PM

Containerizing and Evaluating the WRF model for Cloud-Based HPC

MASTER'S DEGREE PROJECT

Diogo Manuel Sales Gouveia
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2025

Containerizing and Evaluating the WRF model for Cloud-Based HPC

MASTER'S DEGREE PROJECT

Diogo Manuel Sales Gouveia
MASTER IN INFORMATICS ENGINEERING

SUPERVISOR
Karolina Baras

CO-SUPERVISOR
Rui Ricardo Almeida Vieira

Faculdade de Ciências Exatas e da Engenharia

Master of Science Degree in Informatics Engineering

Containerizing and Evaluating the WRF model for Cloud-Based HPC

Supervised by:

Karolina Baras

Rui Ricardo Almeida Vieira

Constituição do júri de provas públicas:

Mónica da Silva Cameirão (Professora Auxiliar da Universidade da Madeira), Presidente

Eduardo Marques (Professor Auxiliar da Universidade da Madeira), Vogal

Karolina Baras (Professora Auxiliar da Universidade da Madeira), Vogal

15 de setembro de 2025

Abstract

Numerical models are mathematical representations that solve algebraic or differential equations to simulate phenomena across time and space. Some models, such as the Weather Research and Forecasting (WRF) model, are applied to weather prediction, providing valuable information to individuals, companies, institutions, and governments on potential future events. However, the WRF model can become highly computationally demanding, requiring increasingly powerful hardware resources with the increase of the resolutions and complexity of the simulations. High-Performance Computing (HPC) allows users to leverage powerful instances tailored to their needs, providing the necessary hardware resources for running complex WRF simulations. Meanwhile, cloud computing presents an enticing option for computationally demanding programs, offering advantages such as economies of scale, high reliability, and availability. On the software side, compiling, configuring, and installing multiple libraries are essential to exploit WRF's potential, which can be an arduous task. This study explores a cloud-enabled approach using preconfigured Docker images and compares it to on-premises, HPC, and cloud solutions. We also analyze whether cloud computing is economically and performance-wise viable, using a comparable base architecture across multiple cloud providers, and assess if the convenience of Docker meets the requirements despite potential overhead.

Keywords: Docker, HPC, Cloud, Kubernetes, WRF

Resumo

Os modelos numéricos são representações matemáticas que resolvem equações algébricas ou diferenciais para simular fenómenos ao longo do tempo e do espaço. Alguns modelos, como o modelo Weather Research and Forecasting (WRF), são aplicados à previsão meteorológica, fornecendo informações valiosas a indivíduos, empresas, instituições e governos sobre possíveis eventos futuros. No entanto, o modelo WRF pode tornar-se altamente exigente do ponto de vista computacional, requerendo recursos de hardware cada vez mais potentes com o aumento da resolução e da complexidade das simulações. A Computação de Alto Desempenho (HPC) permite aos utilizadores recorrer a instâncias poderosas adaptadas às suas necessidades, disponibilizando os recursos de hardware necessários para a execução de simulações complexas com o WRF. Paralelamente, a computação em nuvem apresenta-se como uma opção atrativa para programas com grandes exigências computacionais, oferecendo vantagens como economias de escala, elevada fiabilidade e disponibilidade. Do lado do software, a compilação, configuração e instalação de múltiplas bibliotecas são essenciais para explorar o potencial do WRF, o que pode ser uma tarefa árdua. Este estudo explora uma abordagem baseada na nuvem utilizando imagens Docker pré-configuradas e compara-a com soluções locais, HPC e na nuvem. Analisamos também se a computação em nuvem é viável do ponto de vista económico e de desempenho, utilizando uma arquitetura base comparável entre diferentes fornecedores de serviços na nuvem, e avaliamos se a conveniência do Docker satisfaz os requisitos, apesar de uma possível carga adicional.

Keywords: Docker, HPC, Nuvem, Kubernetes, WRF

Acknowledgments

I sincerely thank my supervisors for their invaluable guidance and oversight throughout this project, as well as Ricardo Faria from OOM for his expert advice and guidance on Docker-related matters. I am also profoundly thankful to OOM and ARDITI for allowing me to contribute to this project. Finally, we extend our gratitude to CNCA for granting access to the Cirrus and Stratus platforms, which enabled the testing of WRF model simulations. As for Stratus access, we were given the project **Numerical modeling in Docker on the Cloud**, labeled with the **ID: 2024.08929.CPCA.A0**, beginning on 01/01/2025 and ending on 01/07/2025, and with a budget of 765.64 €.

Table of Contents

List of Figures

List of Tables

Lista de Acrónimos

ACI Azure Container Instances

AFIPS American Federation of Information Processing Societies

AMD Advance Micro Devices

ARDITI Agência Regional para o Desenvolvimento da Investigação, Tecnologia e Inovação

AWS Amazon Web Services

Bash Bourne-Again SHell

CCI Cloud Container Instance

CCT Cloud Computing Technology

CESGA Centro de Supercomputación de Galicia

CNCA Centro Nacional de Computação Avançada

CPU Central Processing Unit

CSP Cloud Service Provider

DEMATEL Decision Making Trial and Evaluation Laboratory

DI Docker Image

DM decision-maker

ECS Elastic Container Service

ECT Estimated Completion Time

EDR Enhanced Data Rate

EFA Elastic Fabric Adaptor

EKS Elastic Kubernetes Service

FAA Federal Aviation Administration

FCT Fundação para a Ciência e a Tecnologia

FORTRAN IBM Mathematical FORmula TRANslation System

GB Gigabyte

GCC GNU Compiler Collection

GCP Google Cloud Platform

GFS Global Forecast System

GRIB General Regularly-distributed Information in Binary form

GUI Graphical User Interface

Gb Gigabit

GbE Gigabit Ethernet

GiB Gibibyte

HDR High Data Rate

HPC High-Performance Computing

HTML HyperText Markup Language

I/O Input/Output

IP Internet Protocol

IPMA Instituto Português do Mar e da Atmosfera

IT Information Technology

KB Kilobyte

KVM Kernel-based Virtual Machine

MCDA Multi-Criteria Decision Analysis

MCDM Multi-Criteria Decision-Making

MCSR Multi-core Score Ratio

ML Machine Learning

MPI Message Passing Interface

NASA National Aeronautics and Space Administration

NCAR National Center for Atmospheric Research

NFS Network File System

NOAA National Oceanic and Atmospheric Administration

NVMe Non-Volatile Memory Express

NWP Numerical Weather Prediction

OCI Oracle Cloud Infrastructure

OOM Observatório Oceânico da Madeira

OS Operating System

PaaS Platform as a Service

QEMU Quick Emulator

RAM Random Access Memory

RHEL Red Hat Enterprise Linux

SSH Secure Shell

ST Simulated Time

TCP/IP Transmission Control Protocol/Internet Protocol

TFLOPS Tera Floating Point Operations Per Second

U.S. United States

UI User Interface

UX User Experience

VM Virtual Machine

WPS WRF Pre-Processing System

WRF Weather Research and Forecasting

vCPU Virtual CPU

1 Introduction

Numerical models are mathematical representations designed to simulate phenomena across time and space using differential equations and algebraic formulas [1]. These models have a wide range of applications, including weather forecasting, known as Numerical Weather Prediction Models (NWPs), among which the Weather Research and Forecasting (WRF) model is a popular choice. These models enable governments, organizations, and individuals to prepare in advance for potential weather events [2]. However, WRF can be computationally demanding, as the simulation is typically carried out on High-Performance Computing (HPC) machines [3]. These kinds of machines allow the solving of problems across a wide range of fields, including weather prediction, where clustered resources are used to gather better performance compared to traditional desktops, laptops, servers, or a single workstation [4, 5]. Examples of HPC clusters are Cirrus, by the Centro Nacional de Computação Avançada (CNCA), and Finisterrae III, by the Centro de Supercomputación de Galicia (CESGA), both used for multiple kinds of workloads, including WRF simulations. However, these platforms include the need for an application and shared resource usage using the Slurm Workload Manager, which means that resources required for WRF simulations may not always be available.

Due to economies of scale, Cloud Service Providers (CSPs) offer customizable access to computing resources with flexibility, reliability, and competitive pricing [6, 7]. Multiple authors have evaluated WRF on cloud platforms and HPC clusters, suggesting certain configurations. Meanwhile, WRF has technical barriers that limit accessibility for potential users, such as needing expertise in configuring, compiling, and installing WRF and its dependencies, often requiring advanced knowledge of IT systems and a significant time investment [8, 9]. Lightweight virtualization techniques could, therefore, be a possible answer to these concerns. Technologies such as Docker, which use layered images from a set of instructions, could have the required configurations for a portable WRF installation. Recognizing these issues, the National Center for Atmospheric Research (NCAR) has introduced the I-WRF framework, which includes a ready-to-use Docker image of WRF to simplify its deployment [8]. Furthermore, container orchestration tools, such as Kubernetes, can introduce the use of these images in multi-node environments, such as CSP services.

The objectives of this work are to evaluate whether Docker containers have the potential to be used in the scope of the execution of daily WRF for Madeira data, and evaluate platforms for such executions, both on an HPC cluster (Cirrus) and CSP instance (on Stratus), provided by CNCA. Meanwhile, based on our findings, we research possible cloud implementations on other CSPs (AWS, Azure, GCP, and OCI). Furthermore, a web application prototype will be developed to visualize WRF-generated data.

In this work, we will develop our own Docker images and compare their performance against native executions on different kinds of platforms. Furthermore, this document tests the execution of the model on a cloud platform, in single and multi-node modes. To complement this line of work, we will also develop a solution using a lightweight virtualization orchestration tool that enables WRF on multiple environments in a similar fashion. Moreover, within this project's scope, we will evaluate, discuss, and propose Cloud Computing Technology (CCT) services for WRF execution on the cloud. In this project, we will also present a prototype in the form of a web app for interactive mapping purposes of generated WRF data.

This document is divided into six sections, as follows: **Chapter 2, Background**, presents the relevant concepts and research; followed by **Chapter 3, Methods**, where we explain how we prepared our devices, software, and tools, as well as the procedures for gathering and obtaining results; then **Chapter 4, Results**, where we discuss the outcomes of our WRF tests across multiple environments and propose CCTs based on estimations; **Chapter 5, Conclusions**, which summarizes our findings, and discusses the Future Work, where we outline goals inspired by this study.

2 Literature Review

This chapter elaborates on the following subsections: **Numerical models:** usefulness of numerical models and introduction to the WRF model; **High-Performance Computing (HPC):** what is HPC and what kinds of workloads are done using them; **Containerization with Docker:** what is it and why it could be useful in the WRF context; **Cloud Computing:** concepts of Cloud Computing; WRF on HPCs and the Cloud: relevant work done by other researchers; **Multi-Criteria Evaluation:** introduction to the topic and a relevant method in the scope of this work.

2.1 Numerical Models

Numerical models are mathematical representations of natural behaviors, such as physical processes, and rely on algebraic or differential equations to evaluate phenomena across dimensions of time and space [1]. These models are essential in weather prediction, providing valuable data for individuals, industries, and governments to anticipate and prepare for various events [1, 2]. The most used NWP in the world is the WRF model [9], developed and maintained by the NCAR, the National Oceanic and Atmospheric Administration (NOAA) the United States (U.S.) Air Force, the Naval Research Laboratory, the University of Oklahoma, and the Federal Aviation Administration (FAA) [10]. NCAR describes WRF as a state-of-the-art mesoscale NWP system designed for atmospheric research and operational forecasting applications. WRF is highly scalable, enabling forecasts across multiple resolutions (domains) and providing accurate predictions based on real data [10]. Its flexibility allows for applications at scales ranging from tens to thousands of meters. However, computational requirements increase with model complexity and detail, often necessitating high-performance machines [10, 11]. The WRF is characterized by domains, regions of the Earth that are the target of a workload. These domains are partitioned using a Message Passing Interface (MPI) communication protocol [12].

The WRF workflow (see Figure 1) can be split into two main executions: the WRF Pre-Processing System (WPS) and the WRF itself. As for WPS, there are three main programs [13]:

- **geogrid:** Defines the projection, resolution, and the model domain extension, creating a geogrid file (`geo_em*`) based on an IBM Mathematical FORmula TRANslation System (Fortran)

namelist, `namelist.wps` [14, 15]. These files will be used in conjunction with Global Forecast System (GFS) files in the ungrib program [13–15].

- **ungrib:** Uses tables to decide which meteorological data to extract based on General Regularly-distributed Information in Binary form (GRIB) files, [13, 15], which are files consisting of information about the initial condition and lateral boundary conditions [15]. This program generates files with the prefix `FILES` for the metgrid program.

- **metgrid:** Ingests static and time-dependent data, interpolating fields to a model domain [13]. Generates `met_em*` files, for each specified period [15].

Having run the three main programs, the real (`real.exe`) program of WRF can be run, generating `wrfbdy_d01` and `wrfinput_d0*`, boundary files for the wrf (`wrf.exe`) program that includes and applies the WRF model [15, 16]. The wrf program generates `rsl.out.*` files with the forecast data.

Using WRF software requires a complex installation process and several libraries [9]. Hatheway et al. [9] developed a cross-platform tool that allows users to configure the WRF model according to their needs by the use of Bourne-Again SHell (Bash) scripts, which work in multiple Operating Systems (OS), such as Debian-based distros and Darwin-based systems, such as macOS. Despite not working in Red Hat Enterprise Linux (RHEL)-based systems and requiring 350 GB of storage, it shows a possible way to lower the WRF model's entry barrier. Another solution comes from NCAR, where Knepper et al. [8], acknowledging such issues, are developing the I-WRF framework. As further discussed in the WRF on HPCs and the Cloud section of this chapter, this provides a ready-to-use WRF model installation.

2.2 High-Performance Computing (HPC)

HPC is a technology where clustered resources are used to obtain better performance when compared to traditional desktops, laptops, servers, or a single workstation [4,5]. This type of technology is composed by three main components [4]: Computing, Storage, and Network. HPCs run differently from traditional computing workloads, where serial computing is implemented, meaning workload tasks are run one after another [4, 17]. The two most common workloads in HPC are [4]:

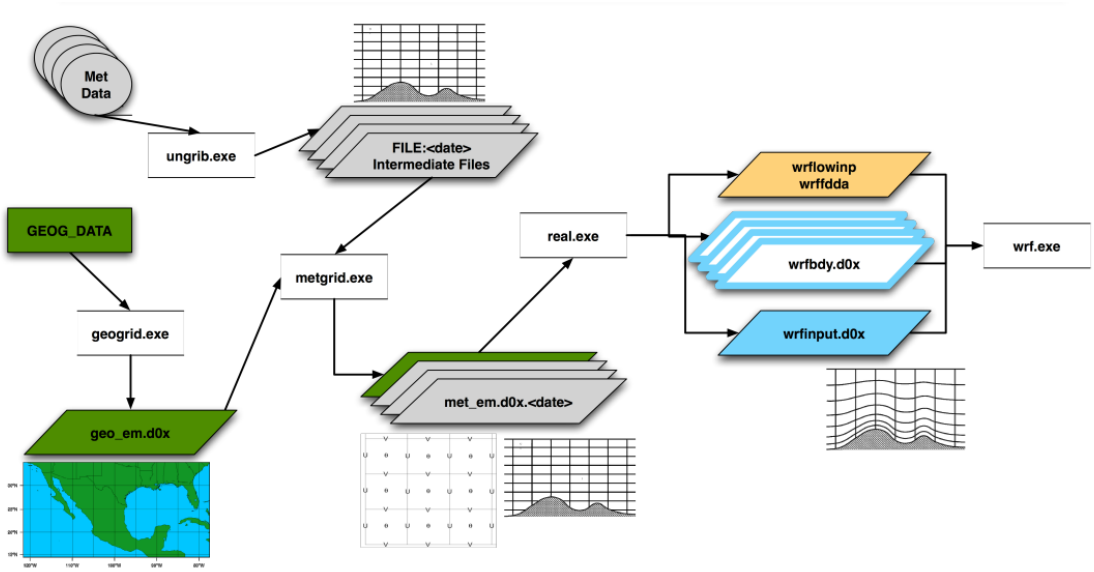

Figure. 1. WRF Workflow [16]

– **Embarrassingly parallel workloads:** Multiple independent tasks that can be run simultaneously on multiple CPU cores or servers. They can share storage, but they do not require communication among them [17].

– **Tightly coupled workloads:** Multiple tasks run across different cluster nodes, requiring low latency between nodes for communication purposes, as they communicate continuously [4, 17]. The communication is supported by a Message Passing Interface (MPI) [4]. A practical example is in weather forecasting, such as in WRF, due to interdependent physical phenomena [4].

HPC allows the solving of problems across multiple fields of academia, science, simulation, and Business Intelligence. These problems include manufacturing, retail, energy, race car optimization, aerospace, risk analysis methods, and weather prediction [4, 17]. One example of an HPC center that offers high-performant resources is Centro de Supercomputación de Galicia (CESGA) in Spain. Vourlioti et al. [18] utilized the Finisterrae II system, provided by CESGA, to study grapevine diseases by combining WRF with Machine Learning (ML). Their setup used Kubernetes to divide workloads among pods for efficient execution. Currently, Finisterrae III is available, with upgraded capabilities, multiplying the computing capacity by 12 when compared to the previous model [19]. Another HPC is Cirrus, located in Lisbon, Portugal, by the Centro Nacional de Computação Avançada (CNCA) [20]. This HPC is also used for running NWP models, such as WRF. This HPC characteristics are shown in Table 1.

Table 1. Cirrus HPC characteristics

Characteristic	**Description**
CPU	AMD EPYC 7643
Cores/node	96
RAM/node	512 GB
OS	AlmaLinux 8.10
Infiniband	1 Gbps Gigabit Ethernet
Servers	20
Queue	fct/hpc

2.2.1 Amdahl's Law

Gene Myron Amdahl developed Amdahl's Law, and it was presented at the American Federation of Information Processing Societies (AFIPS) Spring Joint Computer Conference in 1967, in the context of parallel computing [21,22]. This law states that the enhancement will provide a performance improvement compared to the original environment while keeping the workload constant [21–23]. However, the maximum potential improvement is limited to the parts of a system that cannot be improved (bottlenecks), such as serial code execution, memory bandwidth, number of CPU cores, I/O operations, and network latency [22, 24]. Therefore, Amdahl's Law tells the benefit of adding more resources to parallel tasks, being often used to predict how adding more processors enhances these tasks [22, 24]. As per equation 1, this translates to the division of the execution time of a task with no enhancements by the execution time of using enhancements.

