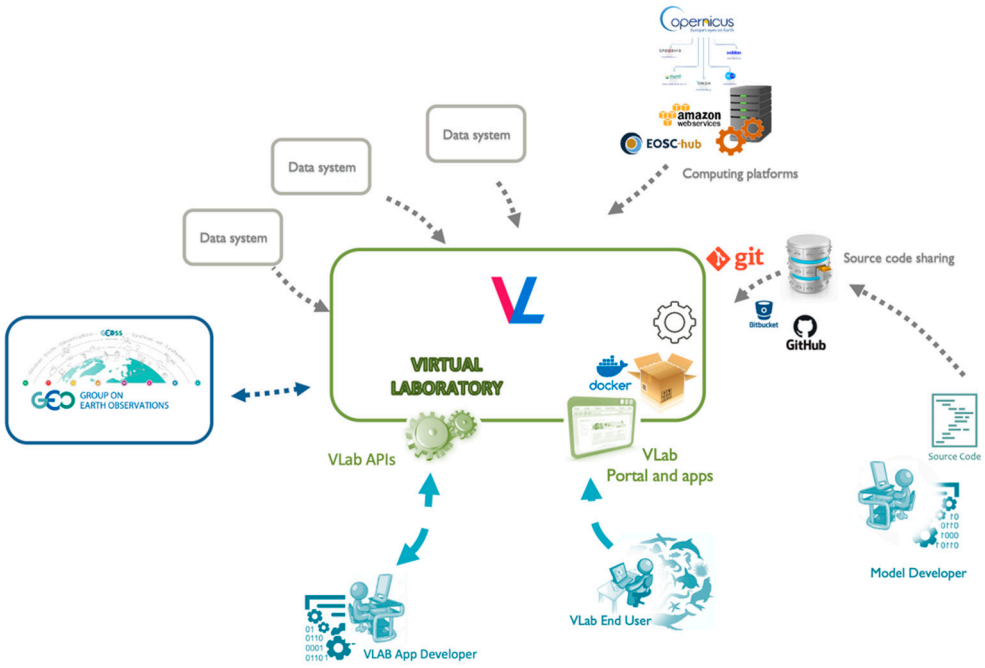

Figure 8. Conceptual Framework of VLab Experimentation.

As far as data, VLab must be able to utilize available data from existing (and possibly future) data systems. The connection to data systems is obtained by utilizing brokering technology, already described in other manuscripts [21,22], such as, e.g., the GEO Discovery and Access Broker [48].

5.2. Model Publication on VLab

In order to publish a model on the VLab platform, modelers are required to:

1. Share the model source code on a code-sharing platform (e.g., GitHub);
2. Generate a Docker image for model execution;
3. Specify VLab conventions files in the source code repository.

This procedure was tested in the context of the ECOPOTENTIAL [6] and GEOEssential [49] H2020 projects and online documentation was published [50]. Both webinars and workshops were organized to provide modelers with initial guidance on the publication process. Besides, a detailed step-by-step guide [51] was published.

As far as step 1 (code-sharing), nearly all modelers already use this approach for the maintenance of their source code and thus, this step did not represent a barrier for model publication.

The use of Docker technology to perform step 2 is where most difficulties were found. Very few modelers knew this technology before testing the model publication on VLab. This required modelers: (i) to spend some time to familiarize with Docker concepts (e.g., Docker image and Docker container), and (ii) install new software (the Docker engine) in their working environment to create the Docker image needed in step 2. Besides, the installation of the Docker engine and the execution of the Docker containers with shared folders on the host machine was not always straightforward for modelers working with the Windows operating system.

To address these issues, two approaches were followed: enrichment of online documentation and provision of pre-defined Docker images. The online documentation was enriched with specific sections dedicated to FAQs about the use of Docker in VLab and issues related to the use of Docker in the Windows environment. Besides, a complete "end to end" example is provided to publish a simple code on VLab. The example shows all the steps necessary to create a Docker image and to test it in the VLab environment. To do so, the example makes use of the ESA SNAP [52] library (including

its Python interface) to compute a simple conversion of a Copernicus Sentinel 2 Level 2A product to the natural color png image. The example utilizes the Python programming language. As far as pre-defined Docker images, these were created for different programming languages and software libraries according to the requirements by the modelers, including the Docker image utilized in the "end to end" example which can be re-used for ESA SNAP-based models. Available images were published on the Docker Hub VLab organization page [53]. New images for additional programming languages and software libraries will be published as soon as they will be available.

