Cloud Orchestration for Optimized Computing Efficiency: The Case of Wind Resource Modelling

Marios Touloupou[1*], Evgenia Kapassa[2], Stamatia Rizou[3]

[1,*] Europen Projects Department, Singular Logic S.A, Athens, Greece, mtouloupou@ep.singularlogic.eu; Department of Digital Innovation, University of Nicosia, Cyprus, touloupos.m@unic.ac.cy
[2] Europen Projects Department, Singular Logic S.A, Athens, Greece, ekapassa@ep.singularlogic.eu; Department of Digital Innovation, University of Nicosia, Cyprus, kapassa.e@unic.ac.cy
[3] Europen Projects Department, Singular Logic S.A, Athens, Greece, srizou@singularlogic.eu

Abstract— **The Weather Research and Forecasting (WRF) model is a multi-purpose open source weather model, which is widely used by the wind energy community. However, the state of the art applications that use the WRF model for wind resources do not always meet the sector's needs. The fundamental challenge is the massive quantity of data required for accurate modelling, as well as the expenses associated with the computing of such massive volumes of data, implying that state-of-the-art approaches are not appropriate. Our work tackles this issue with the use of a novel orchestration framework relying on cloud computing infrastructure. Cloud computing platforms provide robust infrastructure that allows for the deployment of production-level services as well as extended processing and storage capacity. This paper presents the overall software architecture of a Cloud Orchestration System, enabling the dynamic and flexible cloud deployment of containerized wind resource model chains. Moreover, a set of performance, functional and operational criteria are presented and evaluated, using a real use case of a wind resource model.**

Keywords—cloud, orchestration, optimal deployment, computing efficiency, wind resources

I. Introduction

Wind power accounts for around 4% of the worldwide renewable energy mix (12% in Europe) [3], which is insufficient to limit temperature rises and reduce carbon emissions. One of the reasons for the low amount of energy generated by wind is the gap in automated tools to wind farm developers and their investors/lenders to generate quick, accurate, and financially sustainable wind site yield estimates, which hinder the sector's efficiency [4].

Notably, every year, the wind energy sector loses vast amounts of wind power generating capacity and earnings, as a result of errors made in wind resource and yield estimations, as well as ineffective modelling of the wind plants. [1]. For wind farm design and construction, the rapidly expanding wind sector demands accurate and efficient wind resource evaluation across varied topography configurations [2]. Typically, winds vary according to the location and time in daily, yearly and seasonal patterns all around the world. Assessing the wind power potential at any particular site or area is critical in determining the efficiency of wind resource for energy production within the available time constraint of wind duration [3]. Commercially successful wind modelling solutions that tackle this problem by analysing the selected regions and assessing resource capacity, necessitates extensive processing as well as advanced modelling approaches to achieve bankable accuracy, low error, and high predictability [4]. The main issue is the large amount of data necessary for effective modelling, as well as the costs associated with computing and storing such massive amounts of data, meaning that current methodologies are resource consuming [5].

Automation is necessary to significantly enhance the process and minimize the time and money required for new wind farm resource evaluation. The present paper presents a Cloud Orchestration solution, that allows scaling up the computation, increasing the computing efficiency and, potentially, reducing the computing costs. More specifically, the paper presents a cloud orchestrator for the flexible deployment of models assessing wind farm yields that is suited to the demands of wind power plant developers. Modern solutions use high-performance computing (HPC) infrastructures which have considerable limitations in terms of scalability and flexibility of HPC centres [27]. These limitations have consequently a significant impact on operational costs and restrict the upscaling possibilities of wind resource modelling as a commercial service [6]. As an alternative, this work tackles this issue with the use of a cloud orchestrator. Cloud computing systems provide an infrastructure that enables the development of production-level services, as well as processing and storage capacity. The proposed approach uses containers, a lightweight, OS-level virtualization technology that dramatically decreases the time required to launch an application. The system also includes an optimization module for continual adjustment of the containerized deployment to reduce costs while maintaining performance requirements. The implementation description of the optimization process, however, will not be disclosed in this work, and it will be treated as a "black box." In summary, the following contributions are made by this paper:

- Design of the overall software architecture, namely Cloud Orchestrator, enabling the dynamic, flexible cloud deployment of containerized wind resource model chains and WRF jobs [29].
- Interconnect appropriate monitoring tools, by enabling the processing of monitoring data, extracted from the cloud instances.
- Define a set of specific, measurable, achievable and relevant key performance indicators (KPIs) to test and evaluate the proposed architecture.
- Evaluate the technical aspects of the proposed Cloud Orchestrator including performance, functional and operational criteria. The testing and evaluation of the proposed architecture is based on the case of a real wind resource model.