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}} \tag{1}$$

2.2.2 Slurm Workload Manager

Slurm is an open-source scaling manager written in the C language [25, 26]. It is scalable, fault-tolerant, secure, and system administrator-friendly [26]. It has three key functions [25]: Allocate resources; Provide a framework for starting, executing, and monitoring work; Manage a queue of pending work.

A concept called entities, which are terminologies used in Slurm, provides core roles in this workload manager. These entities are [25, 26]:

- **Job:** allocation of resources for a set time.
- **Job step:** set of tasks within a job.
- **Cluster:** a collection of nodes sharing a common network.
- **Node:** compute resource.
- **Partition:** name given to a job queue.

In Figure 2, there is an example of a partition, job, and job step allocation.

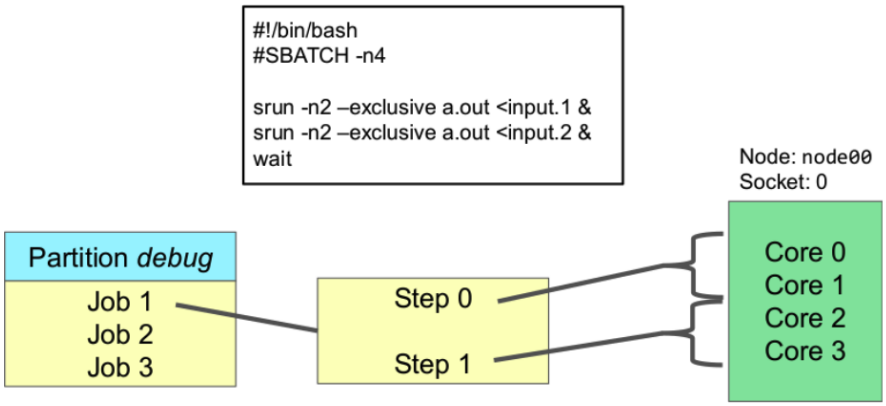

Figure. 2. Example partition, job, and job step allocation. [26]

There are also commands that provide functionalities to the user, and were used throughout this work, these are [25]:

- `scontrol`: tool to view or modify Slurm state
- `squeue`: reports the state of jobs and job steps
- `sbatch`: command to submit jobs
- `srun`: used to submit jobs for execution and resources
- `sacct`: reports about active or completed jobs

Other commands, such as `slurmctld` act as controller daemons, and `slurmd` act as compute daemons [27].

In Figure 3, there is a visual representation of the scope of all the mentioned commands.

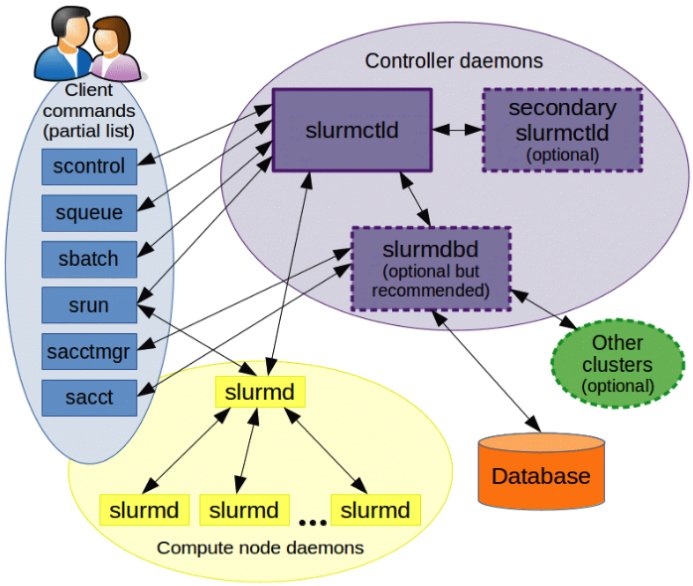

Figure. 3. Slurm components [25]

2.3 Containerization with Docker

Containerization is a popular technique for running multiple services under a common OS [28, 29]. In Docker, containers are created through Docker images, which contain instructions for a specific installation [30]. These images are based on layers generated from the instructions, each stacked on a unique directory [28]. Other parts of the Docker ecosystem include key components such as the Docker Engine, Docker Hub (a repository for sharing container images), and Docker Desktop, which provides a user-friendly GUI and supports command-line operations. Docker's capabilities are further enhanced by orchestration tools like Docker Swarm or Kubernetes, enabling efficient management of container clusters [28]. Due to the benefits of Docker, it has become an integral part of modern Information Technologies (IT) infrastructure, supporting isolated services and applications efficiently [30, 31]. Therefore, for WRF, which can be a demanding application due to the input data, Docker can simplify its deployment by offering ready-to-use container images.

These images provide a flexible and accessible solution for diverse users, including new adopters and those extending their workflows to the cloud [8]. While Docker offers excellent benefits, it is bound by root access to the system by users [30, 32]. So, to solve that aspect, Gomes et al. [32] developed an application called udocker, which enables rootless Docker execution in multi-user systems, i.e., allows users to use Docker with the proper permissions. This tool is currently used in Cirrus as a module package, providing access to container technologies in this HPC environment.

2.3.1 Kubernetes

Kubernetes is an open-source container orchestrator platform developed by Google [33], that provides a flexible way of scaling services, being available as Platform as a Service (PaaS) [34]. Major CSPs such as Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), and Oracle Cloud Infrastructure (OCI) offer their own Kubernetes services, enabling organizations to build cloud-native applications [34, 35].

The atomic unit of deployment in Kubernetes is a pod, which is composed of one or more tightly coupled containers and can be used with other container solutions, such as Docker [36]. For example, Docker images are compatible with Kubernetes, allowing seamless integration with that platform. Pods are isolated using namespaces and cgroups, Kubernetes has the advantage of using pods in such a way that automates the management of containers and load balancing [34, 37].

In a Kubernetes system, there should be a master node and one or more worker nodes. [35] These nodes can have multiple pods, according to the required resources. In Figure 4, an example is presented containing other Kubernetes components as well.

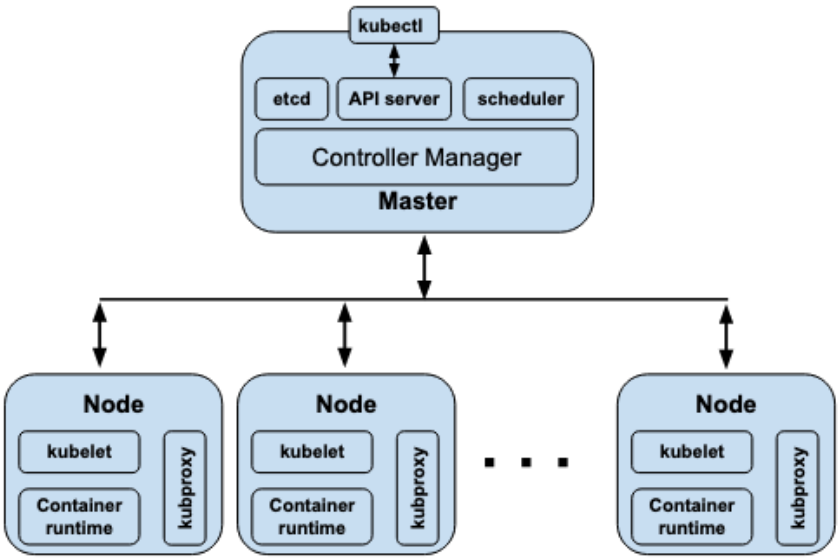

Figure. 4. Kubernetes multi-node example architecture. The master has the administrative role, while the workers perform a task [34].

In the context of MPI-based applications, communication is a key factor. If pods reside on the same host machine, they can communicate with each other, since they are under the same sub-network. However, if they are across multiple nodes, an overlay network such as Flannel is required. Other factor includes Secure Shell (SSH) communication between containers [34].

Multiple authors studied the use of MPI-based applications through Kubernetes in HPC environments. Beltre et al. [34] compared Kubernetes and Docker Swarm, another container orchestration platform, and with native execution, where they found an overhead in Kubernetes when compared to the other solutions, mainly due to the whole virtualized network. However, Milroy et. al [36] developed a custom scheduler, Fluence, and used Elastic Fabric Adaptor (EFA) on AWS for communication purposes, where significant improvements were verified when compared to the standard and other state-of-the-art solutions. On a similar note, Hossen et al. [37] studied four common MPI applications across AWS Elastic Kubernetes Service (EKS) clusters, with their developed scaling algorithm reducing the run time up to 5x. Additionally, Vourlioti et al. [18] used Kubernetes as the orchestrator for their GRAPEVINE platform on HPC and the cloud, essential for running all models, including WRF. In Figure 5, their Kubernetes architecture is presented, where WRF is used as a single container. They mention that a benefit that could be implemented in the future is that of portability, as in transferring data between multiple platforms.

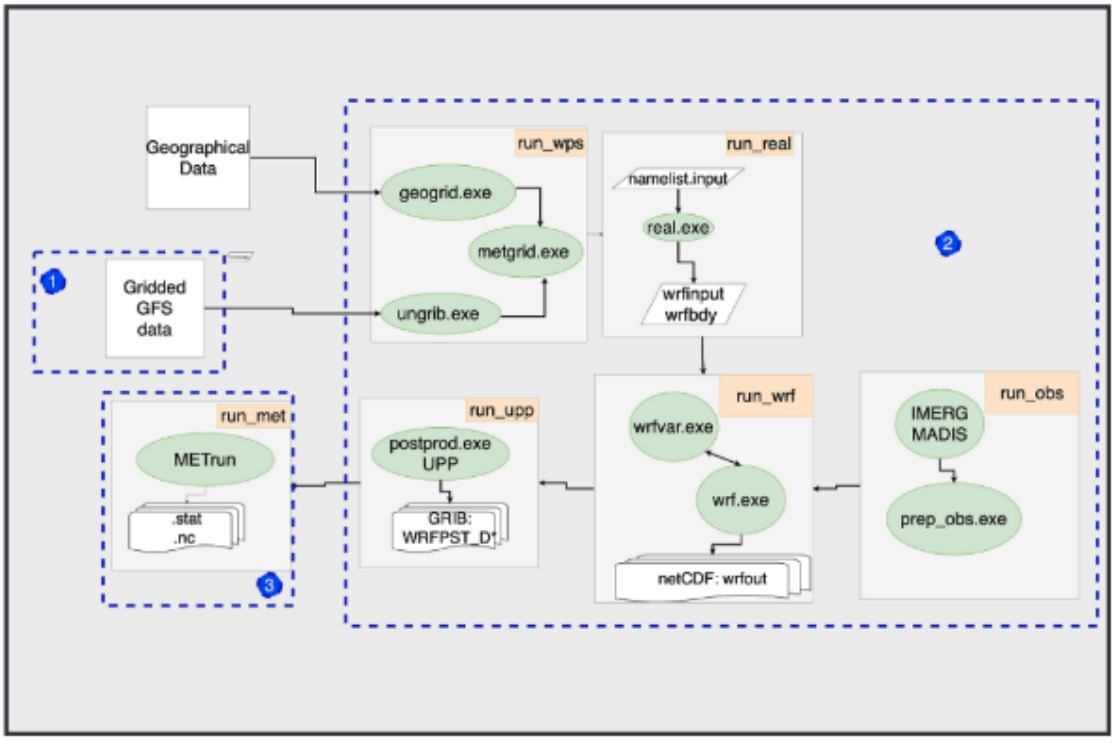

Figure. 5. Kubernetes GRAPEVINE architecture. [18] This shows a division of roles through blueprints 1, 2, and 3, based on the relationship of execution of tasks.

2.4 Cloud Computing

Cloud computing involves accessing servers, software, and databases over the Internet, all of which are hosted in data centers worldwide, enabling users to avoid managing physical infrastructure and providing access to multiple resources over the Internet [38–40]. Major CSPs include Amazon (AWS) (31%), Microsoft (Azure) (20%), Google (GCP) (12%), Alibaba (Alibaba Cloud) (4%), and Oracle (OCI) (3%) [41]. Apart from the above-mentioned CSPs, CNCA is also a CSP, which provides access to the Stratus platform through an application [42]. As for the advantages of cloud computing, it offers strong benefits regarding performance, scalability, and elasticity when compared to conventional on-premise IT infrastructure, making cloud computing usage grow exponentially [43, 44].

The key advantages of cloud computing include:

– **Cost:** A pay-as-you-go model eliminates fixed expenses like hardware and software acquisition, benefiting from economies of scale [7, 38, 44].

- **Global Scale:** Cloud resources are accessible worldwide, facilitating seamless scalability [6, 7, 45].
- **Performance:** Optimized data centers ensure high-speed, reliable performance [7].
- **Speed:** Resources can be deployed rapidly to meet changing demands [7].
- **Productivity:** Cloud eliminates on-premises infrastructure maintenance, enabling teams to focus on core tasks [7].
- **Reliability:** Provides data backups, disaster recovery, and high availability [7, 45].
- **Agility and Elasticity:** Resources can be dynamically scaled based on real-time requirements [6, 7, 45].

2.5 WRF on HPCs and the Cloud

The WRF model benefits from cloud computing's scalability and cost-efficiency. However, standard cloud environments may introduce challenges such as heterogeneous processing speeds and high network latency.

To address these challenges, Krishnan et al. [46] evaluated WRF on AWS, optimizing node and thread configurations, and compared it to the traditional HPC cluster, SingTel, used in weather forecasting in Singapore. As for AWS, they configured a CC1.4xlarge instance running Ubuntu 12.04 LTS with a compiled WRF 3.4.1 installation (Intel Fortran). They aimed to identify the optimal configuration of node counts and physical and virtual cores. The instance offered 16 cores in total, with 8 of them being hyper-threaded. They found that the best performance was achieved by running 4 WRF tasks on each of the 8 instances. Meanwhile, when comparing AWS and SingTel, AWS outperformed SingTel due to the placement groups, which are high-bandwidth and low-latency networking node groups, present in the former.

Building on this, Xu et al. [45] investigated multiple MPI libraries on Frontera (Intel Xeon Platinum 8280), a traditional HPC, Azure, and AWS instances. For Azure, they tested Azure-HC (Intel Xeon Platinum 8168) and Azure-HBv2 (AMD 7002-series) instances, while for AWS, they used an AWS-c5n.18xlarge (Intel Xeon Platinum 8124M) instance. Their findings highlighted that Frontera and Azure outperformed AWS in message latency (2 ms>18 ms), with the Azure HBv2

instance being particularly effective due to its 200 Gbps bandwidth, compared to the 100 Gbps offered by other cloud instances. Additionally, AMD EPYC CPUs demonstrated superior intra-node latency. When comparing the Execution Time across the four solutions, Frontera emerged as the superior option due to the overhead of Virtual Machines (VMs) on AWS and Azure.

Complementing the studies of WRF on the Cloud, Amazon [47] provides a sample architecture for a WRF-ready solution. Their example highlights the use of hpc6a.48xlarge instances (96 CPU cores), although other instance types are also compatible with their recommendations. Table 2 lists some advised AWS instances, which include the other types.

Table 2. Available EC2 HPC Optimized instances in Europe

AWS Region	EC2 HPC Optimized
eu-north-1	hpc6id.32xlarge
	hpc6a.48xlarge
eu-west-1	hpc7a.12xlarge
	hpc7a.24xlarge
	hpc7a.48xlarge
	hpc7a.96xlarge

Microsoft [48] complements these by emphasizing the importance of weather forecasting in disaster prevention and preparation. According to Microsoft, four key architectures are suitable for running WRF: HBv2 and HC (both tested by Xu et al. [45]), HBv3 (AMD 3rd Gen "Milan"), and HB (AMD EPYC 1st Gen "Naples") (see Table 3).

Table 3. Microsoft recommended Cloud HPC instances

Instance	HBv3	HBv2	HB	HC
CPU Family	AMD EPYC 3rd Gen "Milan"	AMD EPYC 2nd Gen "Rome"	AMD EPYC 1st Gen "Naples"	Intel Xeon Platinum 1st Gen "Skylake"
CPU	AMD EPYC 7V73X @ 3.5 GHz	AMD EPYC 7V12 @ 3.1 Ghz	AMD EPYC 7551 @ 2.55 GHz	Intel Xeon Platinum 8168 @ 3.7 GHz (single core), 2.7-3.4 GHz (all cores)
Cores/VM	120	120	60	44
TFLOPS/ VM (FP64)	8 TF	4 TF	0.9 TF	2.6 TF
Memory	448 GB	4 GB/core, 480 GB total	4 GB/core, 240 GB total	8 GB/core, 352 GB
Local Disk	2x960 GB NVMe	900 GB NVMe	700 GB NVMe	700 GB NVMe
Infiniband	200 Gb HDR	200 Gb HDR	100 Gb EDR	100 Gb EDR
Network	50 GbE	32 GbE	32 GbE	32 GbE

Similarly, Google [49] provides an example of a cloud HPC implementation for WRF. They tested WRF v4.2 with OpenMPI 4.02 using the CONUS 2.5km benchmark, a test case by UCAR on a 2.5 km grid, and a 1501x1201 resolution [50]. The c2-standard-60 instance type outperformed alternatives like n1-standard-96 and n2-standard-80. Moreover, using "intel-oneapi-compilers" resulted in approximately 47% faster performance at 68% of the cost compared to GCC versions 11.2.0 and 10.3.0. Despite this, a file Input/Output (I/O) bottleneck was identified, prompting the recommendation of parallel file systems like Lustre FS. Notably, their study also confirmed that increasing the number of MPI tasks does not guarantee a proportional performance boost due to overhead; for example, doubling tasks from 960 to 1920 resulted in only a 1.5x speedup.

Adding to this body of work, Munhoz and Castro [51] tested multiple HPC applications on AWS and Vultr instances, two cloud services. They found that the best results were generally achieved with a 4x8 configuration at 32 MPI ranks (tasks).