Finally, step 3 (generation of VLab conventions files) was achieved without major issues by the majority of modelers with some exception mainly for the Input/Output description file. This is where modelers must provide information about where (i.e., in which folder) the model expects to find its input(s) and where it stores the output(s) of the computation. In some cases, creating this file generated some confusion primarily for two reasons: (i) an absolute path is provided while VLab expects a relative one and (ii) the provided relative path is not correct because the execution folder is different from the one the model provider expects.

Presently, about 20 models were published on VLab and are publicly accessible. The models address different use scenarios including, but not limited to: environmental monitoring (land cover, hydroperiod estimation), species distribution dynamics, UN SDG indicators calculation, etc. Noticeably, specific procedures (e.g., pre-processing techniques implementations) for given scenarios can be shared as "models" and, thus, managed by the VLab. As far as programming languages, the following programming languages are utilized by the available models: Python, R [54], Fortran, NetLogo [55], Matlab [56], NCL [57].

5.3. Use of VLab

VLab functionalities are available via RESTful APIs, which can be exploited by client applications with which users interact.

Different client applications have been developed for different purposes and user types. The VLab Web Application (Figure 9) was developed to provide modelers with a web application to test the publication of their model on VLab and run some experiments for assessing the correctness of the outputs generated by the model running on VLab. Published models remain private until the owner decides to share the model with others. The web application was used during the VLab webinars and workshops and is the main entry point for modelers willing to publish their models on VLab.

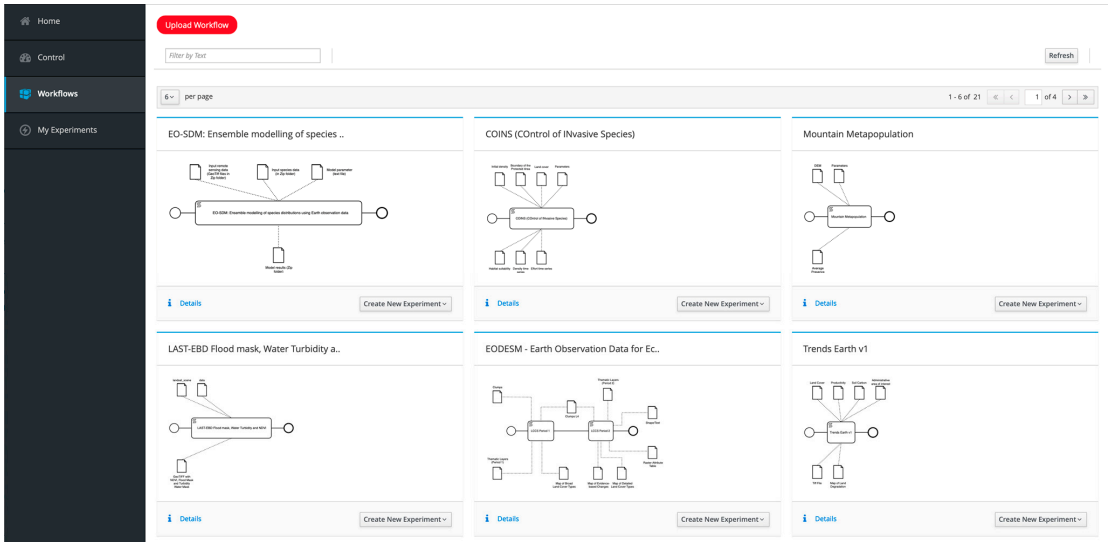

Figure 9. VLab Web Application for Model Publication and Sharing (https://vlab.geodab.org/).

In the context of the EuroGEOSS Sprint to Ministerial activity, promoted by the European Commission, a web application was developed by the Joint Research Centre of the European Commission (EC JRC) to show the use of Copernicus Sentinel data to calculate land cover and land cover changes in a set of protected areas belonging to different ecosystems. To this aim, the Earth Observation Data for Ecosystem Monitoring (EODESM) model [58] was utilized. The model is available on VLab and classifies land covers and changes according to the Land Cover Classification System (LCCS) of the Food and Agricultural Organization (FAO). EODESM requires as input data two Copernicus Sentinel 2 Level 2A products covering the same area of interest at two different points in time. First, the model processes the two products for generating land cover maps. Then, EODESM calculates the difference in the two land cover maps, generating a third output which visually captures the identified changes.