The remaining parts of the paper are organized as follows. In Section 2, we discuss a set of related work of microservices orchestration in the cloud computing. Section 3 provides the description of our proposed architecture as well as the description of its APIs. Section 4 presents an overview of the use case under which the proposed Cloud Orchestrator was tested and evaluated. Section 5 presents how we conducted the evaluation to show the usefulness of the proposed solution. Finally, Section 6 concludes this paper and outlines future work.

II. Related Work

Cloud computing has drastically altered the way computing is delivered to both consumers and commercial users. Since its debut, cloud services adoption has increased, and it is anticipated that global public cloud service revenues would increase from 266.4 billion dollars in 2020 to 354.6 billion dollars in 2022 [7]. Thus, there are many (industrial and academic) systems and tools that can be considered as related work to our approach, due to the similarities in functionality with cloud orchestrators.

To begin with, Swarm [8], Kubernetes [9], and Apache Mesos [10] are container orchestration technologies that execute orchestration at the container level on previously existent resources. On the other hand, multi-cloud programming libraries such as jClouds [11], libclouds [12] and boto [13] are suitable for abstracting cloud APIs, but they are not designed to orchestrate sophisticated virtual infrastructure. Such services offer low-level APIs for accessing cloud environments, so they may be integrated as plugins into the proposed Cloud Orchestrator to enable access to even more cloud resources. Closer to our approach, Cloudify [14] is another example of an orchestrator, having been released in 2014. It is an open source cloud orchestration framework that allows users to design services and automate their complete lifecycle, including deployment on any cloud, monitoring, detecting issues and failures, manually or automatically resolving them and managing maintenance actions. Moreover, Openstack Heat [15] is a template-based orchestration, that supports auto-scaling via Telemetry integration. Although Openstack Heat has its own template format, it can also handle CloudFormation templates. Heat is open source; however, it only works with OpenStack clouds.

In addition, there is a lot of academic research in this topic [16,17]. The authors of [18] conducted a thorough study and comparison of the most important cloud resource orchestration frameworks, as well as emphasized the multi-cloud computing open challenges that the scientific community must solve in the coming years.

The authors in [19] developed an extended mixed power spectral density kernel as a prediction approach that appropriately dealt with the complexity in resource usage for cloud computing. Furthermore, the authors of [20] suggested a solution that uses a prediction approach and a convex optimization methodology to minimize energy usage in cloud computing.

Despite these efforts and to the best of our knowledge, the aforementioned solutions do not address the cost-efficient and scalable execution of numerical simulations, e.g. in the case of wind resource modelling, in cloud environments to reduce costs and increase flexibility and resource efficiency compared with the traditional HPC systems. Therefore, our proposed approach aims to address this gap by introducing a cloud orchestrator solution tailored to the needs of increased computing capacity and flexibility to support efficient and accurate wind energy modelling, while integrating it in a real wind resource model use case.

III. Cloud Orchestrator Overview

The proposed Cloud Orchestrator is a system responsible for managing the life-cycle of the deployed services (i.e. containers) that require cost-effective HPC. The Cloud Orchestrator should also handle the dynamic orchestration of the infrastructure that is used to host the services. Specifically, the proposed orchestrator provides a system which manages container-based applications consistently on cloud. This system makes the physical resources (e.g. CPUs, storage devices) transparent to the wind resource model chain services. The services' requirements are provided from a third-party component and include the following: (a) maximum allowed costs and (b) completion time for a specific service. The Cloud Orchestrator serves as an abstraction layer over computing infrastructures, physical hardware, virtual hardware, private and public clouds. This abstraction allows the developing of computing, networking and storage management algorithms which can work on a unified system, rather than dealing with the complexity of a distributed system.

The proposed Cloud Orchestrator is a container-based system. The authors of this work have chosen the logic of a container-based application to minimize any installation dependencies while also provide a cross platform application. Moreover, the Cloud Orchestrator's responsibilities are i) to promote the deployment requests for the wind resource model chain services, namely Weather Research and Forecasting *(*WRF*)* jobs, ii) Manage the lifecycle of a running job (Stop, Delete, Restart) and iii) Manage the functional requirements of the jobs (total cost, total execution time etc.). The proposed architecture of the Cloud Orchestrator is depicted in Figure 1. Currently, the Cloud Orchestrator is deployed on top of an Amazon Elastic Compute Cloud (Amazon EC2) instance. Yet, the solution can be extended to other cloud providers. Amazon EC2 and the internal components of the Cloud Orchestrator are going to be described in the following sections.