Similarly, Oana et al. [52] benchmarked WRF across various environments and observed, as other studies did, that efficiency diminishes as the number of MPI processes increases. They also compared how Azure performs (1 to 20 cores) against two traditional HPCs, Infragrid (Intel Xeon quad core - 56 cores total) and Bluegene/P (PowerPC 450 [53]) (32 to 1024 cores). Azure achieved the lowest execution time on low WRF resolutions and achieved similar results to Bluegene/P on higher resolutions.

While these studies elaborate on the use of WRF in a cloud environment, they also reveal a limitation in the sense that they are tied to specific CSPs, such as Amazon, Microsoft, or Google. This dependency can restrict flexibility when deploying WRF on local machines, alternative HPC setups, or other cloud services [8]. To address this, NCAR is developing a framework based on a pre-configured and ready-to-use Docker image. However, the documentation is still in its alpha phase, and an Intel CPU is required [8].

Beyond traditional HPC and VM-based cloud environments, an emerging alternative lies in Cloud Container Instances (CCIs). For instance, Oracle offers OCI Container Instances, a serverless and compute-optimized service designed for containerized workloads. This service eliminates the need for a container orchestration platform like Kubernetes, providing strong isolation, built-in user management, and flexible configurations based on AMD E4 Flex and Ampere A1 Flex CPUs [54]. With a pay-per-use pricing model, OCI Container Instances represent a promising option for efficient and scalable deployments. [55]

2.6 Multi-Criteria Evaluation

Multi-Criteria Decision Analysis (MCDA) or Multi-Criteria Decision-Making (MCDM) is a structured methodology for evaluating and selecting the best option from a set of alternatives while considering more than one criterion in the selection process [56, 57]. It is one of the most accurate methods, reducing subjectivity in decision making [57, 58]. It involves defining criteria, assigning weights, and analyzing options to identify optimal solutions [58]. In the context of CCT, CSPs offer different services, with various prices and performance choices, making choosing the most suitable step difficult [43, 44].

DEMATEL is MCDA/MCDM method that can act as a tool to evaluate CCTs. This method has been a target of continuous research interest, as shown by an increasing number of articles published per year [59]. It is a widely used method for identifying and visualizing relationships between components of complex interactions using matrices or digraphs. [60] Developed at the Geneva Research Center of the Battelle Memorial Institute, it has applications in diverse fields, including computer science, engineering, and management [59,61]. Büyüközkan et al. [40] employed a 2-Tuple DEMATEL, by combining a 2-Tuple linguistic model and the DEMATEL method, to evaluate CCTs for a Turkish company.

Büyüközkan et al. [40] suggest 2 stages:

Stage I

1. Definition of dimensions for CCT assessment
2. Establishment of inter-dependencies of the criteria dimensions and criteria

Stage II

1. Normalization of the Decision Makers (DMs) evaluations under the linguistic terms set that have the highest granularity
2. Constructing the average matrix (A) with linguistic assessments of DMs
3. Calculating the initial direct influence matrix (D)
4. Calculating the total direct/indirect influence matrix (T)
5. Analyzing the cause and effect diagrams

For evaluation purposes, linguistic scales are used in order to allow experts to evaluate criteria according to their level of experience, having used 3 label sets, with deepening granularity (see Table 4). These are used to build an evaluation matrix of the respective experience with CCTs.

Table 4. Linguistic scales [40]. **Meaning:** N - No influence; L - Low influence; ML - Medium Low influence; AM - Almost Medium influence; M - Medium influence; AH - Almost High influence; H - High influence; VH - Very High influence; P - Perfect influence.

Experience/Granularity	1	2	3	4	5	6	7	8	9
Low	N	L	H	-	-	-	-	-	-
Medium	N	L	M	H	P	-	-	-	-
High	N	L	ML	AM	M	AH	H	VH	P

2.7 The Link between the Background and the Project's Development

Based on the reviewed literature and current state-of-the-art solutions, we explored potential directions for our project. In terms of cloud computing, many authors cited services from AWS and Azure as possible deployment options, while GCP and OCI presented viable alternatives. Notably,

Google, alongside Amazon and Microsoft, and other authors, have also explored running the WRF model using CCTs. These studies proved useful in guiding the selection of appropriate instances from each cloud service provider (CSP).

We further examined the instance types recommended by Amazon (see Table 2), Microsoft (see Table 3), and Google for similar computational tasks. Regarding storage, our initial proposal considered a Lustre FS-based solution. However, this approach quickly revealed itself to be prohibitively expensive for the stakeholders.

Additional aspects, such as the compiler version and type, were also evaluated. We initially attempted to use Intel's OneAPI compiler in our custom Docker image, but licensing restrictions rendered this approach unfeasible. Nevertheless, we were able to experiment with the I-WRF Docker image, which utilizes this compiler.

To support the evaluation of our proposed cloud architectures, we developed two questionnaires based on the methodology described by Büyüközkan et al. [40], intending to enable future refinement of our proposal.

Our research into technologies like Kubernetes and Slurm also provided valuable insights. We designed our project's infrastructure following common patterns identified in the literature, such as using a controller node in Slurm or a master node in Kubernetes, followed by multiple worker nodes.

In particular, for the Kubernetes-based solution, we initially considered the GRAPEVINE architecture (see Figure 5), which demonstrates how the WRF model can operate in such an environment. However, we extended our approach to align more closely with the use of Slurm, drawing on the works of Beltre et al. [34], Milroy et al. [36], and Hossen et al. [37].

3 Methodology

3.1 Background

This chapter discusses the methods devised in this project, which used multiple tools and devices. A device, a laptop acting as a Local Machine, and two services provided by CNCA, Cirrus, an HPC cluster, and Stratus, a Cloud platform powered by OpenStack. With these, we developed a methodology across the multiple fields of this work. It is divided into the following sections: **Local Machine**, where we present the device that allowed preliminary testing and development; **WRF Installation and Docker Images Development**, where provide insights about these aspects; **Cirrus - CNCA HPC Platform**, which describes the preparation of the environment for running WRF; **Stratus - CNCA Cloud Platform**, which discusses how it works and the preparation done to run WRF; **Kubernetes Solution**, where we show a multi pod implementation for running WRF akin to Slurm but with a Docker image; **Evaluating Cloud Solutions**, section where we analyze and present CCTs to enable using WRF in the Cloud; **Web application**, which states the preparation for the development of the interactive web mapping protoype.

3.2 Local Machine

A DELL Vostro 5568 laptop was provided by Observatório Oceânico da Madeira (OOM) to evaluate and develop Docker images for WRF. This machine acted as our Local Machine for testing and Docker images development. The specs of this Local Machine are presented in Table 5. We installed Docker Desktop 4.33 through the .deb package for Ubuntu[1]. We used Docker Desktop as it provides a Graphical User Interface (GUI), has integrated Kubernetes, receives automated security patches, can perform image vulnerability scanning, and can deploy images to AWS Elastic Container Service (ECS) and Azure Container Instances (ACI) [62]. In Table 5, it is possible to check the details of the Docker Desktop setup on the Local Machine. To evaluate the WRF model, we did some full tests using pre-processed U.S.A. data (further details in the WRF Installation and Docker Images Development section) and 1-hour tests using pre-processed Madeira data for a 72-hour forecast, due to high Estimated Completion Time (ECT) values. ECT in our work stands for the time a complete daily simulation (full 72-hour weather forecasting preview) would take.

[1] `https://docs.docker.com/desktop/setup/install/linux/ubuntu/`

Table 5. Local machine configuration

Characteristic	**Description**
CPU	Intel Core i5-7200U @ 2.50 GHz (Dual-core)
RAM	16 GB (2x 8 GB DDR4 2400 MT/s
Storage	256 GB SK Hynix SC311 SATA 6.0 Gb/s Solid State Drive (SSD)
GPU	Intel HD Graphics 620
Host OS	Linux Mint 22 Cinnamon (64 bit)
Host Kernel	6.8.0-39-generic
Docker version	4.33
Docker CPU limit	4
Docker memory limit	5.8 GB
Docker SWAP	2 GB
Docker virtual disk limit	128 GB

3.3 WRF Installation and Docker Images Development

Since we aimed to test the WRF performance across multiple solutions, we did a complete installation, i.e., libraries, WPS, and WRF, following a mix of three sources: two installation scripts provided by OOM, and the official WRF Compiling Guide by NCAR [15]; on the native environment of the Local Machine (Linux Mint 22), and two Docker images: Ubuntu 24.04 LTS (Debian-based), AlmaLinux 8.10 Minimal (RHEL-based), both released in 2024 and with no vulnerability issues found at the date. These images can be found in the Docker Hub repository ehxa/oom[2].

OOM provided a Dockerfile for developing Docker images. This Dockerfile configured an image based on CentOS 7, which is deprecated. We used this Dockerfile, making the necessary adaptations and updates, including some optimizations, as the basis for all our Dockerfiles.

Getting back to the installation, as we tested the steps from the three sources, we combined the instructions into a single `.txt` file, called `installguide.txt`, that can act not only as a guide for RHEL and Debian-based distros but can also be adapted as a Bash script for other Unix systems. Based on the `installguide.txt`, we created install scripts, called `wrf_install.sh`, for RHEL-based, Debian-based, and openSUSE distros. For installation and running duties, we created another Bash script file, called `gccvars.sh`, that sets the proper environment variables for the

[2] `https://hub.docker.com/repository/docker/ehxa/oom/general`

complete installation and running WPS and WRF software. This script is automatically applied by the also developed `wrf_run.sh` script, which allows the user to decide which environment to select and what type of test is to be done. These three scripts allow, in case of a distro change, upgrades, or issues, a complete automatic installation and execution. The Dockerfiles, `installguide.txt`, and the scripts can be found on GitHub under the ehxa/OOM-Internship repository[3]. On the native and Ubuntu 24.04 Docker environments, we used GCC, G++, and GFortran 9.5, and for the AlmaLinux 8.10 Minimal, we used 9.2, available through the gcc-toolset-9 package. Therefore, the required, configured, compiled, and installed libraries across all environments (except I-WRF) were: zlib-1.2.11; hdf5-1.10.5; netcdf-c-4.9.2; netcdf-fortran-4.6.1; mpich-3.3.1; libpng-1.6.37; jasper-1.900.1. As for WPS, option `3 (Linux x86_64, gfortran (dmpar))` was selected for configuration before compilation. For the configuration step on WRF, we selected options `34 ((dmpar), GNU (gfortran/gcc))`, and `1 (basic)`, as suggested.

To test if our installation was completed successfully, we used data corresponding to the 25th of August, 2024, sourced from the National Centers for Environmental Prediction (NCEP), which generates the Global Forecast System (GFS) data daily (United States of America (U.S.A.). The files, provided in the GRIB format, are typically deleted after 2-3 days, so they were backed up to a personal OneDrive folder to ensure availability. Three files were used, corresponding to 12:00 am, 3:00 am, and 6:00 am of the specified date. These files were stored in a dedicated directory named `DATA`, created inside the `wrf` directory for better organization.

To process the U.S.A. data, the `namelist.wps` file was updated to match the date and time of the GRIB files. Subsequently, the WPS programs, geogrid, ungrib, and metgrid, were executed. The final step (using metgrid) generated the necessary `met_em*` files for running WRF. These output files were linked to the `test/em_real` directory of the WRF environment, setting up the simulation environment. To validate the Local Machine setup, `real.exe` was executed, confirming that all configurations and dependencies were properly in place.

3.4 Cirrus - CNCA HPC Platform

Access to this platform was granted, allowing us to test its capabilities and use them to propose Cloud solutions. This access is provided with a remote connection through a verified SSH key. For

[3] `https://github.com/ehxa/OOM-Internship`

native environment testing purposes, we created a directory, `proj_D`, that was the place where we ran the WRF simulations. Inside `proj_D`, two subfolders were created, one called `RAM` and another `USA`. Due to credit constraints and for usefulness purposes, we only evaluated WRF using Madeira data for 1 hour of execution, getting the ECT values.

WRF was compiled using an already-developed script by OOM that loads the required compilers and libraries. We linked the Madeira data to the `run` directory of the WRF installation so that the tests with WRF could be executed. To run WRF tasks, we needed to submit the execution of the `wrf.exe` program as a job to the Slurm Workload manager, the queue system [25] used in Cirrus. For that purpose, an also Bash script developed by OOM was used. This script includes options such as the number of requested nodes, the number of CPU cores per node, and the instance type. A waiting time may arise due to the shared architecture. In our case, we only observed some waiting time when requesting 30 cores for a single-node.

When submitted, we checked our job status with the `squeue` command. The `top` command was used to view the usage and `tail -f rsl.out.0000` to view the simulated time. Meanwhile, the Slurm Workload Manager generates a log stating the job's outcome upon completion.

As for Docker testing on Cirrus, the alternative is provided as udocker. Udocker is available as a module, a type of package with the required environment variables and configurations, meaning that no installation is necessary. For testing purposes, we loaded using `module load udocker`. To maintain consistency in our results, we always checked which version we were loading by doing `module avail | grep udocker` (udocker 1.3.17). To run the Madeira tests on a Docker container in Cirrus and compare its performance, we used the developed Ubuntu 24.04 LTS Docker image for a udocker container that we previously developed on the Local Machine by doing `udocker pull`, due to having the best performance in our previous tests on the Local Machine. Running the tests on udocker requires no extra steps compared to running on the Local Machine, since submitting any of the udocker steps as a job to the Slurm Workload Manager is unnecessary. To compare the native and udocker environments, we compared tests with the same number of CPU cores (tasks) on 1 node.

We found that udocker had a limitation with the number of CPU cores available and the number of nodes. Due to permissions, we cannot change this limit. Therefore, udocker can only run in 1 node with a maximum of 12 CPU cores. We did test udocker with a larger number of CPU cores

to verify if that was only an arbitrary value, and we tested if it was scalable. However, it was a rigid boundary.

3.5 Stratus - CNCA Cloud Platform

Stratus is another alternative offered by CNCA. Similar to the access requirements of Cirrus, an application is necessary to get access and leverage its resources. For this purpose, we applied for A0 access to the Stratus platform, which translates to 48000 virtual CPUs (vCPUs)*hours and up to 64 vCPUs. Our application was approved, with a value of 765.74€, with a start date of 01/01/2025 and access up to 01/07/2025, given the ID: 2024.08929.CPCA.A0.

Stratus is based on OpenStack, a tool developed by NASA and based on Python, and is also compatible with multiple virtualization solutions, such as QEMU, Hyper-V, and KVM, making it a flexible and scalable solution, able to deal with massive data [63]. This platform composed of three main components [63, 64]:

1. **OpenStack Compute:** Orchestrate, manage, and offer virtual machine software called "Nova".
2. **OpenStack Object Storage:** Redundant storage of static objects. The responsible software is "Swift".
3. **OpenStack Image Service:** Query and Storage for virtual disk images through the "Glance" software.

Following the approval of this project, we devised a testing plan that allowed us to compare it with other solutions and evaluate its viability for future projects at OOM in the WRF domain, while using OpenStack's components. Our goal was to test WRF on the maximum of vCPUs available, due to performance and consumption (the consumption is the same regardless of the load of an instance) aspects for an instance so that we could observe a speed-up and a time curve, and analyze which instances are the most adequate for running WRF on a daily simulation. Similarly, to the testing in Cirrus, we limit WRF to run for an hour, getting the ECT values. The available flavors can be checked on Table 6. This plan is divided into the following categories:

- **Native:** Evaluate the multiple instances available, from 1 vCPU up to 64 vCPUs, using a native-like environment with the Ubuntu 24.04 LTS RAW image (already pre-loaded in Stratus), akin to the Local Machine and Cirrus.

- **Docker:** Perform tests within the same range of vCPUs as the native execution and evaluate using the developed Ubuntu 24.04 LTS Docker image.

- **Slurm:** Develop a multi-node solution with at least 2 nodes and test how WRF performs.

Table 6. Instances available on Stratus. *For 1 single-node task tests **For 2 and 4 single-node tasks tests

Name	vCPUs	RAM (GB)	Used?
svc.xs	1	2	No
svc.s	2	4	Yes*
svc.m	4	8	Yes**
svc.l	8	16	Yes
svc.xl	16	32	Yes
svc.2xl	32	64	Yes
svc.3xl	64	128	Yes
svc.4xl	128	256	No

Before defining what type of instances were to be used, we tested the execution on all instance types at least once. We found that some instances had limitations for the tests that we intended to do, mainly on the Docker domain, due to RAM constraints. In order to draw performance comparisons, we ran the tests using the same flavor and the same instance for the same amount of tasks, eg.: For 8 tasks, we chose svc.l for both Native and Docker tests.

With the purpose of running the tests across multiple instances, we mounted volumes with the designations of WRF, GEOG, and LOGS. The objectives of these volumes when they were launched were:

- **WRF:** Store the Madeira processed data and WRF and its libraries in a compressed manner so that they can be used whenever a new instance is launched.

- **GEOG:** Store the U.S.A. geographical data. Note: Another use of this volume is discussed in the Stratus - A CNCA Cloud Platform section of this chapter.

- **LOGS:** Store in an organized manner the logs of the WRF tests of each type of environment.

In Figure 6, we present the architecture used for Native and Docker executions.

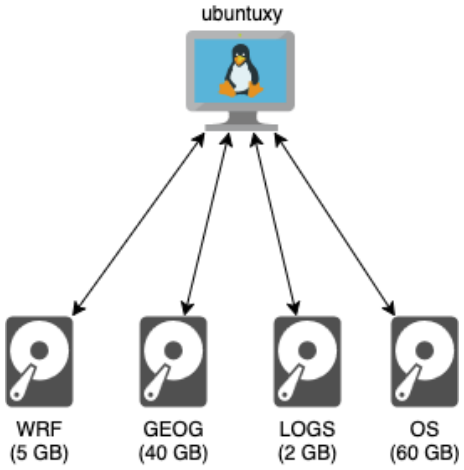

Figure. 6. Generic single-node architecture. As for the nomenclature used, xy denotes the number of CPU cores.