The web application utilizes VLab APIs to request the execution of the EODESM model and show users the execution status and outputs. Figure 10 shows a screenshot of the protected areas web application, displaying the output of the computation over the Gran Paradiso protected area in Italy.

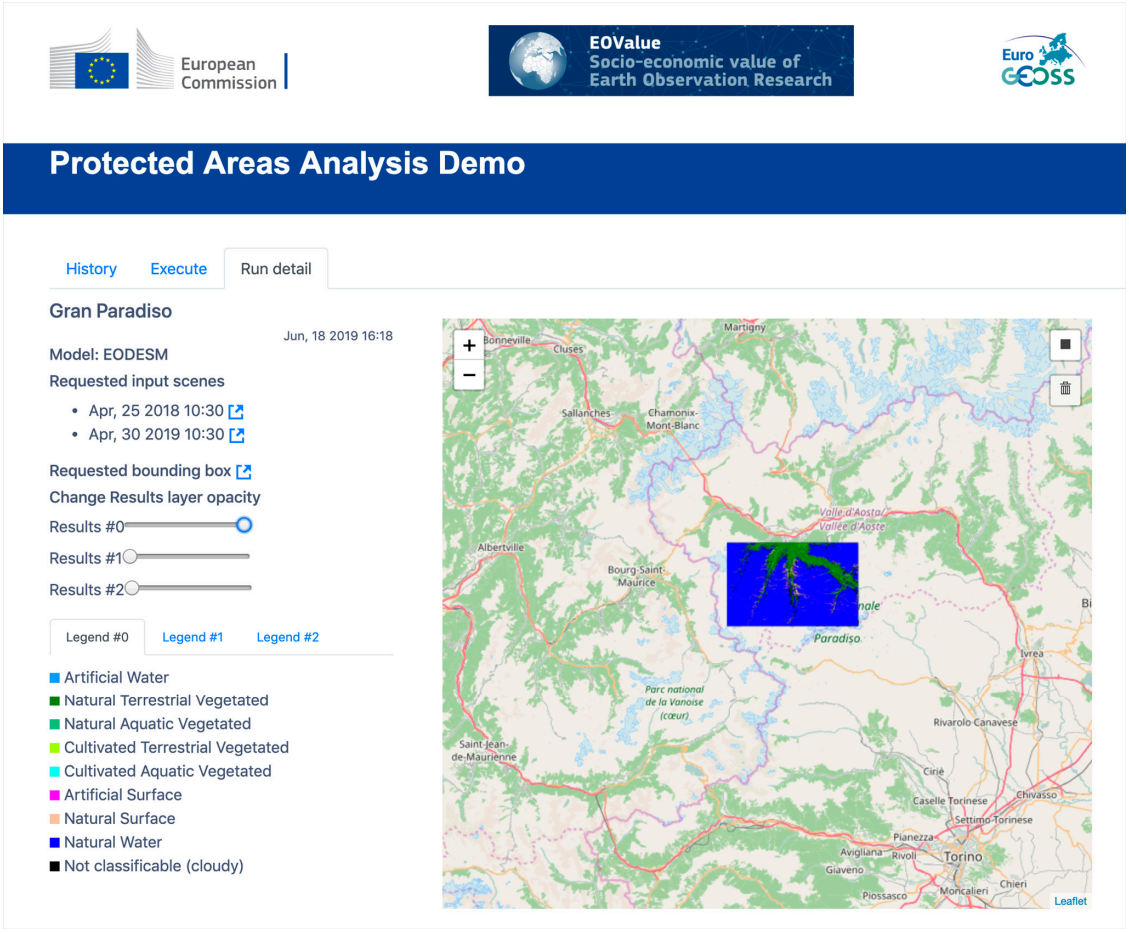

Figure 10. Protected Areas Analysis Web Application (https://pademo.geodab.org/).