A. *Cloud Computing Infrastructure*

Cloud computing is the on-demand distribution of information technology (IT) resources through the Internet at a pay-as-you-go model. Instead of purchasing, operating, and maintaining physical data centers and servers, cloud providers like Amazon Web Services (AWS) and Microsoft Azure can be used to obtain technological services such as processing power, storage, and databases. For the reference implementation of the proposed cloud orchestrator, we used Amazon Elastic Compute Cloud (Amazon EC2), which provides scalable computing power. The usage of cloud computing eliminates the need to invest in hardware upfront, facilitating the creation and rapid deployment of applications. In addition, the proposed Cloud Orchestrator is making use of spot instances provided as an alternative of the on-demand instances, aiming at minimizing the cost [25].

B. *APIs Gateway*

The APIs Gateway component performs all activities involved in receiving and handling concurrent API calls from the third-party services. On the one hand, the API Gateway interacts with the Model Chain Controller, (i.e. a third-party service responsible for all the WRF jobs capabilities) in order to get and post deployment requests and stop or start the deployment of different tasks. Moreover, the API Gateway interacts with a third-party service for the realization of the proposed optimal configurations in a cloud environment. The APIs are available in Swagger (a readable and structured format) [21].

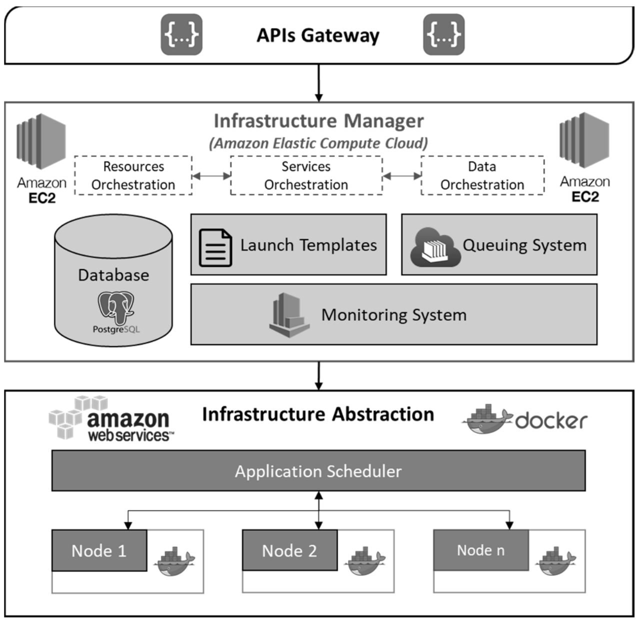

Fig. 1. Cloud Orchestrator Architecture Diagram

C. *Infrastructure Manager*

The Infrastructure Manager is the core component of the Cloud Orchestrator. It is the recipient of all the job requests, while making sure that all tasks associated with a specific job are executed in the correct order. It is also responsible for the reporting of the deployment results back to the initiator of the request (i.e. the Model Chain Controller).

1) *Launch Templates:*

In the context of our reference implementation, it is important to highlight that within EC2, an Amazon Machine Image (AMI) is a template that contains a software configuration (e.g. an operating system, an application server, and applications). From an AMI, an instance is launched, which is a copy of the AMI running as a virtual server in the cloud. Whithin Cloud Orchestrator we are leveraging the capability of Launch Templates. A launch template, specifies instance configuration information and contains the AMI ID, the instance type (e.g. c5.4xlarge, m4.10xlarge etc.), a key pair, security groups, and other EC2 instance-launching information. Defining a launch template allows you to have numerous variations of configuration, optimizing the deployment of jobs.

2) *Queueing System:*

In the current case of wind resource modelling, in order to optimize computing efficiency, the proposed system foresees that multiple WRF jobs could be submitted at the same time. In order to ensure that all jobs would run without interupptions e.t.c, the Cloud Orchestrator contains a custom queueing system which aims for the optimized management of parallel deployment requests. Jobs are submitted to the queueing system where they reside until they can be scheduled to run in a compute environment. The Cloud Orchestrator takes advantage of spot instances in order to deploy the WRF jobs. Moreover, the Queueing System provides a deployment priority to the submitted WRF jobs, that is used by the Infrastructure Abstraction, to determine which jobs in which order should be evaluated for execution first (i.e. the earliest job creation date will always have the highest priority).