To make launching new instances seamless, we first compiled WRF and its libraries and packed them into the WRF volume, and geographical data was put on the GEOG volume. Then we developed a script, called `config.sh`, available in our GitHub repository[4], that allows such automation. This script creates and mounts the appropriate volumes and then extracts WRF and Madeira data to the OS volume. The required packages are installed, and a Docker container is made based on the image we developed. This script also downloads `wrf_run.sh`.

3.5.1 Stratus - Slurm

For Slurm-based workload management, we designed an architecture that enables resource sharing and job scheduling. We developed two configurations: one with a single controller and two workers, and another with a single controller and four workers. Since both setups function similarly, they were configured in the same manner. Our implementation primarily followed the guidelines provided in the **slurm-for-dummies** GitHub repository[5], authored by the Quant Club at the University of Iowa. Figure 7 presents the dual-worker architecture, while Figure 8 illustrates the quad-worker configuration. In these figures, solid lines indicate direct access, whereas dashed lines represent access through the controller.

[4] `https://github.com/ehxa/OOM-Internship`
[5] `https://github.com/SergioMEV/slurm-for-dummies`

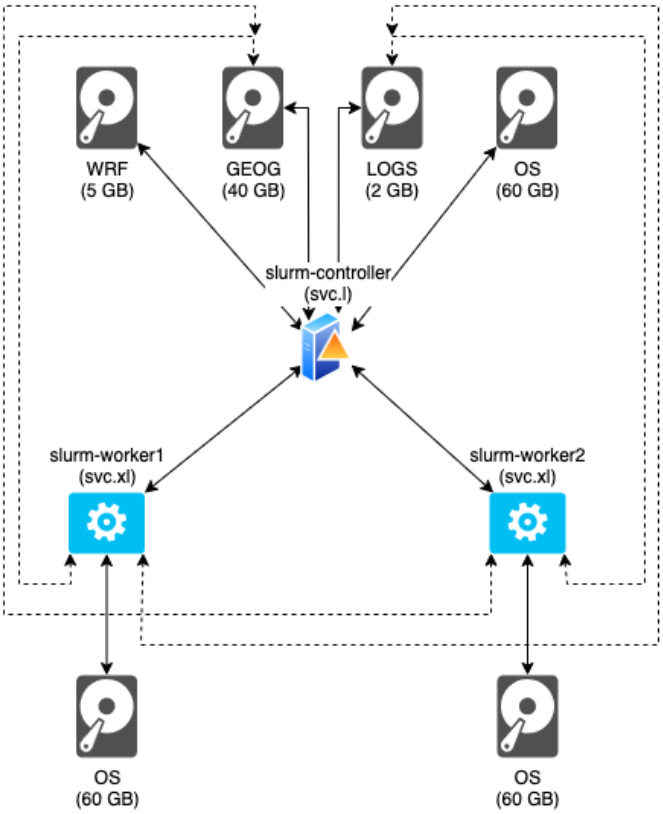

Figure. 7. Single controller, dual-worker architecture

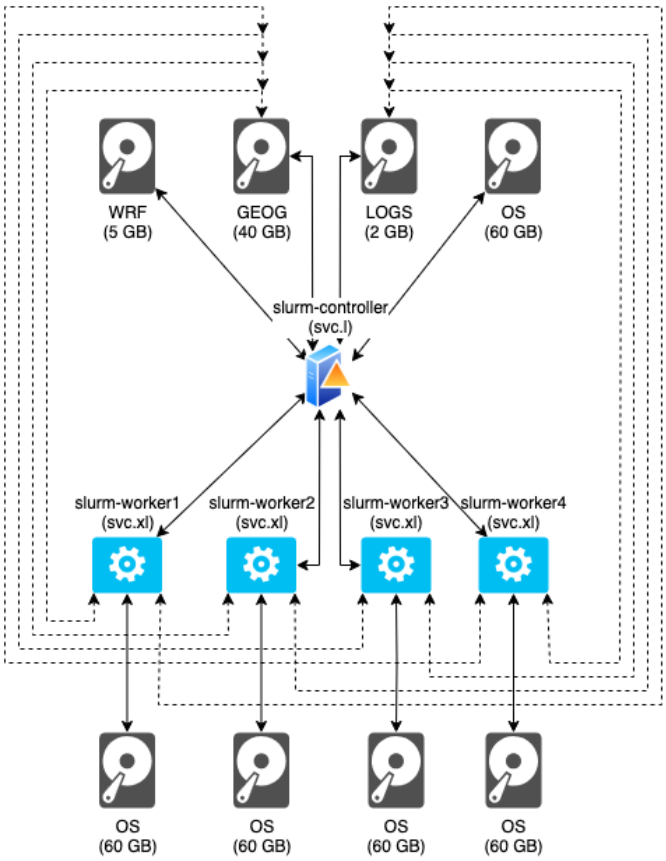

Figure. 8. Single controller, quad-worker architecture

A key distinction between our setup and the architecture depicted in Figure 7 and Figure 8 from the previous architecture from Figure 6 is the inclusion of shared volumes. Specifically, the GEOG and LOGS volumes are shared across instances. Due to OpenStack's limitations, volumes can only be attached to one instance at a time. To work around this restriction, we attached GEOG and LOGS to the Slurm controller and configured them as Network File System (NFS) volumes, allowing worker nodes to access them via NFS. These volumes are mounted on the workers using the controller's private Internet Protocol (IP) address.

Additionally, we had to determine where to store WRF and its required libraries. Given volume size constraints, we opted to extract the WRF installation onto the GEOG volume, as it provides greater capacity. This ensures that WRF remains accessible without exceeding storage limits. Notably, volume performance remains consistent regardless of size or mounting point, suggesting

that Stratus employs a shared storage mechanism within SSD speed ranges. For NFS volume mounting instructions, we followed the guidelines from the Tecmint website[6].

To streamline Slurm cluster management, we developed multiple scripts. The `mount_disks.sh` script, designed for the Slurm controller, assists in mounting disks and can be used across different WRF environments. We identified a bug related to volume mounting order. When instances are shelved and later unshelved, each time an instance is restarted, the volume order changes. To address this, our script displays the currently available volumes and prompts the user to assign the correct mounting points.

For worker nodes, the `mount_disks_worker.sh` script automates volume mounting by requesting the controller's IP and mounting the corresponding NFS shares.

To facilitate WRF execution within Slurm, we created an sbatch script called `wrf_sbatch.sh`. Similar to `wrf_run.sh`, it generates a runtime log. Additionally, to simplify job submission, we export environment variables based on the user's input, making it easier to define the number of tasks and nodes.

For configuration and monitoring, we assigned floating IPs to enable SSH access through terminal instances. Figure 9 provides an example of WRF execution and monitoring in a dual-worker setup. The top-left window runs the `wrf_sbatch.sh` script, while the bottom-left window displays the status of the current job. The remaining two windows use the `top` command to monitor resource usage and analyze WRF task distribution.

[6] `https://www.tecmint.com/install-nfs-server-on-ubuntu/`

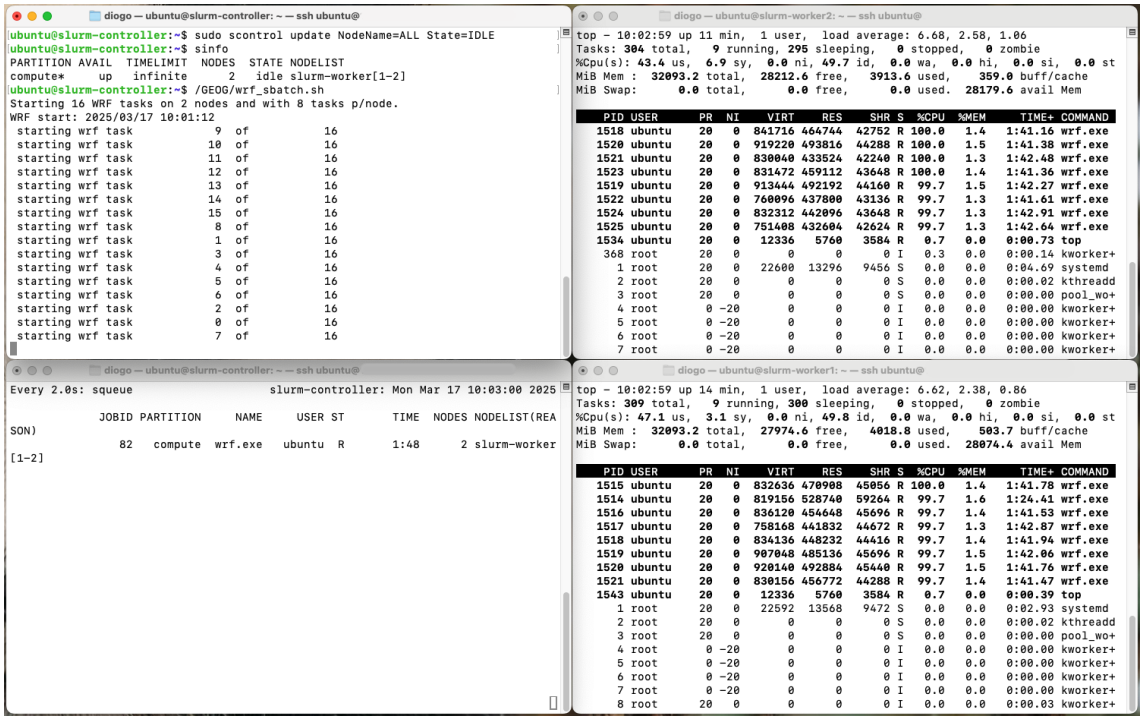

Figure. 9. Example of a WRF execution and monitoring of Stratus using Slurm

3.6 Kubernetes Solution

We developed a Kubernetes implementation that allows running the WRF model, on a common host. For this purpose, we used the integrated Kubernetes available on Docker Desktop on a personal computer. Our implementation should work similarly on other Kubernetes applications. In Table 7, we present our configurations for the development and running of this solution. This work is based on a master pod, which acts as the controller, and on a single or multi-worker pod architecture. We make both kinds of solutions. In Figure 10, we present an architectural diagram of the dual-worker node architecture.

Table 7. Host machine Docker configuration (Kubernetes)

Characteristic	Description
CPU	8
Memory limit	6 GB
Swap	1 GB
Virtual disk limit	64 GB
Docker Desktop	4.36.0
Kubernetes	v1.30.5

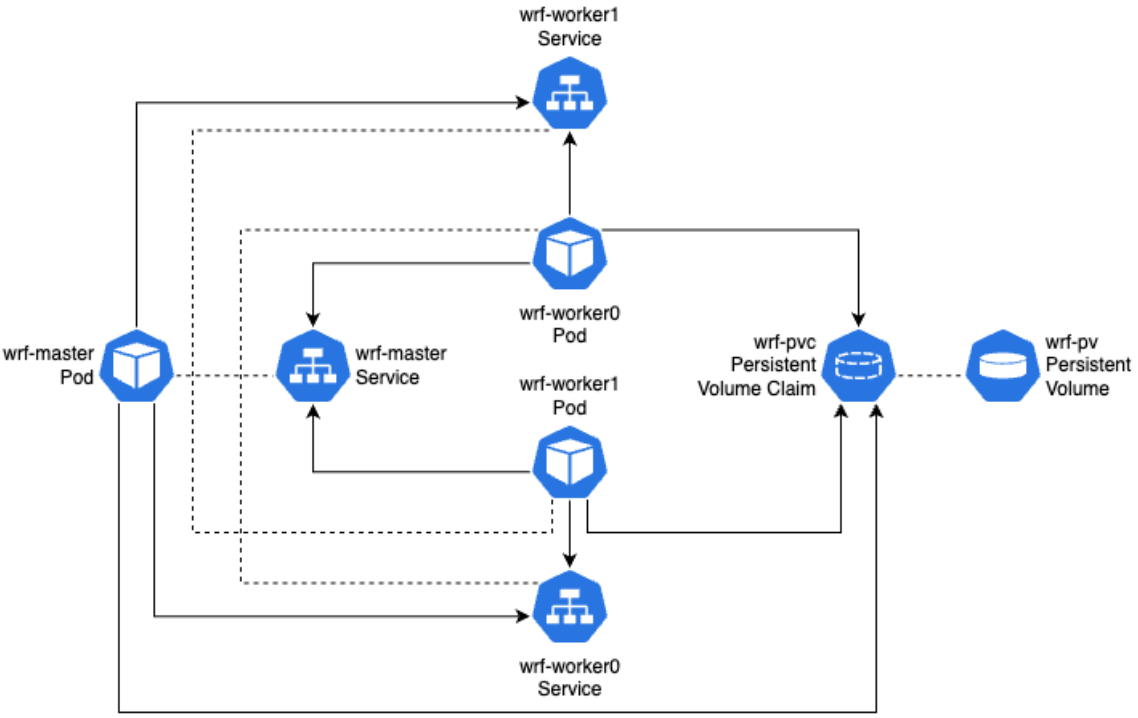

Figure. 10. Kubernetes single master, dual-worker solution architecture. The worker pods communicate with each other and with the master pod through each respective service.

Each pod has a single container with the `ubuntu24.04-gcc-20250130-1250-configured` Docker image. In order to run an MPI-based application, SSH is used over the TCP/IP protocol. Therefore, in addition to the pods deployments, we deployed a service for each. This way, the pods can communicate with each other.

Similarly to the Slurm implementation on Stratus, a common storage path is necessary for WRF executables and input/output data. An NFS volume could be similarly used in a Cloud setting as it was used before. Since this was a local solution, we used a local shared directory, where Docker Desktop with the appropriate permissions can make it available to the running pods. This way, we deployed a Persistent Volume and a Persistent Volume Claim, with 5 GiB of storage. In Figure 11, the corresponding files for all deployments are displayed.

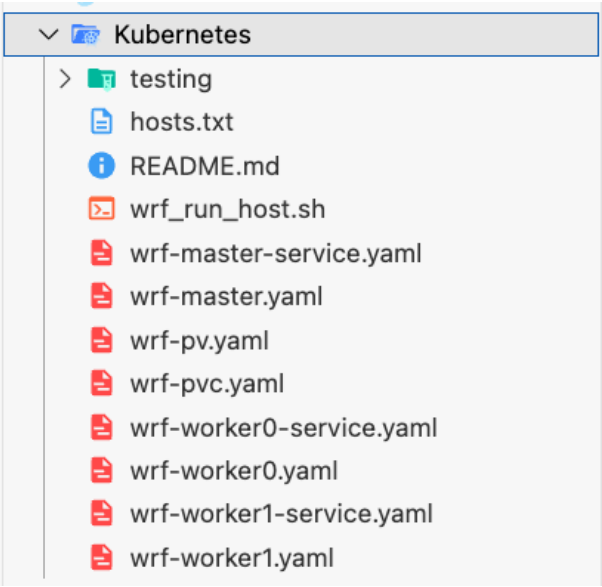

Figure. 11. File tree for the Kubernetes solution

However, since the WRF executables were in the master pod, and not on the host, we used the `ln` (link) command that makes shortcuts from the master pod to the host. As for the inputs, the user can simply change them on the shared directory, and as for the outputs, they are stored and can be checked in the same location. In Figure 12 and Figure 13, we can see an example of how this works.

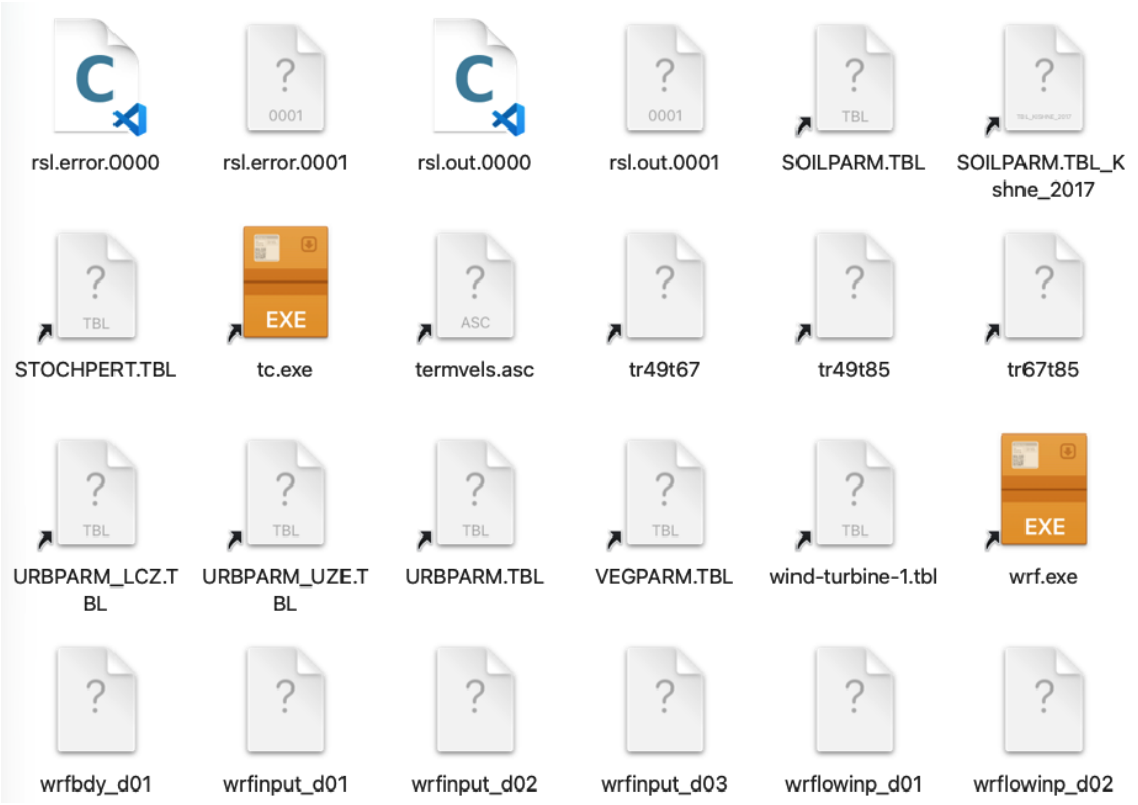

Figure. 12. Window with some of the linked files from wrf-master pod and output results on the host machine

```
rsl.out.0000:
Timing for main: time 2024-09-12_00:00:30 on domain   3:    0.75138 elapsed seconds
Timing for main: time 2024-09-12_00:00:30 on domain   2:    4.23321 elapsed seconds
Timing for main: time 2024-09-12_00:00:35 on domain   3:    1.18903 elapsed seconds
Timing for main: time 2024-09-12_00:00:40 on domain   3:    0.82667 elapsed seconds
Timing for main: time 2024-09-12_00:00:45 on domain   3:    0.78484 elapsed seconds
Timing for main: time 2024-09-12_00:00:45 on domain   2:    4.55049 elapsed seconds
Timing for main: time 2024-09-12_00:00:45 on domain   1:   41.47326 elapsed seconds
Timing for main: time 2024-09-12_00:00:50 on domain   3:    0.97442 elapsed seconds
Timing for main: time 2024-09-12_00:00:55 on domain   3:    0.78900 elapsed seconds
Timing for main: time 2024-09-12_00:01:00 on domain   3:    0.62949 elapsed seconds
Timing for main: time 2024-09-12_00:01:00 on domain   2:    4.14983 elapsed seconds
Timing for main: time 2024-09-12_00:01:05 on domain   3:    6.74237 elapsed seconds
Timing for main: time 2024-09-12_00:01:10 on domain   3:    0.88159 elapsed seconds
Timing for main: time 2024-09-12_00:01:15 on domain   3:    0.86177 elapsed seconds
Timing for main: time 2024-09-12_00:01:15 on domain   2:   10.16248 elapsed seconds
diogo@Mini-de-Diogo Kubernetes %
```

Figure. 13. Kubernetes execution, using previously linked files, and output results for a 60-second, 2-task execution using Madeira data

With this, we need to set up the SSH communication within the pods. The master needed a public SSH key that was copied to the worker pods. Since the worker pods are effectively the ones

that run the WRF model, they both act as clients and servers. Therefore, they require the SSH public keys to be copied to each other and have the `openssh-server` package installed. Having done this part, everything is ready to run WRF. For ease of use purposes, we developed a script called `wrf_run_host.sh` specific to this Kubernetes implementation, which can be run directly on the host terminal. In this script, the user can set the number of desired tasks, running time, and check the output results after running the WRF model. In Figure 14, an example of a 4-task dual-worker execution is shown.