The GEOEssential project is developing a web application (GEOEssential Dashboard) to let users generate and visualize UN SDG Indicators utilizing models published on VLab. A proof-of-concept was developed [59], focusing on indicator 15.3.1 (proportion of land that is degraded over a total land area) calculated utilizing the Trends.Earth model [60] on VLab over entire Europe (Figure 11). The Trends.Earth model is available as a plugin of QGIS desktop application, with source code available on the GitHub platform. The plugin calculates US SDG 15.3.1 indicator either starting from a set of sub-indicators provided by the user or generating the sub-indicators via GEE (Google Earth Engine)

scripts. The experimentation aimed at calculating the indicator from existing sub-indicators. Therefore, the publication of Trends.Earth on VLab simply required minor modifications of the source code to be able to execute the plugin without a GUI.

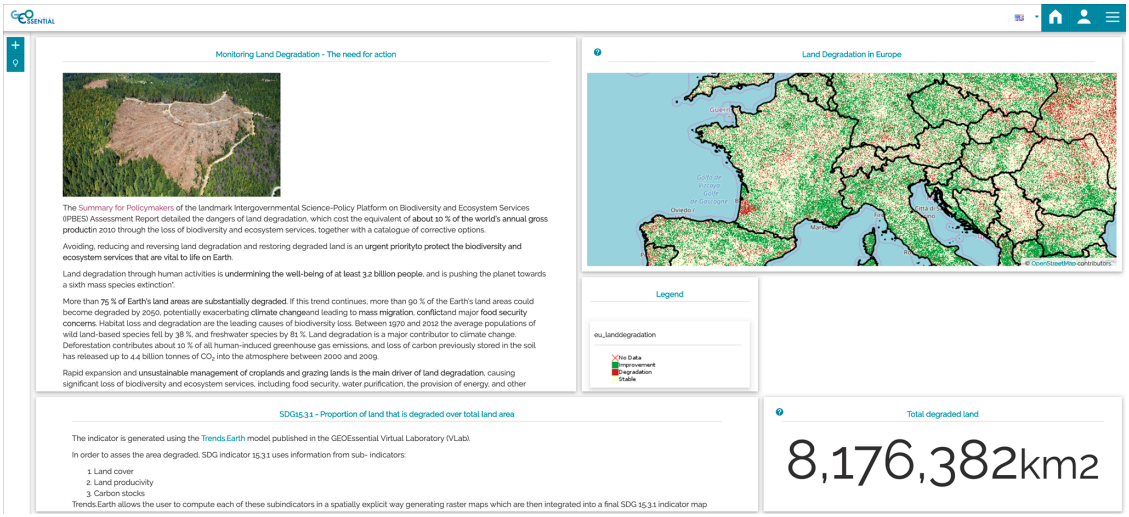

Figure 11. GEOEssential Dashboard.

5.4. Model Execution on Different Cloud Platforms

The design of VLab architecture allows the execution of models in different cloud environments. This allowed us to experiment with the execution of VLab models not only on AWS but also on other cloud infrastructures, namely: the European Open Science Cloud [61] and three of the Copernicus DIAS [62] platforms (Creodias [63], ONDA [64], Sobloo [65]).

This allows the user (or the client application) to choose which platform to use for a specific model run, e.g., depending on the required data which might be available on a specific platform only (move code to data). This was implemented in the web application for protected areas introduced in the previous section; after the selection of satellite products to use, the user is presented with the possible cloud platforms to run the computation.

Kubernetes/OpenStack services (as described in Section 4.4 Table 1) were utilized for running VLab in the experimented non-AWS cloud platforms. Besides, in the case of Copernicus DIAS platforms, the access to required EO data was achieved by exploiting the platform-specific approach: access to a shared folder via NFS for Creodias and ONDA platforms, and access via Web APIs for Sobloo platform.

Another possible scenario empowered by this multi-platform deployment feature is the possibility to let the user choose the computational platform and utilize her/his credentials to request the needed computational resources. Finally, it is also possible to exploit this feature for benchmarking different cloud platforms with respect to their performances.

6. Discussion

The Virtual Earth Laboratory (VLab) was designed and developed to address a set of requirements (see Sections 2 and 3). We think that VLab fits into many use-cases, allowing models sharing without the need to define an a-priori usage scenario. Similar to data sharing, VLab lets users freely decide how to utilize a model resource, even in a way that model providers did not foresee. Examples of usage scenarios facilitated by VLab include: reproducibility in support of open science, replicability and reusability of scientific processes, and model comparison. Innovative scenarios may include the definition and publication of digital twins, by running AI/ML-based replicas of complex physical entities, including Earth system components.