3) Monitoring System:

Whithin the Cloud Orchestrator a Monitoring System has also been enabled at VM level monitoring (machine logs, CPU usage, memory, etc.). The established monitoring system in the orchestrator was enabled to better debug and understand failure of tasks due to issues identified at the VM level. The implemented Monitoring System is based on Amazon CloudWatch. Amazon CloudWatch is a monitoring and observability service designed specifically for DevOps engineers, developers, and IT administrators. CloudWatch delivers data and actionable insights to facilitate monitoring of the running services, responding to system-wide performance changes and optimizing resource use by providing a single picture of operational health. Within the proposed architecture, CloudWatch is used to identify abnormal activity in the deployed WRF jobs and the running instances, display logs and metrics side by side, take automatic actions and uncover insights to keep the Cloud Orchestrator operating smoothly. Moreover, pricing monitoring is also integrated, to enable the monitoring of the costs per wrf job in specific periods of time.

D. Infrastructure Abstraction

The infrastructure abstraction is a subsystem of the Cloud Orchestrator, located in the lower layer that enables the communication between the Cloud Orchestrator components and the dockerized services (i.e. WRF tasks). Moreover, the Infrastructure Abstraction is responsible to build, deploy, serve, and orchestrate the jobs (i.e. a "chain" of tasks) identified by the Infrastructure Manager, which are running in the form of docker containers. As it is depicted in the Figure 1, the Infrastructure Abstraction is targeting to enable multiple instances whenever needed, towards an optimal cloud environment.

IV. WIND RESOURCE MODEL-CHAIN USE CASE

The proposed cloud orchestration was tested in a wind resource model-chain use case, in the scope of the WindSider project [22]. WindSider aims to upend the paradigm by bringing to market a cloud-based tool to radically improve the assessment of new wind farm sites in two ways: (a) By significantly reducing the wind-resource and yield assessment error and (b) by reducing the prospecting and development time through automation. The current use case integrates the fully-automated wind resource assessment and data generation based on an innovative model-chain (i.e. Model Chain Service by 3E [28]) and utilizes cloud computing to enable processing towards optimization of results accuracy from very large data-sets and subsequently reduce costs and delivery time.

The Model Chain Service provides information to the Cloud Orchestrator, related to the WRF Job that is going to be deployed on top of the infrastructure. Specifically, the Model Chain Service provides the WRF *Job id, tasks ids, max cost etc.* to the Cloud Orchestrator through API communication. Then, a request towards the Cloud Optimizer is triggered, requesting the fleet of instances (a set of different instance types are used trying to minimize the probability of interruptions while using spot instances) to run the job. Furthermore, since the proposed system supports the parallel deployment of multiple jobs, a deployment priority is given to the tasks, in case they enter the queue. As soon as all tasks are completed, the Cloud Orchestrator reports back to the Model Chain Service the status of the WRF Job, as well as the results of the mesoscale model.

V. EVALUATION

The evaluation environment is an isolated part of a private Amazon cloud. Specifically, Amazon EC2 resources in the Europe (Ireland) region are used. The Cloud Orchestrator VM is deployed in a a1.large instance, with the following characteristics: (a) Custom built AWS Graviton Processor with 64-bit Arm cores, (b) 2 vCPUs, (c) 4 GB memory, (d) up to 10 Gbps network performance.

A. Evaluation Criteria

As mentioned in Section III, the Cloud Orchestrator is taking advantage of the Launch Templates feature of Amazon EC2 to deploy the WRF jobs on spot instances. Taking into consideration the Cloud Orchestrator architecture, we classified the evaluation criteria into three sub-categories (a) Performance criteria, (b) Functional criteria and (c) Operational criteria.

The following criteria (Table I) define relevant performance requirements for the Cloud Orchestrator, that are based on prior benchmarks of WRF job deployments.