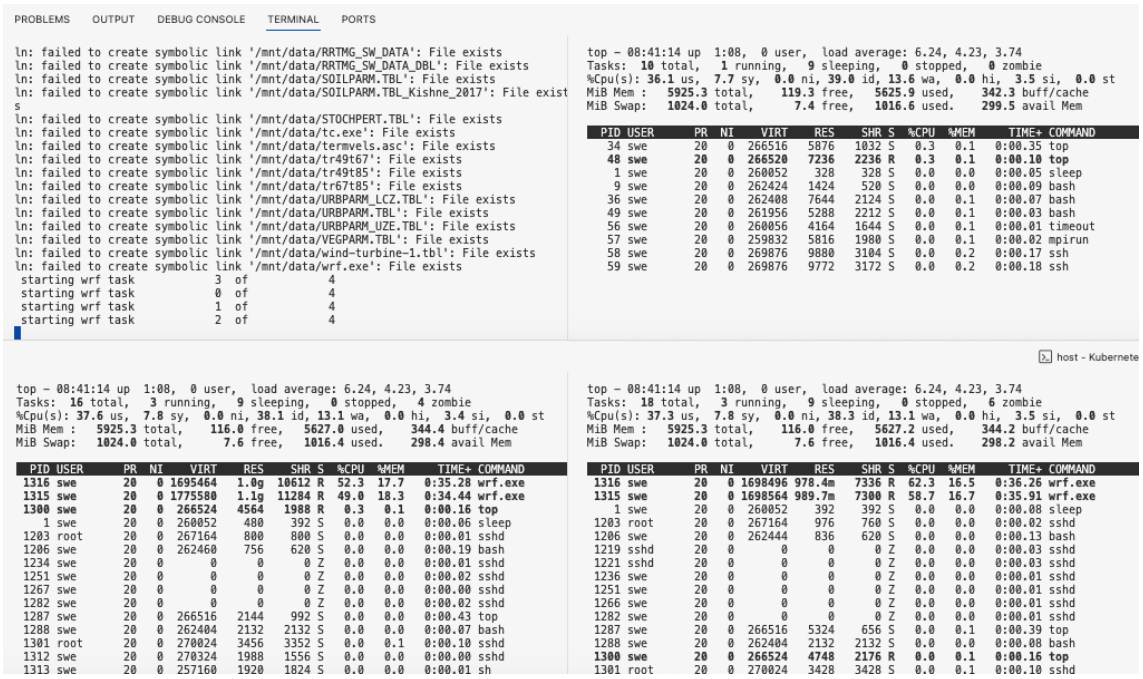

Figure. 14. Kubernetes WRF example execution with 4 tasks (2 tasks per worker using Madeira data). **Top-left: host machine** (files were already linked before); **top-right: wrf-master pod** - starts the WRF execution, distributing the 4 tasks to the 2 workers; **bottom-left: wrf-worker0 pod** - `top` program, showing the workload of 2 WRF tasks; **bottom-right: wrf-worker1 pod** - `top` program, showing the workload of 2 WRF tasks

3.7 Evaluating Cloud Solutions

OOM was interested in exploring CCTs and, if possible, using Lightweight Virtualization solutions with WRF through pre-configured WRF Docker images. As such, we researched two types of solutions based on the type of execution: HPC Instances and CCIs. Alternatives were discussed and proposed to the stakeholders, where the experience, ease of use, cost, and performance factors contributed to this analysis. A spreadsheet was used to calculate the necessary resources for the WRF model for a complete daily simulation of Madeira data.

The calculations for the multiple CCTs used in the proposals were made according to the prices listed on official documentation and online calculators. We gathered all the costs and taxes applied to each service parameter. To get the values for the ECTs according to the number of CPU cores, we used the gathered results. We divided our selection process into eight steps:

1. Calculate the requirements for running WRF using Madeira data;
2. Select the instances across CSPs that satisfy WRF computational requirements;
3. Get the average benchmark values using Geekbench 6 for the CPUs of the selected instances and the Stratus CPU (AMD EPYC 7501);
4. With the benchmark values, calculate the new ECTs for the previously selected instances using the relative performance and Stratus test results;
5. Select the suitable instances according to the new ECTs;
6. Calculate the associated costs for the selected instances;
7. Suggest an instance of each CSP that has the best Performance/Cost ratio;

As for the equation that is based on the Docker results and used for CSPs proposals, we gather the results of the ECTs for Stratus using Madeira data based a 1-hour WRF execution, and obtain the power approximation equation up to 64 vCPU cores on a single-node configuration.

In order to get ECTs for this equation, we test WRF 3 times on 1, 2, 4, 8, 16, 32, 48, 64 vCPUs configurations in Stratus.

We then develop a script based on the work of Oleg Legun [65], adapting it to get Geekbench 6 scores. Additionally, we gather the number of samples available on the Geekbench Browser website. Geekbench 6's tests work akin to WRF, where parallel tasks work together by CPU core [66].

With that equation, we then use the Multi-core score gathered average from a script for a certain CPU. Then, we use the Stratus CPU as the base and compare it to other CSPs CPUs, obtaining the Multi-core Score Ratio (MCSR).

This is possible due to the linear scoring used by Geekbench 6 in multi-core testing, where the double score means double the performance [66]. We verify this in the Results chapter of this document.

$$\text{MCSR} = \frac{\text{CSP Instance CPU Multi-core score}}{\text{Stratus CPU Multi-core score}} \tag{2}$$

As for the storage, we calculated the amount of storage required for input, output, and system (OS and libraries). The size of the respective files should be the same for every run, regardless of the CPU, memory, storage types, and network.

Meanwhile, we developed two questionnaires that would be used to evaluate such criteria. We created a preliminary questionnaire that allows for perceiving the most relevant and important aspects of CCTs for a Cloud WRF solution, simplifying the evaluation matrix (second questionnaire). The first one followed a five-point Likert scale [67] for 31 questions, where we evaluated for each criterion their importance: Very Unimportant, Not Important, Moderately Important, Important, and Very Important. This Criteria influence was presented as a matrix with a five-point scale: No Influence, Low Influence, Medium Influence, High Influence, and Perfect Influence. This scale was chosen according to the experience of the stakeholders and because it provides a balanced level of granularity. Due to size and visualization constraints, related documents can be checked here [7]. We did not proceed with the questionnaires, and therefore with the whole 2-Tuple DEMATEL method [40]. However, if applied, it is expected to provide us with further resourceful insights and complement the importance of certain aspects of CCTs in our Cloud proposals.

3.8 Web application

OOM presented the idea of interactively visualizing the forecasted WRF data. This would be done through a web app, allowing a user to view a variable by hour in a map as a layer. This web app could, in the future, be used as a replacement for the current web app, called Geoportal, available here [8], due to performance constraints. The goal was to develop an easy-to-maintain prototype suited to the stakeholders' requirements.

[7] `https://eu.docworkspace.com/d/cIC2NkL_mAbjnubgG?utm_source=wps_office_mac&utm_medium=&utm_content=link`

[8] `https://oomvisor.arditi.pt/`

An example of a current alternative is the interactive map available on the IPMA's website. [9] We used IPMA's web app as a reference for our prototype. The final application is expected to have similar functionality to this application in terms of UI and UX. Meanwhile, the prototype should implement some of these features. Regarding IPMA's application, the user can select 4 different variables, such as Atmospheric Pressure, Average Wind Intensity at 10 meters, and Air Temperature at 2 meters. The user can also select if they want wind tracers, and check out the forecasting data hourly (24 hours), and the respective playback speed. Additionally, the user can zoom in and zoom out. Below the map, explanations and temperature scales are provided. This web app uses Leaflet, an open-source, lightweight JavaScript library, just about 42 KB, geared for performance, usability, simplicity, and efficiency [68].

For the prototype, the standard web languages were used. The prototype was developed in HyperText Markup Language (HTML) for page structure, Cascading Style Sheets (CSS) for styling, and JavaScript for scripting, in the front-end department. However, it was necessary some data preparation to use WRF output data in the front-end. Since this data comes in a Network Common Data Form (NETCDF) format, Python comes in handy, having libraries that support NETCDF files, such as xarray. Therefore, there is a server-side component that handles NETCDF files and prepares them for the front-end in a JavaScript Object Notation (JSON) format. For server-side purposes, we used Flask, a Web Server Gateway Interface (WSGI) web application framework, and Gunicorn, an HTTP Server for Unix, both Python-based. While we did not use the data generated directly through this work, the prototype uses the data available in OOM's Thematic Real-time Environmental Distributed Data Services (THREDDS) server, which allows using specific variables on demand. The output data generated by our previous testing, such as using Docker, could be published to this server and then used on the prototype or future web application.

In Figure 15, the file tree of the prototype is presented, and its purpose of the directories/files can be described as follows:

- `.github/workflows/main.yaml`: automatic publishing for our web app in Docker Hub;
- `oomvisor-venv`: Python's virtual environment;

[9] `https://www.ipma.pt/pt/otempo/prev.numerica/index-mf2.jsp`

– `static`: Front-end styling and scripting files. The `map.js` file contains and uses Leaflet and generates a layer as well.

– `templates/index.html`: web page;

– `app.py`: server-side NETCDF data processing;

– `requirements.txt`: required Python packages;

– `wgsi.py`: makes the app WSGI compatible so that the WSGI server can use it;

A separate GitHub repository was made specifically for this prototype, and it is publicly available here [10]. Another Docker Hub repository, with the same goal, was also made available[11].

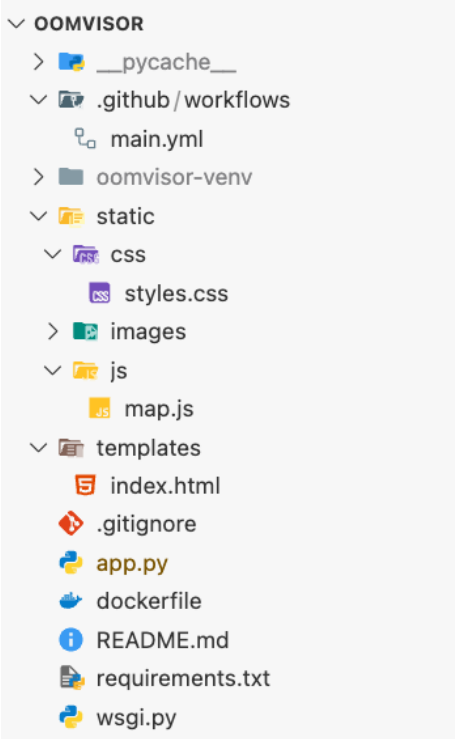

Figure. 15. Prototype's file tree.

[10] `https://github.com/ehxa/oomvisor`
[11] `https://hub.docker.com/repository/docker/ehxa/oomvisor/general`

4 Results

4.1 Background

In this section, we will present and discuss the results for Native and Docker-like solutions on the Local Machine using U.S.A. and Madeira data, and on Cirrus and Stratus using Madeira data. We will also propose CCTs for OOM based on the performance (ECT) estimations, cost, and ease of use, and present a prototype for an interactive mapping web app. Therefore, this chapter is divided as follows: **Native and Docker-Based Solutions Comparison for the Local Machine (U.S.A. data)**, where we present the testing results for a complete run using the Local Machine; **Native and Docker-Based Solutions Comparison on the Local Machine (Madeira data)**, similar to previous section but for OOM's Madeira data; **Native and udocker Solutions Comparison for Cirrus (Madeira data)**, where we analyze the differences in running natively and using a Docker Image; **Single and Multi-node Performance Comparison on Cirrus (Madeira data)**, where we check how WRF using Madeira data performs using multiple nodes; **Native and Docker-Based Solutions Comparison on Stratus (Madeira data)**, where we verify both Native and Docker approaches to running WRF on a Cloud platform; **Multi-node Solution on Stratus using Slurm (Madeira data)**: using Slurm, we evaluate if it is worth running WRF on multiple nodes; **Cloud Solutions**, where we present our proposals for running WRF on the Cloud; **Web Application Prototype**, where we discuss our developed web interactive mapping app.

4.2 Native and Docker-Based Solutions Comparison for the Local Machine (U.S.A. data)

In this topic, we tested using NCEP's data, mainly to evaluate if our WRF installation was done correctly. According to the results in Figure 16 and Table 8 for the average ECT, we found that running WRF natively was the fastest alternative, as expected. However, among the Docker solutions, the results change according to the number of requested CPU cores (tasks). One typical pattern observed in all configurations was that WRF was the slowest with 1 CPU core. With 2 CPU cores, WRF runs the best, with 3 CPU cores slower than 2 and 4. This is possibly due to the Local Machine's CPU being dual-core, using 3 and 4 CPU cores through Hyper-Threading. But, apart from the Native solution, the Ubuntu container was the fastest alternative, whichever the

CPU core count. With 1 CPU core, AlmaLinux was slower than the other two Docker solutions, but it was faster with 2, 3, and 4 CPU cores compared to I-WRF.

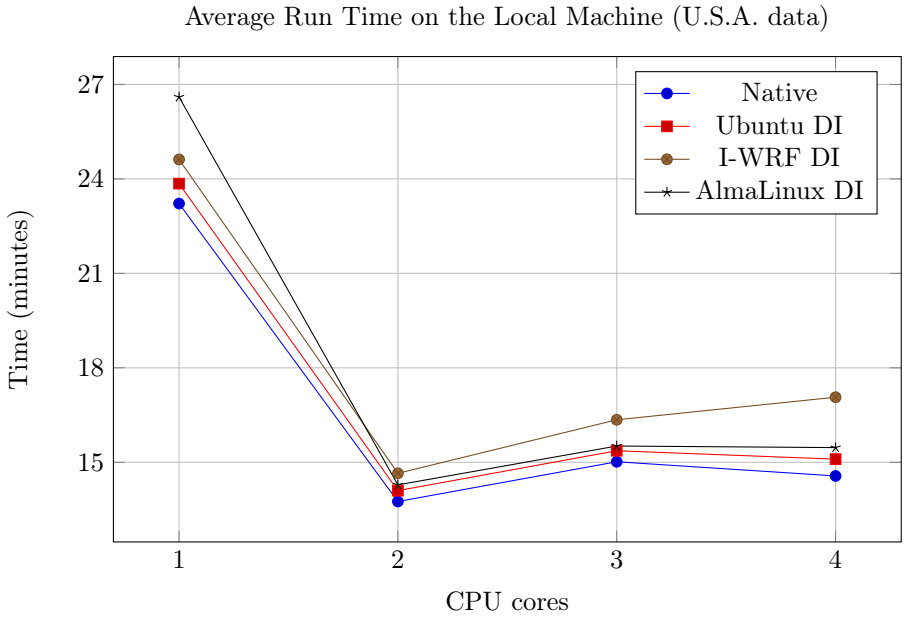

Figure. 16. Comparison of the Average Run Time for different configurations on a Local Machine using U.S.A. data. DI: Docker Image.

Table 8. Comparison of the Average Run Time for U.S.A. data on the Local Machine. DI: Docker Image.

CPU cores	Native (hh:mm:ss)	Ubuntu DI (hh:mm:ss)	I-WRF DI (hh:mm:ss)	AlmaLinux DI (hh:mm:ss)
1	00:23:13	00:23:51	00:24:37	00:26:36
2	00:13:47	00:14:06	00:14:39	00:14:17
3	00:15:01	00:15:22	00:16:21	00:15:31
4	00:14:34	00:15:06	00:17:04	00:15:28

4.3 Native and Docker-Based Solutions Comparison on the Local Machine (Madeira data)

The tests done on this section aimed to avail how Docker-based solutions run compared to the Native solution. Furthermore we select the Docker image with the best overall results, for further endeavors on the Cirrus and Stratus. According to the results registered using Madeira data in

Figure 17 and Table 9, the Native environment had the lowest values for ECT. At the same time, Ubuntu proved to be the fastest regardless of the number of CPU cores in the Docker domain. AlmaLinux had similar results to Ubuntu in the 3 CPU cores benchmark, and I-WRF was the fastest with 4 CPU cores.