However, VLab may not be the best solution in other contexts, such as the one where complex simulation models (e.g., weather forecasts) require dedicated computing resources to achieve high performances. Another case, on the other hand, could be represented by very light models (e.g., simple indicators generation from imagery data), which could be easily run on-the-fly in a browser, not needing cloud resources. In other cases, when the model is used in a fixed scenario, dedicated solutions can provide better performances, e.g., datacubes for the generation of time-series on a fixed geographic area with fixed data sources.

VLab is already adopted in support of research and innovation projects and experimented in the context of international initiatives and programmes, noticeably EuroGEO and GEO. However, there are still significant organizational challenges to move it to a pre-operational context. In a complex and collaborative infrastructure, having a technological solution is not sufficient, since governance aspects are often the greatest barrier to its adoption. Examples of governance aspects include: defining policies for model access and use, documenting data and model quality, establishing processes for resources publications, coordinating the evolution of the core components, and addressing its financial sustainability. These are all challenges to be taken into account to move VLab in a multiorganizational context.

Another limitation of the currently implemented framework is that VLab connects data and models based on the data structure (i.e., on syntactic bases), while the workflow developer/user must take care of the meaningfulness of the scientific process. This represents a barrier, especially for non-scientist users.

7. Conclusions and Future Work

Science-informed decision-making requires the generation of knowledge from collected data and scientific models, also outlining the existence of possible different options. To address environmental policies, there is often the need for processing big datasets through complex physical or empirical (learning-based AI) models. Current cloud technologies provide valuable solutions for addressing several of the Big Earth Data challenges. However, building a workflow that implements a scientific process to generate knowledge on a multi-cloud environment, and considering diverse modeling options, is a complex task requiring multiple expertise on policy needs, different scientific domains, data and model interoperability, cloud services management. Therefore, it is necessary to automate, as much as possible, the technical tasks, in order to lower the existing barriers and allow the different stakeholders to focus on their specific fields of expertise.

The Virtual Earth Laboratory (VLab) is a software framework for the orchestration of data access and model invocation services that leverages the capabilities of existing cloud platforms to face important Big Earth data challenges. In particular, the framework addresses scientific model interoperability making them more usable with minimal agreements. VLab makes it possible to access models that are not published through web services, being just available as open-source code in a public or private repository. Therefore, VLab enables providers to share models without the need for maintaining a complex infrastructure. This gives greater visibility to the existing open models, making them more easily runnable, creating the opportunity to share "dark" models, i.e., those models still not publicly available due to technical barriers.

Based on containerization technologies, VLab is able to run models on different cloud platforms that offer IaaS capabilities. This makes VLab fitting in the multi-cloud landscape, which is going to characterize the Big Earth Data analytics domain in the next years, overcoming potential financial and legal barriers on the utilization of different cloud offerings, e.g., characterizing credit availability/costs or data/software policies.

Equipped with a geospatial data interoperability mediator, VLab is able to exploit the big amount of Earth data collected every day from many different sources, i.e., remote sensing, in situ observations, and simulations. Moving code to the clouds, which already stores the needed big data streams, reduces the need for big data movements and improves workflow execution time.

After being published on VLab, a model becomes available as a resource on the web for machine-to-machine interaction. Using the VLab RESTful APIs, software developers can build mobile and desktop applications exploiting the framework capabilities to run models. In this way, developers can build applications tailored to end-users, including policy-makers and decision-makers, widening the scope of models and their potential user audience. As an example, during the Group on Earth Observation (GEO) XVI Plenary meeting, the European Commission presented a showcase based on VLab technology. The application, developed by EC-JRC, allows assessing the extent of burnt areas after a large fire, utilizing Copernicus Sentinel-2 high-resolution images and the EODESM model.

In the framework of several European Commission funded projects, webinars and workshops have been organized to demonstrate and test VLab with modelers who largely expressed positive feedback. Some of them contributed more than general comments and suggested VLab improvements, applying a co-design and co-development agile approach.

The experience of webinars/workshops was very positive and will be continued in future, both for dissemination activities and users' feedback gathering. Besides, future work will focus on the integration of VLab with a dedicated knowledge base. Containing information on dataset types (according to a common vocabulary), the knowledge base will enable semantic interoperability, for example, by improving dataset search and providing suggestions.