TABLE I. PERFORMANCE CRITERIA

KPI	Description	Goal
Latency	Latency is defined as the time of a Cloud Orchestrator APIs to respond.	
	API to return the cost spent of a completed job	<=5s response time
	APIs to return the application and container level logs of a specific task	<=5s response time
	All the rest available APIs [21]	<=1s response time
Completion time	Completion time is defined as the time a WRF job needs to be completed with exit code '0'	<=24h competition time
Deployment time	Deployment time is defined as the time an on-spot instance needs to be ready for a job deployment	<=5m deployment time

TABLE II. FUNCTIONAL CRITERIA

KPI	Description	Goal
Number of concurrent deployment requests	How many requests can be handled at the same time through the Cloud Orchestrator (concurrency is referred to the number of requests that can be entered in the queue)	unlimited

Table II defines relevant functional requirements for the Cloud Orchestrator.
Finally, in regards to the operational criteria, the Cloud Orchestrator should be able to compete available HPC solutions and minimize the computing costs comparing to them.

B. *Evaluation Results*

For a full run of a real WRF job, 122 tasks have been deployed on c5.4xlarge instances in the eu-west-1 AWS region. Several variabilities in the duration of each task have been observed.
The next sub-sections provide more details on the evaluation criteria.

1) *Performance Metrics:*

- *Latency:* The latency of Cloud Orchestrator is defined as the time of the developed APIs to respond and the goal set is less than 1000ms (i.e. 1 second). For measuring purposes of the KPI 1 we are generating requests to all APIs of the Cloud Orchestrator using Postman [23]. The evaluation of the results of this experimentation was found promising, as presented in Figure 3.
- *Deployment Time:* The deployment time KPI of Cloud Orchestrator is defined as the time needed for an Amazon EC2 instance to be ready for deployment of a WRF Job. The goal was set to 5 minutes, and the mean for measuring is through the established logging system (i.e. Dozzle [24]). The deployment time was 4 minutes, which is acceptable since the target goal is maximum 5 minutes.
- *Completion Time:* The completion time KPI of Cloud Orchestrator is defined as the time a WRF job needs to be succefully completed. Even though several variations have been observed, the WRF job was completed within the target of 24 hours.

2) *Functional Metrics:*

The main functional requirement that was set, was the number of concurrent deployment requests, aiming at handling an unlimited number towards the Cloud Orchestrator. For that purpose, we set up a Queueing System, managing the targeted KPI. With the implemented Queueing System multiple requests are getting deployed in parallel whenever there is free space in our predefined virtual cluster. Currently, the Cloud Orchestrator can handle an unlimited number of requests, of which 122 VMs are running concurrently (on average), and the rest are waiting within the queue.

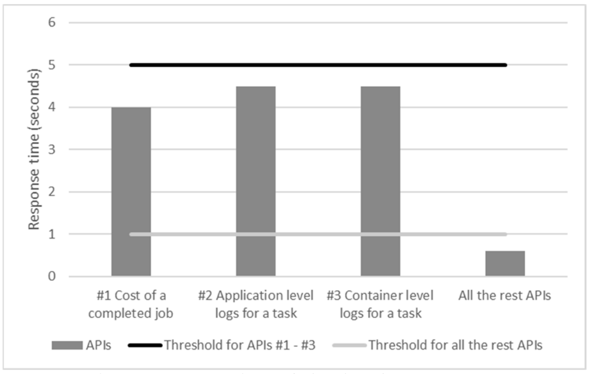

Fig. 3. Response time of cloud orchestrator APIs

3) *Operational Metrics:*

The total cost of this run has been successfully reached our expectations minimizing the computing costs. During the evaluation we took under consideration the tasks that were successfully deployed and we concluded that the operational cost to run a WRF job on top of the Cloud Orchestrator is less comparing to other solutions.

VI. CONCLUSIONS

Within this article we presented a Cloud Orchestration architecture which enables the dynamic and flexible cloud deployment of containerized wind resource model chains. The proposed architecture promotes the deployment automation and minimizes the time and cost required for a new wind farm resource evaluation. The Cloud Orchestrator is responsible for acquiring the deployment requests for the wind resource model chain services, namely the Weather Research and Forecasting *(*WRF*)* jobs, as well as for more application specific tasks such as stopping a running job, checking its status, automatically restart a job, request costs etc. The proposed system is based in a wind resource model-chain use case, in the scope of WindSider [22] and is built on top of Amazon EC2, which was used to build and host the proposed system.
Considering the massive dimension that resources have attained, as well as the proliferation of diverse cloud providers supplying resources at different levels of the cloud stack, resource orchestration is seen as a difficult undertaking. Thus, future work could evolve the proposed architecture in order to support multi-cloud deployment of containerized model chains. Specifically, we envision our approach to be more cloud-native in the future, using Cloud-native container orchestration services (e.g. Amazon Managed Kubernetes Service). The use of Amazon Managed Kubernetes Service (EKS) could help also to support multi-clouds through the EKS anywhere, which is an emerging multi-cloud container management service.