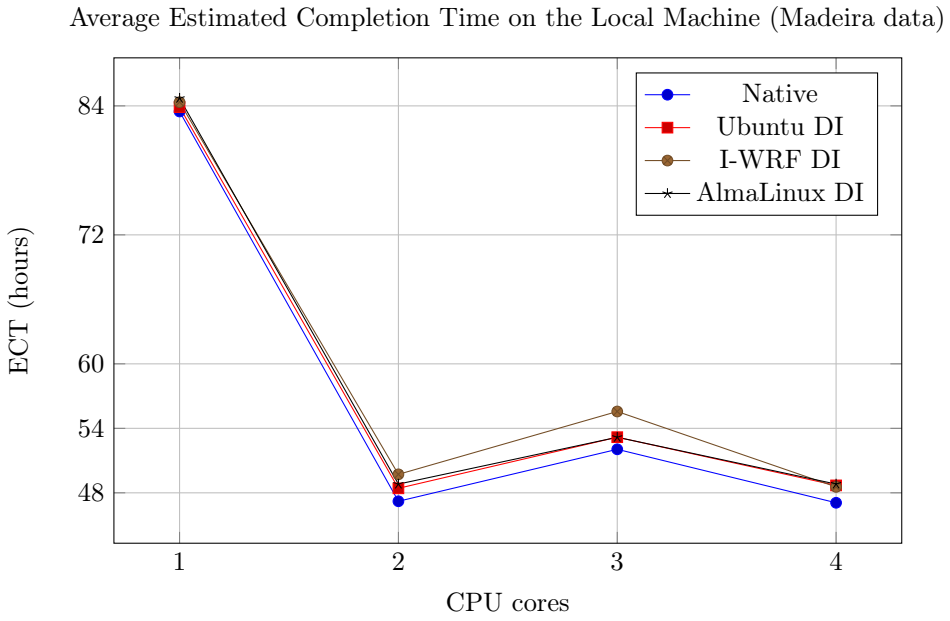

Figure. 17. Comparison of the Average Estimated Completion Time on the Local Machine based on 1 hour execution of Madeira data. DI: Docker Image

Table 9. Comparison of the Average Estimated Completion Time for Madeira data on the Local Machine based on 1 hour execution of Madeira data. ECT: Estimated Completion Time; DI: Docker Image.

CPU cores	**Native ECT (hh:mm:ss)**	**Ubuntu DI ECT (hh:mm:ss)**	**I-WRF DI ECT (hh:mm:ss)**	**AlmaLinux DI ECT (hh:mm:ss)**
1	83:28:42	83:53:15	84:20:44	84:42:21
2	47:12:55	48:25:10	49:43:12	48:48:49
3	52:02:57	53:10:34	55:34:02	53:10:13
4	47:04:29	48:41:26	48:33:06	48:48:49

4.4 Native and udocker Solutions Comparison for Cirrus (Madeira data)

Our test setup, despite the limitation of 12 CPU cores for udocker, provided a glimpse into the potential benefits of using Lightweight Virtualization technologies. The results in Figure 18 and Table 10 show that the ECTs were the same for an equal number of CPU cores, showing no signs of overhead, with 12 and 24 CPU cores being the exception due to the limitation of the udocker CPU cores. According to our results, the ideal number of CPU cores for udocker on Cirrus is 10.

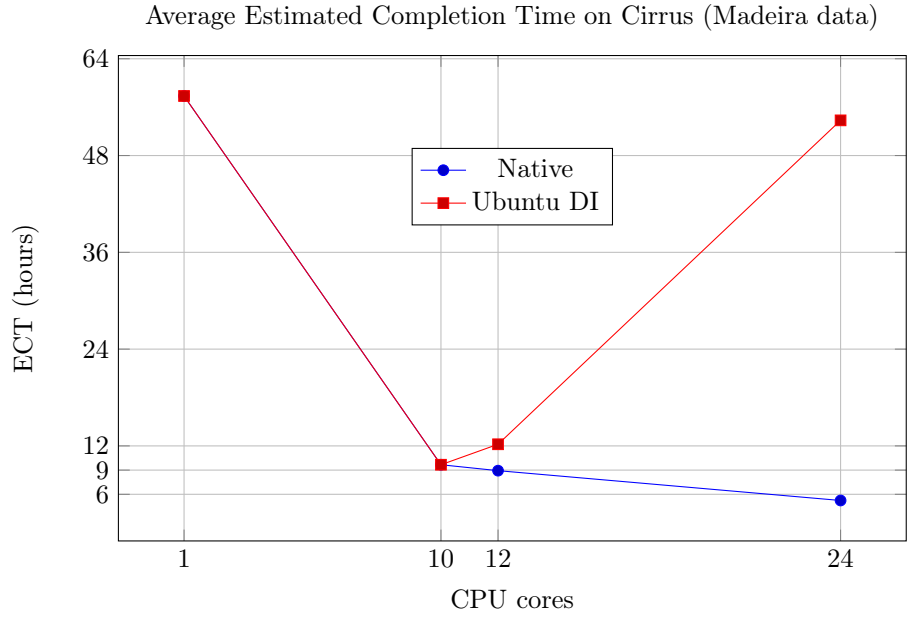

Figure. 18. Comparison of the Average Estimated Completion Time on Cirrus based on 1 hour execution of Madeira data. DI: Docker Image

Table 10. Comparison of the Average Estimated Completion Time between Native and udocker on Cirrus based on 1 hour execution of Madeira data. ECT: Estimated Completion Time; DI: Docker Image.

CPU cores	Native ECT (hh:mm:ss)	Ubuntu DI ECT (hh:mm:ss)
1	55:23:05	55:23:05
10	09:39:52	09:39:52
12	08:55:49	12:12:12
24	05:24:30	52:21:49

4.5 Single and Multi-node Performance Comparison on Cirrus (Madeira data)

The test results demonstrate a clear correlation between node count and ECT. As shown in Figure 19 and Table 11, higher node counts generally yield increased ECT values for identical task loads. Consequently, our findings favor configurations with lower node counts. The single-node implementation emerges as the optimal choice in nearly all scenarios, with the sole exception being the 12-task configuration, where alternative node counts may prove more efficient. We would also like to note that the following combinations were impossible to run: 4 nodes with 3 tasks each (12 tasks total); 2 nodes with 12 tasks each (24 tasks total); 6 nodes with 4 tasks each (24 tasks total).

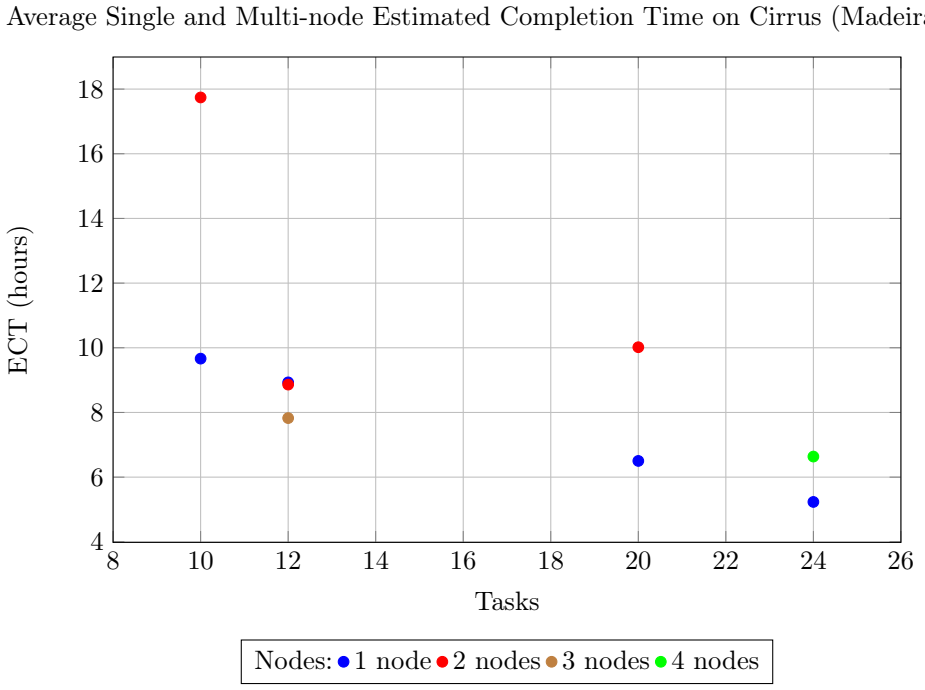

Figure. 19. Comparison of the Average Estimated Completion Time on Cirrus based on 1 hour execution of Madeira data.

Table 11. Comparison of the Estimated Completion Time based on 1 hour execution of Madeira data. ECT: Estimated Completion Time.

Nodes	**Tasks**	**Tasks p/node**	**ECT (hh:mm:ss)**
1	**10**	10	09:39:52
1	**12**	12	08:55:49
1	**20**	20	06:30:04
1	**24**	24	05:13:58
2	**10**	5	17:44:29
2	**12**	6	08:51:42
2	**20**	10	10:01:03
3	**12**	4	07:49:34
4	**24**	6	06:38:09

4.6 Native and Docker-Based Solutions Comparison on Stratus (Madeira data)

Our evaluation of WRF on the Stratus platform examined both single-node and multi-node configurations, with this analysis focusing specifically on single-node performance comparisons between ative and Docker implementations. The results demonstrate that 48 tasks (vCPUs) represent the optimal configuration for processing Madeira data, delivering peak performance while maintaining compliance with OOM's 6-hour runtime limit.

While alternative task counts (40 and 56) also remained within acceptable time constraints in the Native environment, they provided no performance benefits. Given that all tests utilized the same svc.3xl instance type, where resource allocation remains constant regardless of task count variations, there is no practical advantage to configurations other than the identified 48-task optimum.

Performance comparisons revealed the Native implementation's general superiority, benefiting from reduced overhead despite requiring more complex setup procedures. The sole exception occurred at 64 tasks, where Docker unexpectedly outperformed its Native counterpart. Nevertheless, 48 tasks maintained their position as the optimal configuration across both environments, with only negligible performance variations observed between them, as seen in Table 12, Table 13, and Figure 20.

Table 12. Average Speedup results on Stratus (Native) based on 1 hour execution of Madeira data. ECT: Estimated Completion Time.

Tasks	Native ECT (hh:mm:ss)	Speedup	Flavor
1	108:00:00	1.00	svc.s
2	54:21:26	1.97	svc.m
4	30:09:25	3.58	svc.m
8	20:46:39	5.20	svc.l
16	10:29:08	10.30	svc.xl
32	06:18:57	17.10	svc.2xl
40	05:26:45	19.83	svc.3xl
48	05:08:01	21.04	svc.3xl
56	05:17:21	20.29	svc.3xl
64	06:32:44	16.50	svc.3xl

Table 13. Average Speedup results on Stratus (Ubuntu DI) based on 1 hour execution of Madeira data. ECT: Estimated Completion Time.

Tasks	Ubuntu DI ECT (hh:mm:ss)	Speedup	Flavor
1	108:00:00	1.00	svc.s
2	72:36:18	1.84	svc.m
4	39:21:44	3.40	svc.m
8	24:22:20	5.50	svc.l
16	11:42:26	11.44	svc.xl
32	06:34:22	20.38	svc.2xl
48	05:19:19	25.37	svc.3xl
64	05:23:24	24.85	svc.3xl

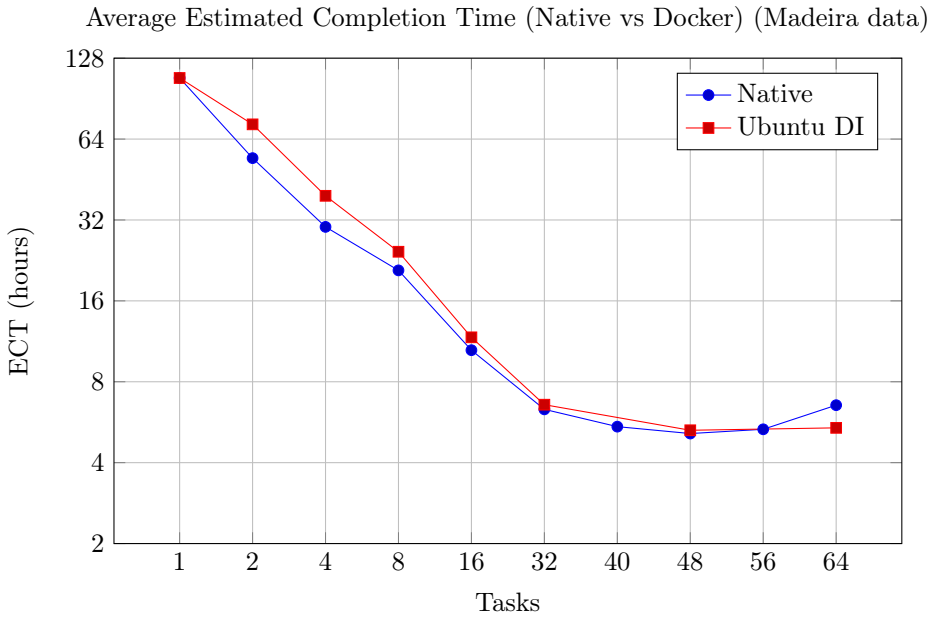

Figure. 20. Comparison of the Average Estimated Completion Time between Native and Docker executions (logarithmic scale) based on 1 hour execution of Madeira data. DI: Docker Image.

Hardware analysis suggests that at 48 tasks, the AMD 7501 CPU may be leveraging Hyper-Threading capabilities as it potentially operates beyond its physical core count. However, Stratus's support for dual-CPU configurations prevents definitive confirmation of this hypothesis.

We observe (see Figure 21) that per Amdahl's Law, diminishing returns occur the higher the number of tasks is set (number of used CPU cores), particularly when running natively. The performance degradation observed at higher task counts (56 and 64), on this type, can result from inter-CPU communication challenges rather than Hyper-Threading limitations.

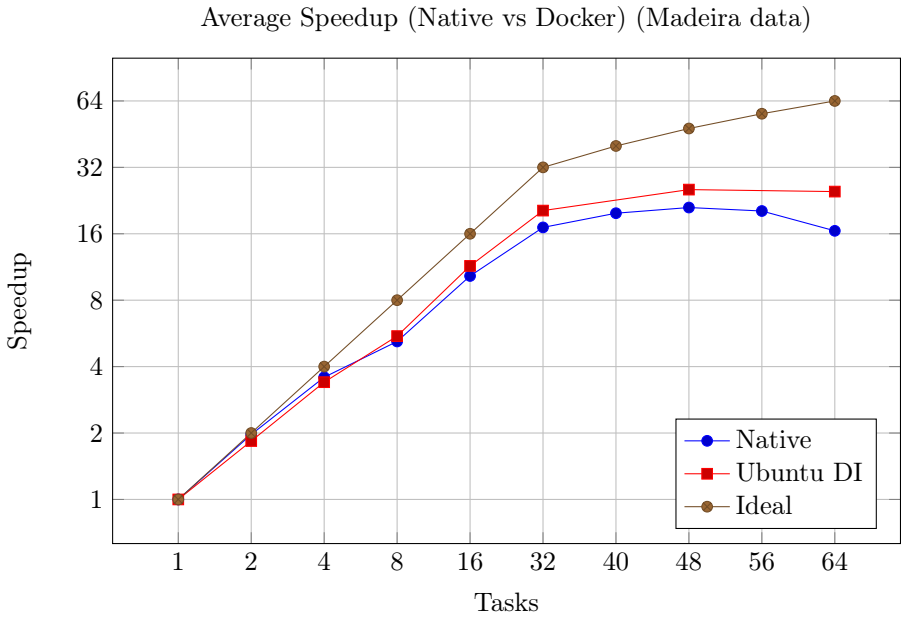

Figure. 21. Comparison of the Average Speedup between Native and Docker executions (logarithmic scale) based on 1 hour execution of Madeira data. DI: Docker Image.

Based on these comprehensive results, we strongly recommend adopting the 48-task configuration for all single-node WRF executions on Stratus, regardless of the implementation environment. This configuration delivers consistent, optimal performance while respecting all operational constraints.

4.7 Multi-node Solution on Stratus using Slurm (Madeira data)

Our analysis of multi-node configurations demonstrates a clear trade-off between node count and performance metrics. As observed across multiple tests, increasing the number of nodes leads to longer ECTs, a pattern consistent with our previous findings on the Cirrus cluster. This performance degradation, which can be attributed to inter-node communication overhead, suggests that fewer nodes generally yield better results, as seen in Table 14, and in Figure 22.

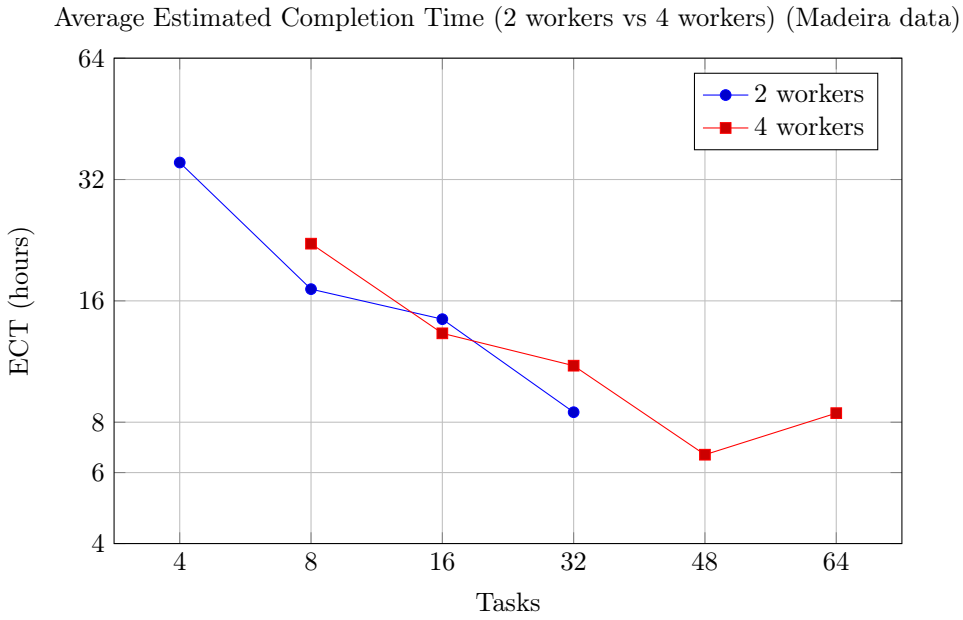

Figure. 22. Comparison of the Average Estimated Completion Time between 2 workers and 4 workers configurations (logarithmic scale) based on 1 hour execution of Madeira data.