Author Contributions: Conceptualization, M.S., P.M., S.N.; methodology, M.S., P.M., S.N.; software, M.S., P.M., S.N.; writing—M.S., P.M., S.N.; validation, M.S., P.M., S.N. All authors have read and agreed to the published version of the manuscript.

Funding: The research leading to these results benefited from funding by the European Union's Horizon 2020 research and innovation programme under grant agreements: n. 641762 (ECOPotential), n. 689443 (ERA-PLANET), n. 777536 (EOSC-hub), n. 776136 (EDGE), n. 34538 (EO Value).

Acknowledgments: The authors want to acknowledge the contributions of Nicholas Spadaro (EC-JRC) in the software development of the Protected Area Demo web application, Fabrizio Papeschi (CNR-IIA) and Massimiliano Olivieri (CNR-IIA) for VLab software development and cloud administration. The authors would also like to thank Joost van Bemmelen (ESA) and Gregory Giuliani (University of Geneva) for helpful discussions about the VLab potential use in real-case contexts, and Richard Lucas (University of Aberystwyth) for valuable suggestions for improving VLab. Finally, the Authors would like to thank the Reviewers for their inputs that helped to significantly improve the quality of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. United Nations. Transforming Our World: The 2030 Agenda for Sustainable Development. Available online: https://www.un.org/ga/search/view_doc.asp?symbol=A/RES/70/1&Lang=E (accessed on 31 May 2020).
2. COP 21. COP 21 Paris France Sustainable Innovation Forum 2015 Working with UNEP. Available online: http://www.cop21paris.org/ (accessed on 31 May 2020).
3. UNDRR. Sendai Framework for Disaster Risk Reduction—UNDRR. Available online: https://www.unisdr.org/we/coordinate/sendai-framework (accessed on 31 May 2020).
4. Sugimoto, C.R.; Ekbia, H.R.; Mattioli, M. *Big Data Is Not a Monolith*; MIT Press: Boston, MA, USA, 2016.
5. Guo, H.; Nativi, S.; Liang, D.; Craglia, M.; Wang, L.; Schade, S.; Corban, C.; He, G.; Pesaresi, M.; Li, J.; et al. Big Earth Data Science: An Information Framework for a Sustainable Planet. Available online: https://www.tandfonline.com/doi/full/10.1080/17538947.2020.1743785 (accessed on 31 May 2020).
6. EC. Ecopotential: Improving Future Ecosystem Benefits through Earth Observations. Available online: https://cordis.europa.eu/project/id/641762 (accessed on 31 May 2020).
7. Nativi, S.; Mazzetti, P.; Santoro, M. Design of the ECOPOTENTIAL Virtual Laboratory. Available online: https://www.ecopotential-project.eu/images/ecopotential/documents/D10.1v2.pdf (accessed on 31 May 2020).
8. EC. The European Network for Observing Our Changing Planet. Available online: https://cordis.europa.eu/project/id/689443 (accessed on 31 May 2020).