ACKNOWLEDGMENT

This work has been partially supported by the WindSider project, funded by the European Commission under Grant Agreement number: 878788.

REFERENCES

[1] Sadorsky, Perry. "Wind energy for sustainable development: Driving factors and future outlook." Journal of Cleaner Production 289 (2021): 125779.

[2] Dykes, Katherine. "Optimization of wind farm design for objectives beyond LCOE." Journal of Physics: Conference Series. Vol. 1618. No. 4. IOP Publishing, 2020.

[3] Ortega-Izquierdo, Margarita, and Pablo del Río. "An analysis of the socioeconomic and environmental benefits of wind energy deployment in Europe." Renewable Energy 160 (2020): 1067-1080.

[4] Kashem, Saad Bin Abul, et al. "Wind Power Integration with Smart Grid and Storage System: Prospects and Limitations." International Journal of Advanced Computer Science and Applications 11 (2020): 552.

[5] Peters, Jared L., et al. "A systematic review and meta-analysis of GIS use to reveal trends in offshore wind energy research and offer insights on best practices." Renewable and Sustainable Energy Reviews 128 (2020): 109916.

[6] Dorrell, John, and Keunjae Lee. "The cost of wind: Negative economic effects of global wind energy development." Energies 13.14 (2020): 3667.

[7] "Gartner forecasts worldwide public cloud revenue to grow 17% in 2020" (2019). Available at: https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecastsworldwide-public-cloud-revenue-to-grow-17-percent-in-2020

[8] “Docker swarm container orchestration tool” Available at: https://docs.docker.com/engine/swarm/

[9] “Kubernetes: Production-Grade Container Orchestration”, Available at: https://kubernetes.io/

[10] “Apache Mesos: An open-source project to manage computer clusters”, Available at: http://mesos.apache.org/

[11] “Apache jclouds toolkit”, Available at: https://jclouds.apache.org/

[12] “Apache Libcloud library”, Available at: https://libcloud.apache.org/

[13] “Boto3 SDK for Amazon Web Services”, Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

[14] “Cloudify Orchestration Platform”, Available at: https://cloudify.co/

[15] “OpenStack Heat Orchestration program”, Available at: https://wiki.openstack.org/wiki/Heat

[16] Tom-Ata, Jean-Didier Totow, and Dimosthenis Kyriazis. Real-time adaptable resource allocation for distributed data-intensive applications over cloud and edge environments. No. 4459. EasyChair, 2020.

[17] Kousiouris, George, and Dimosthenis Kyriazis. "Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay." CLOSER. 2021.

[18] https://journalofcloudcomputing.springeropen.com/track/pdf/10.1186/s13677-020-00194-7.pdf

[19] Energy efciency in cloud computing based on mixture power spectral density prediction

[20] Optimizing Power Consumption in Cloud Computing based on Optimization and Predictive Analysis

[21] “Windsider: API Specification”, Available at: http://54.247.160.180:8080/api/v1/

[22] “WindSider: AI-Powered Wind Data Platform”, Available at: https://www.windsider.io/

[23] “Postman API Platform”, Available at: https://www.postman.com/

[24] “Dozzle is a real-time log viewer for docker containers.”, Available at: https://dozzle.dev/

[25] “Amazon Spot Instances”, Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

[26] “Hybrid Cloud with AWS”, Available at: https://aws.amazon.com/hybrid/

[27] Al-Jody, Taha, Hamza Aagela, and Violeta Holmes. "Inspiring the Next Generation of HPC Engineers with Reconfigurable, Multi-Tenant Resources for Teaching and Research." Sustainability 13.21 (2021): 11782.

[28] “3E - Digitalise Your Renewable Energy and Maximise Your Asset Value”, Available at: https://3e.eu/

[29] Witha, Björn, Hahmann, Andrea, Sile, Tija, Dörenkämper, Martin, Ezber, Yasemin, García-Bustamante, Elena, González-Rouco, J. Fidel, Leroy, Grégoire, & Navarro, Jorge. (2019). WRF model sensitivity studies and specifications for the NEWA mesoscale wind atlas production runs. Zenodo. https://doi.org/10.5281/zenodo.2682604