Table 14. Comparison of the Average Estimated Completion Time for Madeira data on Stratus (Slurm). ECT: Estimated Completion Time.

Workers	**Tasks**	**Tasks p/node**	**ECT (hh:mm:ss)**
2	**4**	**2**	35:15:55
2	**8**	**4**	17:06:32
2	**16**	**8**	14:24:43
2	**32**	**16**	08:28:14
4	**8**	**2**	22:10:56
4	**16**	**4**	13:17:32
4	**32**	**8**	11:02:55
4	**48**	**12**	06:38:43
4	**64**	**16**	08:25:31

The evaluation revealed that a dual-node configuration using svc.l instances provides adequate performance for WRF execution with Madeira data, though it's worth noting that ECT values approach the critical 6-hour threshold. Even when scaling up to the maximum tested configuration of 32 tasks across dual nodes, we observed performance limitations comparable to single-node

setups. While a quad-node arrangement did show marginal improvements at higher task counts (notably with 48 tasks), these gains were insufficient to justify the added complexity, particularly given that the model still failed to meet operational requirements.

From a cost perspective, the quad-node configuration proves particularly inefficient. The cumulative resource expenditure for four svc.l instances equals that of a single svc.3xl instance, yet delivers no meaningful performance advantage. This cost parity, combined with the operational limitations, reinforces the conclusion that simpler configurations are preferable. Based on these comprehensive findings, we recommend single-node implementations as the optimal solution for WRF execution, offering the most effective balance between computational performance and resource efficiency.

4.8 Cloud Solutions

With the aim of delivering the execution of the WRF model in a Cloud environment, we identified four capable of providing the necessary resources for a potential implementation.

We divided our proposals into two types of solutions:

- **HPC Instances**: AWS, Azure, GCP
- **CCI Solutions**: OCI

The goal in both is to run the Ubuntu 24.04 Docker image.

We designed an architecture that addresses the implementation requirements across multiple CSPs (see Figure 23).

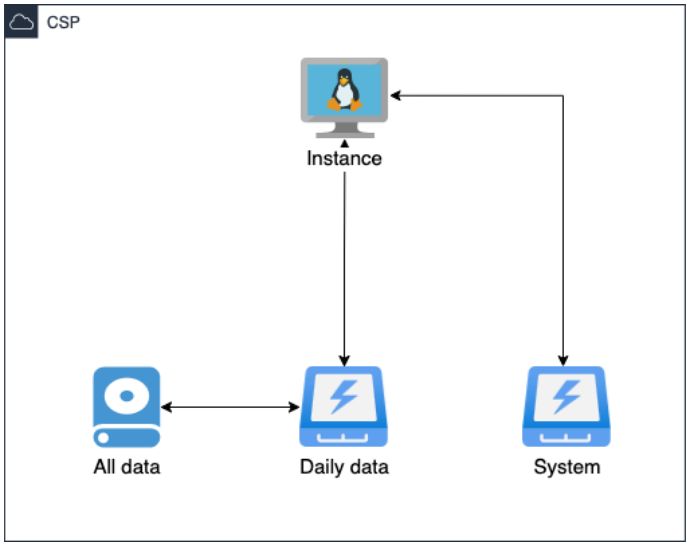

Figure. 23. Generic Cloud architecture.

Based on our previous tests on Cirrus and Stratus, the best performance is achieved with a single-node (instance). Therefore, we suggest a single-node running WRF for computation. Additionally, three storage solutions are necessary: two high-performance solutions for system storage (including OS, WPS, WRF, and libraries), daily input data, and WRF-generated forecast outputs, and a third solution focused on resilience for long-term data storage.

As for the equation that is based on the Docker results on Stratus and used for CSPs proposals, we gathered the results for Stratus using Madeira data of a 1-hour WRF execution, and obtained the power approximation equation up to 64 cores on a single-node configuration.

$$y = 125.8056x^{-19.44} \tag{3}$$

To obtain the final ECTs (y) for our proposals, we use the equation, replacing x with the number of CPU cores, and then dividing the whole result by the MCSR:

$$y = \frac{125.8056x^{-19.44}}{\text{MCSR}} \tag{4}$$

But before proceeding with the ECT calculations (which will influence the average monthly cost of our proposals), we verify if this whole process was correct. To verify this, we used our defined

formula 4 to estimate the values of ECT for the Cirrus CPU, and compared them to the actual values of our past data, i.e., by comparing the ECTs based on the score adjustment (MCSR) between the Stratus's CPU, AMD EPYC 7501, and Cirrus's CPU, AMD EPYC 7643, and the actual values.

For 24 and 30 CPU core counts, we have that as per Table 15, the values are very similar, having a difference of 00:12:04 for 24 CPU cores and 00:02:54 for 30 CPU cores.

Table 15. ECT values comparison: Formula vs Actual values. ECT: Estimated Completion Time; MCSR: Multi-core Score Ratio.

CPU cores (x)	**Stratus CPU Multi-core score**	**Cirrus CPU Multi-core score**	**MCSR**	**Formula ECT (y)**	**Actual ECT**	**ECT abs. dif.**
24	2602	4812	1.85	05:11:04	05:13:58	00:02:54
30				04:19:38	04:31:42	00:12:04

Having validated the formula, we calculated the ECT values based on the CPU core count and the Geekbench 6 multi-core score, for selected instances, allowing us to calculate the average monthly cost for the instances. In Table 16 we present the best options out of the studied, including the supporting storage solutions, average monthly costs, and ECTs.

Table 16. CCTs proposal - Best instance by each CSP. *Average ECT from previous testing on Stratus using Docker. CCT: Cloud Computing Technology; ECT: Estimated Completion Time; CCT: Cloud Computing Technology.

Service	AWS	Azure	GCP	OCI	INCD (Stratus)
Instance	hpc7a.96xlarge	HB176rs v4	h3-standard-88	CI.Standard.A1.Flex	svc.3xl
Cores	96	176	88	125	48
CPU	AMD EPYC 9R14 [69]	AMD EPYC 9V33X [70]	Intel Xeon 8481C [71]	Ampere Altra Q80-30 [72]	AMD EPYC 7501 [73]
OS and Libraries Storage	EBS	Premium SSD v2 (Managed Disks)	Boot Disk (Block Volume)	Ultra high performance	Cinder
Daily data Storage	EOZ	Premium SSD v2 (Managed Disks)	Local SSD	Ultra high performance (Block Volume)	Cinder (Block Volume)
All Input and Output data Storage	S3 Standard	Standard HDD (Managed Disks)	Standard Storage	Infrequent access (Object Storage)	Cinder (Object Storage)
ECT (hh:mm:ss)	00:25:37	00:14:11	00:48:54	00:50:31	05:19:19*
Avg. Monthly Cost (EUR)	250.05	299.95	195.94	125.70	N/A

AWS, Azure, and GCP offer their best (out of the studied) solutions in an instance with x86-64 CPUs (see Table 16). Meanwhile, OCI delivers a highly flexible range of CPU cores selection, in a ARM-based CPU, Ampere Altra Q80-30. While we initially considered a CI.Standard.E4.Flex instance in OCI, based on the AMD EPYC 7J13 (x86-64), Geekbench 6 scores, shown worse performance than Stratus's AMD EPYC 7501, therefore, it was discarded.

As for our suggestion to the stakeholders, all options available in Table 17, can be deemed viable, according to their requests. We present different average, ECTs, and CSPs, which are to be availed by them. However, we can extrapolate the best choices in terms of ECT x Avg. monthly cost ratio. The graph of the Figure 24 can be generated for better data visualization and ideal instance selection purposes. The points on the left and bottom sides of the graph are preferred. As for AWS, the best instance could be hpc7a.96xlarge, based on the AMD EPYC 9R14 CPU and with 192 cores, offering an average monthly cost of 250,05€ and a ECT of 00:25:37. As for Azure, the AMD EPYC 9V33X based HB176rs v4 instance with 176 cores, could have the best ratio, with an ECT of 00:14:11 and a average montly cost of 299.95€. As for GCP, the h3-standard-88 instance, powered by an Intel Xeon 8481C CPU and 88 cores, could be the best fit. Lastly, as for OCI, a CI.Standard.A1.Flex instance with 125 cores, an ECT of 00:50:31, and an average monthly cost of 125.70€ could be the best option for this CSP. If there is no preference for a specific CSP, there could be two suggestions in terms of ECT/Average monthly cost ratio:

1. **OCI CI.Standard.A1.Flex (125 cores)**: Offers the best ratio, but there's the concern of it being based on the Ampere's CPU having an ARM architecture, which means further development and testing are required. While we have prepared an ARM Ubuntu Docker image, we cannot assure that it will work successfully at this moment. Another concern is the number of samples of the Geekbench 6 results, being just 2 available at the time of writing this document. There is a Geekbench 5 result by Der8auer [74], citing the doubling of power efficiency, but the sample data remains low. If there is interest among the stakeholders in further research, this could be a solution with untapped and yet-discovered potential for running WRF simulations.

2. **GCP h3-standard-88**: Offers the best ratio on the x86-64 domain. While it does not have a ratio as good as the OCI's alternative, it offers a more ready-to-go solution, meaning that the developed Ubuntu Docker image could be used as is. Furthermore, the wider number of

sample tests available means that the Geekbench 6 results could be more reliable ($241 > 2$), making it a more risk-free alternative.

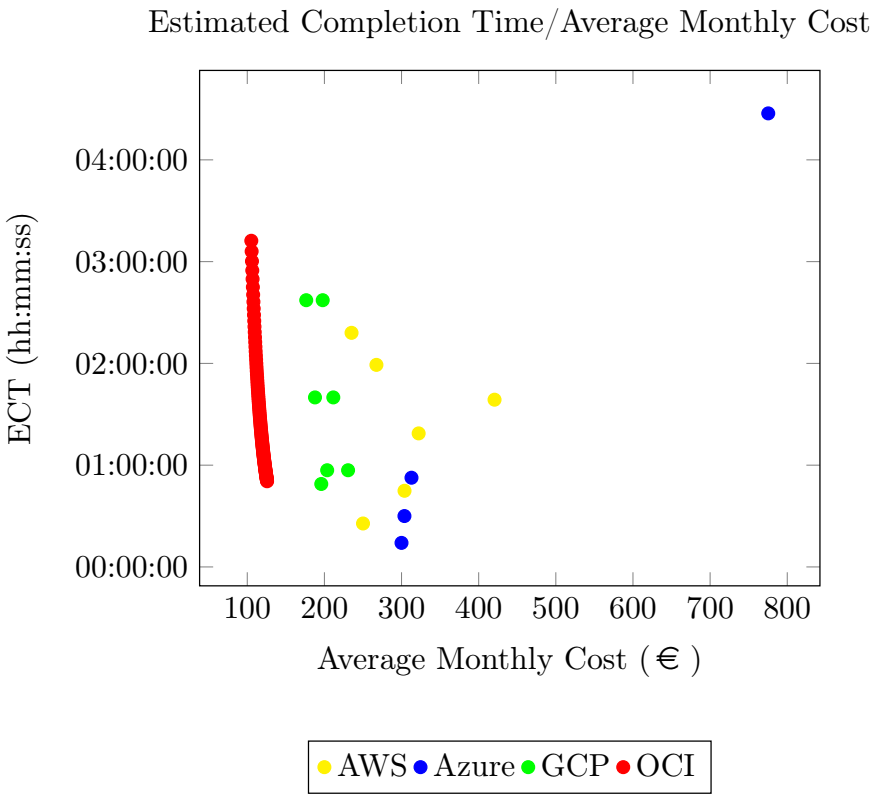

Figure. 24. Comparison of the Average Estimated Completion Time and Average Monthly Cost ratios between different CSPs. CSP: Cloud Service Provider.

Table 17. CCTs proposal - Multi-core scores. CCT: Cloud Computing Technology.

CSP	Instance	CPU	Cores	Samples	Multi-core score	Multi-core score dif.
CNCA	svc.3xl	AMD EPYC 7501 [42]	48	66	2602	1
AWS	hpc6a.48xlarge	AMD EPYC 7R13 [75]	96	1525	5677	2.18
	hpc6id.32xlarge	Intel Xeon 8375C [76]	64	814	6859	2.64
	hpc7a.48xlarge	AMD EPYC 9R14 [69]	96	463	10845	4.17
	hpc7a.96xlarge		192			
	hpc7a.24xlarge		48			
	hpc7a.12xlarge		24			
Azure	HB60rs	AMD EPYC 7V73X [77]	60	8	13548	5.20
	HB120rs v3		120			
	HB176rs v4	AMD EPYC 9V33X [70]	176	9	21019	8.08
	HC44rs	Intel Xeon Platinum 8168 [78]	44	719	3426	1.32
GCP	c2d-standard-32	AMD EPYC 7B13 [79]	32	2268	7540	2.90
	c2d-standard-56		56			
	c2d-standard-112		112			
	c2d-highcpu-32		32			
	c2d-highcpu-56		56			
	c2d-highcpu-112		112			
	h3-standard-88	Intel Xeon 8481C [71]	88	241	10685	4.11
OCI	CI.Standard.A1.Flex	Ampere Altra Q80-30 [72]	16	199	1146	0.44

4.9 Web Application Prototype

Following the requirements of the stakeholders, we developed a prototype that previews a possible implementation for the web application. In Figure 25, we have a screenshot of this prototype. Here, we can check the visualization of the Temperature at 2 meters for the 25th of May 2025. The data is selectable from the day before up to the available data in the calendar input. Step controls are present, allowing going back or forward in the forecasting of the selected date. A play button is present as well, allowing the user to play 24 hours of data automatically. Both in manual and automatic modes, the data are divided into 1-hour periods. If the user clicks the Air temperature (2 m) button, that temperature layer disappears or appears, depending on the previous state. Zoom controls allow the user to zoom in or out to a certain degree. Additionally, a temperature scale guide is provided on the bottom right, ranging from -28ºC up to 48ºC. This prototype provides a glimpse into a possible future implementation of a web app and allows for visualization of WRF-generated data interactively.

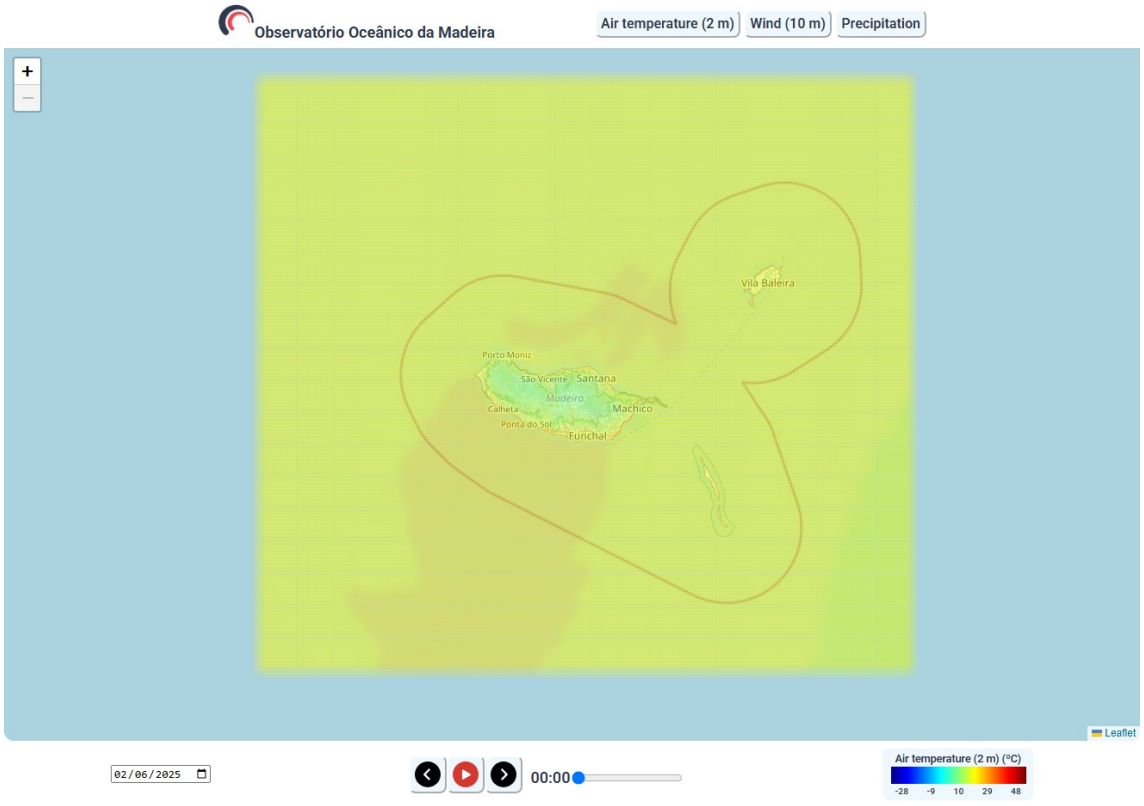

Figure. 25. Implemented interactive mapping web app prototype, showing the Air temperature (2 m) layer for the 00:00 of the 2nd of June 2025.

5 Conclusions

Our work involved testing across multiple platforms and environments. The main goal was to evaluate a Docker solution for working with the WRF model due to its ease of use, maintenance, and portability. Additionally, based on these findings, we propose CCTs to OOM.

On our Local Machine, we found that Docker, regardless of the base OS for the image, had a slight overhead when compared to the Native solution. Regarding Docker images, the Ubuntu 24.04 LTS-based image generally had the best performance, when compared to the Almalinux 8.10-minimal and NCAR's I-WRF Ubuntu 22.04 LTS-based image.