9. Mazzetti, P.; Santoro, M.; Nativi, S. Knowledge Services Architecture-GEOEssential Deliverable 1.1 2018. Available online: http://www.geoessential.eu/wp-content/uploads/2019/01/GEOEssential-D_1.1-v1.1-final.pdf (accessed on 31 May 2020).
10. Nativi, S.; Santoro, M.; Giuliani, G.; Mazzetti, P. Towards a knowledge base to support global change policy goals. *Int. J. Digit. Earth* **2019**, *13*, 188–216. [CrossRef]
11. Bojinski, S.; Verstraete, M.; Peterson, T.; Richter, C.; Simmons, A.; Zemp, M. The concept of essential climate variables in support of climate research, applications, and policy. *Bull. Am Meteorolog. Soc.* **2014**, *95*, 1431–1443. [CrossRef]
12. Lehmann, A.; Masò, J.; Nativi, S.; Giuliani, G. Towards integrated essential variables for sustainability. *Int. J. Digit. Earth* **2020**, *13*, 158–165. [CrossRef]
13. Giuliani, G.; Nativi, S.; Obregon, A.; Beniston, M.; Lehmann, A. Spatially enabling the global framework for climate services: Reviewing geospatial solutions to efficiently share and integrate climate data & information. *Clim. Serv.* **2017**, *8*, 44–58.
14. Bombelli, A.; Serral, I.; Blonda, P.; Masò, J.; Plag, H.-P.; McCallum, I. D2.2. EVs Current Status in Different Communities and Way to Move Forward. Available online: https://ddd.uab.cat/pub/worpap/2015/146882/D2_2_EVs_current_status_in_different_communities_and_way_to_move_forward.pdf (accessed on 31 May 2020).
15. Nativi, S.; Mazzetti, P.; Geller, G. Environmental model access and interoperability: The GEO Model Web initiative. *Environ. Model. Softw.* **2013**, *39*, 214–228. [CrossRef]
16. Santoro, M.; Mazzetti, P.; Nativi, S. Contributing to the GEO Model Web implementation: A brokering service for business processes. *Environ. Model. Softw.* **2016**, *84*, 18–34. [CrossRef]
17. Bigagli, L.; Santoro, M.; Mazzetti, P.; Nativi, S. Architecture of a process broker for interoperable geospatial modeling on the web. *ISPRS Int. J Geo-Inform.* **2015**, *4*, 647–660. [CrossRef]
18. Steele, K.; Stefánsson, O.H. *Decision Theory*; Zalta, E.N., Ed.; Metaphysics Research Lab, Stanford University: Stanford, CA, USA, 2016.
19. Lehmann, A.; Giuliani, G.; Ray, N.; Rahman, K.; Abbaspour, K.; Nativi, S.; Craglia, M.; Cripe, D.; Quevauviller, P.; Beniston, M. Reviewing innovative Earth observation solutions for filling science-policy gaps in hydrology. *J. Hydrol.* **2014**, *518*, 267–277. [CrossRef]
20. Object Management Group. *Business Process Model and Notation (BPMN)*, 2nd ed.; Elsevier: Amsterdam, The Netherlands, 2011.
21. Nativi, S.; Craglia, M.; Pearlman, J. Earth science infrastructures interoperability: The brokering approach. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 1118–1129. [CrossRef]
22. Vaccari, L.; Craglia, M.; Fugazza, C.; Nativi, S.; Santoro, M. Integrative research: The EuroGEOSS experience. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2012**, *5*, 1603–1611. [CrossRef]
23. Soltesz, S.; Pötzl, H.; Fiuczynski, M.E.; Bavier, A.; Peterson, L. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. Available online: https://dl.acm.org/doi/abs/10.1145/1272996.1273025 (accessed on 31 May 2020).
24. GitHub Inc. GitHub. Available online: https://github.com (accessed on 31 May 2020).
25. Oracle, Java. Available online: https://www.java.com/ (accessed on 31 May 2020).
26. Python Software Foundation. Welcome to Python.org. Available online: https://www.python.org (accessed on 31 May 2020).
27. ECMA. Standard ECMA-262. Available online: https://www.ecma-international.org/publications/standards/Ecma-262.htm (accessed on 31 May 2020).
28. Amazon Web Services Inc. Amazon Web Services (AWS). Available online: https://aws.amazon.com/ (accessed on 31 May 2020).
29. OpenStack Foundation. Openstack. Available online: https://www.openstack.org (accessed on 31 May 2020).
30. Linux Foundation. Production-Grade Container Orchestration. Available online: https://kubernetes.io/ (accessed on 31 May 2020).
31. ESSI-Lab. VLab RESTful API. Available online: http://vlabapi.geodab.org/ (accessed on 31 May 2020).
32. Docker Inc. Empowering App Development for Developers|Docker. Available online: https://www.docker.com (accessed on 31 May 2020).
33. Amazon Web Services Inc. What Is Amazon EC2? Available online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (accessed on 31 May 2020).

34. OpenStack Foundation. OpenStack Docs: OpenStack Compute (Nova). Available online: https://docs.openstack.org/nova/latest/ (accessed on 31 May 2020).
35. Amazon Web Services Inc. Auto Scaling Groups. Available online: https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html (accessed on 31 May 2020).
36. OpenStack Foundation. OpenStack Docs: Welcome to the Heat Documentation! Available online: https://docs.openstack.org/heat/latest/ (accessed on 31 May 2020).
37. Amazon Web Services Inc. What is Amazon Elastic Container Service? Available online: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html (accessed on 31 May 2020).
38. Amazon Web Services Inc. Amazon Elastic File System (Amazon EFS). Available online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEFS.html (accessed on 31 May 2020).
39. Shepler, S.; Eisler, M.; Noveck, D. Network File System (NFS) Version 4 Minor Version 1 Protocol. 2010. Available online: https://tools.ietf.org/html/rfc5661 (accessed on 31 May 2020).
40. Linux Foundation. Persistent Volumes. Available online: https://kubernetes.io/docs/concepts/storage/persistent-volumes/ (accessed on 31 May 2020).
41. Amazon Web Services Inc. What is Amazon S3? Available online: https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html (accessed on 31 May 2020).
42. MinIO Inc. MinIO|High Performance, Kubernetes Native Object Storage. Available online: https://min.io/ (accessed on 31 May 2020).
43. Amazon Web Services Inc. What Is Amazon Simple Queue Service? Available online: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html (accessed on 31 May 2020).
44. KubeMQ. KubeMQ: A Kubernetes Message Queue and Message Broker. Available online: https://kubemq.io/ (accessed on 31 May 2020).
45. Amazon Web Services Inc. What is AWS Lambda? Available online: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (accessed on 31 May 2020).
46. Amazon Web Services Inc. What is Amazon API Gateway? Available online: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html (accessed on 31 May 2020).
47. Workflow Management Coalition. Workflow Management Coalition—Terminology & Glossary. Available online: http://www.aiai.ed.ac.uk/project/wfmc/ARCHIVE/DOCS/glossary/glossary.html (accessed on 31 May 2020).
48. Nativi, S.; Mazzetti, P.; Santoro, M.; Papeschi, F.; Craglia, M.; Ochiai, O. Big data challenges in building the global earth observation system of systems. *Environ. Model. Softw.* **2015**, *68*, 1–26. [CrossRef]
49. GEOEssential. GEOEssential Project. Available online: http://www.geoessential.eu (accessed on 31 May 2020).
50. ESSI-Lab. VLab Documentation. Available online: https://confluence.geodab.eu/display/VTD/VLab+Documentation (accessed on 31 May 2020).
51. ESSI-Lab. How to Publish a Model from a Git Repository? Available online: https://confluence.geodab.eu/pages/viewpage.action?pageId=16580641 (accessed on 31 May 2020).
52. ESA. SNAP. Available online: https://step.esa.int/main/toolboxes/snap/ (accessed on 31 May 2020).
53. Docker Inc. Vlaboratory's Profile—Docker Hub. Available online: https://hub.docker.com/u/vlaboratory (accessed on 31 May 2020).
54. R Foundation. The R Project for Statistical Computing. Available online: https://www.r-project.org (accessed on 31 May 2020).
55. Wilensky, U. NetLogo, Center for Connected Learning and Computer-Based Modeling. Available online: https://ccl.northwestern.edu/netlogo/ (accessed on 31 May 2020).
56. MathWorks Inc. Matlab. Available online: https://mathworks.com/products/matlab.html (accessed on 31 May 2020).
57. NCAR. NCAR Command Language (NCL). Available online: https://www.ncl.ucar.edu (accessed on 31 May 2020).
58. Lucas, R.; Mitchell, A. Integrated Land Cover and Change Classifications. In *The Roles of Remote Sensing in Nature Conservation*; Springer: Cham, Sweitzerland, 2017; pp. 295–308.
59. Giuliani, G.; Mazzetti, P.; Santoro, M.; Nativi, S.; van Bemmelen, J.; Colangeli, G.; Lehmann, A. Knowledge generation using satellite earth observations to support sustainable development goals (SDG): A use case on Land degradation. *Int. J. Appl. Earth Obs. Geoinf.* **2020**, *88*, 102068. [CrossRef]

60. Conservation International. Trends.Earth. Available online: http://trends.earth/ (accessed on 31 May 2020).
61. EC. European Open Science Cloud (EOSC). Available online: https://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud (accessed on 31 May 2020).
62. Copernicus. DIAS|Copernicus. Available online: https://www.copernicus.eu/en/access-data/dias (accessed on 31 May 2020).
63. CloudFerro. Home Page—CREODIAS. Available online: https://creodias.eu (accessed on 31 May 2020).
64. Serco. Home—ONDA DIAS. Available online: https://www.onda-dias.eu/cms/ (accessed on 31 May 2020).
65. Airbus. Sobloo|Beyond the Data. Creative Grounds. Available online: https://sobloo.eu (accessed on 31 May 2020).