We then compared on a HPC cluster environment, Cirrus, a Native installation, and the one we developed using the Docker image. This environment had the particular limitation of allowing up to 12 cores for a Docker (udocker) container. With 1 and 10 CPU cores, the results were the same; however, when going with 12 cores or higher, the results were poorer, due to the mentioned limitation. Besides this, we tested on higher core counts natively, which allowed us to gather insights into the shared resource model applied in Cirrus and the Slurm Workload Manager, namely, in running WRF in multiple nodes. We found that 24 cores on a single-node were sufficient for a daily execution within the maximum runtime defined by the stakeholders (05:24:30 $<$ 06:00:00), but there could be concerns for the waiting time due to resource sharing. Multiple nodes did not meet such criteria, and we verified that lower node counts are preferable.

We then applied and gained access to the Stratus cloud platform offered by CNCA. This platform gave us insights into evaluating its capabilities for the WRF model. We did testing on multiple fronts. We used Ubuntu 24.04 LTS virtual machines, where we similarly compared native and Docker executions, and used Slurm for evaluating multi-node solutions. With this, we found that 48 cores on a single-node were the optimal core setup on both configurations.

Using the baseline of 24 cores found on Cirrus and the Docker results of Stratus, we searched and proposed CCTs for running WRF on other CSPs. These CSPs are AWS, Azure, GCP, and OCI. Using the Geekbench 6 Multi-core score average for the CPU of each instance, and evaluated against Stratus's AMD EPYC 7501 CPU Multi-core score. Using a power approximation equation based on Docker Stratus testing and the CPU performance differences, we valued the ECT and average monthly cost of CCTs. Among AWS, we recommend the hpc7a.96xlarge instance. In Azure, the

HB176rs v4 instance. In GCP the h3-standard-88 instance. And in OCI, CI.Standard.A1.Flex with a 125-core configuration. As for the best ECT/Avg. monthly cost solutions, regardless of the CSP, GCP's h3-standard-88 and OCI's CI.Standard.A1.Flex, offers the best ratio. OCI's solution has the particular caveat of being ARM-based, where further compiling and testing would be necessary. However, if fine with the stakeholders, it could surpass the other solutions present in this work. On the other hand, GCP's solution should be more ready to work, since it would work right away without further tweaking, and it is the best x86-64 solution in this work for the aforementioned ratio.

A prototype for a web application was also developed, and it uses OOM's WRF data store on a THREDDS server, using web technologies such as Leaflet and Flask. As it is, the temperature at 2 meters is working for the 24-hour period of the data, and a similar setup could theoretically work for other kinds of variables.

5.1 Limitations and Future Work

There are some aspects that we would like to further test and complement this line of work. In Cirrus, we would like to do the same kind of tests as were done on Stratus, in order to fully compare an HPC cluster to Cloud environments. However, an application for continued Cirrus access may be required. Still on the Cirrus topic, since its resources are shared, there could be problems with tests with higher CPU counts, as we observed earlier. For example, for a 30-core test, we waited 12 hours before an execution.

As for Stratus, the platform is in the process of migration. Therefore, we could not test a Kubernetes solution at the time of this document. We would like to test it out in future work and then devise further Kubernetes-based solution proposals for CCTs.

On the CCTs domain, we would like to test at least one solution, in order to gather further insights and validate our estimations. If validated, this process could be useful for other kinds of applications and researchers.

As for the web app, we would like to add support for wind and rain variables, and a color label for the ranges of these variables, including the temperature at 2 meters. Also, some testing could be necessary to evaluate the performance on multiple devices.

References

[1] M. Larson, "Numerical Modeling," in *Encyclopedia of Coastal Science*, M. L. Schwartz, Ed. Dordrecht: Springer Netherlands, 2005, pp. 730–733, doi: 10.1007/1-4020-3880-1_232.

[2] K. Yonekura, H. Hattori, and T. Suzuki, "Short-term local weather forecast using dense weather station by deep neural network," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1683–1690, doi: 10.1109/BigData.2018.8622195.

[3] G. Shainer, T. Liu, J. Michalakes, J. Liberman, J. Layton, O. Celebioglu, S. Schultz, J. Mora, and D. Cownie, "Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters," *10th LCI ICHPCC*, Jan. 2009.

[4] Google, "What is high performance computing (HPC)," [Accessed: 2024-12-02]. [Online]. Available: https://cloud.google.com/discover/what-is-high-performance-computing

[5] HPE, "What is High Performance Computing (HPC)? | Glossary," [Accessed: 2024-12-02]. [Online]. Available: https://www.hpe.com/us/en/what-is/high-performance-computing.html

[6] Amazon, "O que é cloud computing (computação em nuvem)? - Amazon Web Services," [Accessed: 2024-12-02]. [Online]. Available: https://aws.amazon.com/pt/what-is-cloud-computing/

[7] Microsoft, "What Is Cloud Computing? | Microsoft Azure," [Accessed: 2024-12-02]. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing

[8] R. Knepper, S. Pryor, B. Wineholt, M. Bukovsky, J. Lee, and X. Zhou, "The I-WRF Framework: Containerized Weather Modeling, Validation, and Verification," in *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, ser. PEARC '23. Association for Computing Machinery, 2023, pp. 206–210, doi: 10.1145/3569951.3597547.

[9] W. Hatheway, H. Snoun, H. ur Rehman, and A. Mwanthi, "WRF-MOSIT: a modular and cross-platform tool for configuring and installing the WRF model," *Earth Science Informatics*,

vol. 16, no. 4, pp. 4327–4336, Dec. 2023, doi: 10.1007/s12145-023-01136-y.

[10] UCAR, "Weather Research & Forecasting Model (WRF) | Mesoscale & Microscale Meteorology," [Accessed: 2024-12-02]. [Online]. Available: https://www.mmm.ucar.edu/models/wrf

[11] NSF, "NSF National Center for Atmospheric Research | National Center for Atmospheric Research," [Accessed: 2024-12-02]. [Online]. Available: https://ncar.ucar.edu/

[12] R. Moreno, E. Arias, D. Cazorla, J. J. Pardo, A. Navarro, T. Rojo, and F. J. Tapiador, "Analysis of a New MPI Process Distribution for the Weather Research and Forecasting (WRF) Model," *Scientific Programming*, vol. 2020, no. 1, p. 8148373, 2020, doi: 10.1155/2020/8148373.

[13] M. Duda, D. Gill, W. Wei, A. Islas, K. Werner, and J. Bresch, "wrf-model/WPS," Dec. 2024, [Accessed: 2024-12-13]. [Online]. Available: https://github.com/wrf-model/WPS

[14] UCAR, "Lesson-wps," [Accessed: 2024-12-13]. [Online]. Available: https://ral.ucar.edu/sites/default/files/public/Lesson-wps.html

[15] ——, "Compiling WRF," [Accessed: 2024-12-13]. [Online]. Available: https://www2.mmm.ucar.edu/wrf/OnLineTutorial/compilation_tutorial.php#STEP8

[16] J. Dudhia, "WRF Modeling System Overview," *WMO GURME Workshop 2015*, 2015.

[17] IBM, "O que é computação de alto desempenho (HPC)? | IBM," Dec. 2022, [Accessed: 2024-12-02]. [Online]. Available: https://www.ibm.com/br-pt/topics/hpc

[18] P. Vourlioti, S. Kotsopoulos, T. Mamouka, A. Agrafiotis, F. Nieto, C. Sánchez, C. Llerena, and S. González, "Maximizing the potential of numerical weather prediction models: lessons learned from combining high-performance computing and cloud computing," *Advances in Science and Research*, vol. 20, pp. 1–8, Mar. 2023, doi: 10.5194/asr-20-1-2023.

[19] R. G. Tiagonce, "CESGA's new FinisTerrae III," Jul. 2021, [Accessed: 2024-12-18]. [Online]. Available: https://www.cesga.es/en/cesga-actualiza-el-finisterrae/

[20] INCD, "INCD - Infraestrutura Nacional de Computação Distribuída," [Accessed: 2024-12-10]. [Online]. Available: https://www.incd.pt/?p=sobre-nos&lang=en

[21] L. Mitton, “Amdahl’s Law: Understanding the Basics,” Jul. 2023, [Accessed: 2024-12-09]. [Online]. Available: https://www.splunk.com/en_us/blog/learn/amdahls-law.html

[22] GeeksforGeeks, “Computer Organization | Amdahl’s law and its proof,” Jun. 2018, [Accessed: 2024-12-09]. [Online]. Available: https://www.geeksforgeeks.org/computer-organization-amdahls-law-and-its-proof/

[23] J. L. Hennessy, D. A. Patterson, and K. Asanović, *Computer Architecture: A Quantitative Approach*. Elsevier, 2012.

[24] F. Mora, “What is Amdahl’s Law? (Definition, Formula, Examples),” [Accessed: 2024-12-09]. [Online]. Available: https://builtin.com/articles/amdahls-law

[25] SchedMD, “Slurm Workload Manager - Overview,” [Accessed: 2024-12-18]. [Online]. Available: https://slurm.schedmd.com/overview.html

[26] M. A. Jette and T. Wickberg, “Architecture of the Slurm Workload Manager,” in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, J. Corbalán, and G. P. Rodrigo, Eds. Cham: Springer Nature Switzerland, 2023, vol. 14283, pp. 3–23, doi: 10.1007/978-3-031-43943-8_1.

[27] A. B. Yoo, M. A. Jette, and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2003, pp. 44–60.

[28] R. Shu, X. Gu, and W. Enck, “A Study of Security Vulnerabilities on Docker Hub,” in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, Mar. 2017, pp. 269–280, doi: 10.1145/3029806.3029832.

[29] D. M. Jacobsen and R. S. Canon, “Contain This, Unleashing Docker for HPC,” *CUG 2015*, 2015.

[30] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, “A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds,” in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Dec. 2017, pp. 74–81, doi: 10.1109/CloudCom.2017.40.

[31] S. P. Mullinix, E. Konomi, R. D. Townsend, and R. M. Parizi, "On Security Measures for Containerized Applications Imaged with Docker," *CoRR*, vol. abs/2008.04814, 2020, doi: 10.48550/arXiv.2008.04814.

[32] J. Gomes, E. Bagnaschi, I. Campos, M. David, L. Alves, J. Martins, J. Pina, A. López-García, and P. Orviz, "Enabling rootless Linux Containers in multi-user environments: The *udocker* tool," *Computer Physics Communications*, vol. 232, pp. 84–97, Nov. 2018, doi: 10.1016/j.cpc.2018.05.021.

[33] R. P. Signell and D. Pothina, "Analysis and Visualization of Coastal Ocean Model Data in the Cloud," *Journal of Marine Science and Engineering*, vol. 7, no. 4, p. 110, Apr. 2019, number: 4, Publisher: Multidisciplinary Digital Publishing Institute, doi: 10.3390/jmse7040110.

[34] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms," in *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, Nov. 2019, pp. 11–20, place: Denver, CO, USA, doi: 10.1109/CANOPIE-HPC49598.2019.00007.

[35] J. Noor, M. B. Faysal, M. S. Amin, B. Tabassum, T. R. Khan, and T. Rahman, "Kubernetes application performance benchmarking on heterogeneous CPU architecture: An experimental review," *High-Confidence Computing*, vol. 5, no. 1, p. 100276, Mar. 2025, doi: 10.1016/j.hcc.2024.100276.

[36] D. J. Milroy, C. Misale, G. Georgakoudis, T. Elengikal, A. Sarkar, M. Drocco, T. Patki, J.-S. Yeom, C. E. A. Gutierrez, D. H. Ahn, and Y. Park, "One Step Closer to Converged Computing: Achieving Scalability with Cloud-Native HPC," in *2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, Nov. 2022, pp. 57–70, place: Dallas, TX, USA, doi: 10.1109/CANOPIE-HPC56864.2022.00011.

[37] M. R. Hossen, V. Sochat, A. Sarkar, M. A. Islam, and D. J. Milroy, "Enabling Workload-Driven Elasticity in MPI-based Ensembles," in *2024 IEEE International Conference on Clus-*

ter Computing (CLUSTER), Sep. 2024, pp. 250–262, iSSN: 2168-9253, doi: 10.1109/CLUSTER59578.2024.00029.

[38] J. G. Powers, K. K. Werner, D. O. Gill, Y.-L. Lin, and R. S. Schumacher, "Cloud Computing Efforts for the Weather Research and Forecasting Model," *Bulletin of the American Meteorological Society*, vol. 102, no. 6, pp. E1261–E1274, Jun. 2021, doi: 10.1175/BAMS-D-20-0219.1.

[39] R. Garg, "MCDM-Based Parametric Selection of Cloud Deployment Models for an Academic Organization," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 863–871, Apr. 2022, conference Name: IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2020.2980534.

[40] G. Büyüközkan, D. Uztürk, and A. Maden, "Influential factor analysis for cloud computing technology service provider," *Technological Forecasting and Social Change*, vol. 192, p. 122531, 2023, doi: 10.1016/j.techfore.2023.122531.

[41] Statista, "Infographic: Amazon Maintains Cloud Lead as Microsoft Edges Closer," Nov. 2024, [Accessed: 2024-12-27]. [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers

[42] FCT, "Concurso de Projetos de Computação Avançada (5ªed) - A1 Acesso Desenvolvimento (lote A)," 2024, [Accessed: 2024-12-27]. [Online]. Available: https://www.fct.pt/concursos/concurso-de-projetos-de-computacao-avancada-5-ed-a1-acesso-desenvolvimento-lote-a-

[43] A. Tomar, R. R. Kumar, and I. Gupta, "Decision making for cloud service selection: a novel and hybrid MCDM approach," *Cluster Computing*, vol. 26, no. 6, pp. 3869–3887, Dec. 2023, doi: 10.1007/s10586-022-03793-y.

[44] N. Ghorui, S. P. Mondal, B. Chatterjee, A. Ghosh, A. Pal, D. De, and B. C. Giri, "Selection of cloud service providers using MCDM methodology under intuitionistic fuzzy uncertainty," *Soft Computing*, vol. 27, no. 5, pp. 2403–2423, Mar. 2023, doi: 10.1007/s00500-022-07772-8.

[45] S. Xu, S. M. Ghazimirsaeed, J. M. Hashmi, H. Subramoni, and D. K. Panda, "MPI Meets Cloud: Case Study with Amazon EC2 and Microsoft Azure," in *2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*, 2020, pp. 41–48, doi: 10.1109/IPDRM51949.2020.00010.

[46] S. P. T. Krishnan, B. Veeravalli, V. H. Krishna, and W. C. Sheng, "Performance Characterisation and Evaluation of WRF Model on Cloud and HPC Architectures," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, 2014, pp. 1280–1287, doi: 10.1109/HPCC.2014.218.

[47] Amazon, "Guidance for Building a High-Performance Numerical Weather Prediction System on AWS," [Accessed: 2024-12-05]. [Online]. Available: https://aws-solutions-library-samples.github.io/compute/building-a-high-performance-numerical-weather-prediction-system-on-aws.html

[48] Microsoft, "Azure HPC+AI for Climate and Weather Modeling (WRF) - YouTube." [Online]. Available: https://www.youtube.com/watch?v=zVQbG6vwwKA

[49] Google, "Weather Forecasting using the WRF model on Google Cloud," Mar. 2022, [Accessed: 2024-12-02]. [Online]. Available: https://cloud.google.com/blog/topics/hpc/weather-forecasting-using-the-wrf-model-on-google-cloud

[50] UCAR, "WRF v4.4 Benchmark Cases," [Accessed: 2025-01-08]. [Online]. Available: https://www2.mmm.ucar.edu/wrf/users/benchmark/v44/benchdata_v44.html

[51] V. Munhoz and M. Castro, "HPC@Cloud: A Provider-Agnostic Software Framework for Enabling HPC in Public Cloud Platforms," in *Anais do Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*, Oct. 2022, pp. 157–168, doi: 10.5753/wscad.2022.226528.

[52] L. Oana and M. Frincu, "Benchmarking the WRF Model on Bluegene/P, Cluster, and Cloud Platforms and Accelerating Model Setup Through Parallel Genetic Algorithms," in *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2017, pp. 78–84, doi: 10.1109/ISPDC.2017.24.

[53] IBM, "IBM Blue Gene Architecture," [Accessed: 2024-12-27]. [Online]. Available: https://www.nersc.gov/assets/Uploads/IBM-Blue-Gene-Architecture.10-10-11.pdf

[54] Oracle, "Run Containers in the cloud without managing servers," [Accessed: 2024-12-09]. [Online]. Available: https://www.oracle.com/cloud/cloud-native/container-instances/

[55] ——, "Discover the Features of OCI Container Instances," [Accessed: 2024-12-09]. [Online]. Available: https://www.oracle.com/cloud/cloud-native/container-instances/features/

[56] 1000minds, "Multi-Criteria Decision Analysis (MCDA/MCDM)," [Accessed: 2024-12-02]. [Online]. Available: https://www.1000minds.com/decision-making/what-is-mcdm-mcda

[57] H. Taherdoost and M. Madanchian, "Multi-Criteria Decision Making (MCDM) Methods and Concepts," *Encyclopedia*, vol. 3, no. 1, pp. 77–87, Mar. 2023, number: 1, Publisher: Multidisciplinary Digital Publishing Institute, doi: 10.3390/encyclopedia3010006.

[58] H. Cui, S. Dong, J. Hu, M. Chen, B. Hou, J. Zhang, B. Zhang, J. Xian, and F. Chen, "A hybrid MCDM model with Monte Carlo simulation to improve decision-making stability and reliability," *Information Sciences*, vol. 647, p. 119439, Nov. 2023, doi: 10.1016/j.ins.2023.119439.

[59] G. Koca and S. Yıldırım, "Bibliometric analysis of DEMATEL method," *Decision Making: Applications in Management and Engineering*, vol. 4, no. 1, pp. 85–103, Mar. 2021, number: 1, doi: 10.31181/dmame2104085g.

[60] A. Solanki and D. Sarkar, "Analysis of Internet of Things and Cloud Computing Implementation in the Construction Industry Using DEMATEL and TOES Framework," in *Energy and Infrastructure Management in Post Covid-19 Era*. Allied Publishers, Aug. 2022, googleBooks-ID: 1KGEEAAAQBAJ.

[61] J.-I. Shieh and H.-H. Wu, "Measures of Consistency for DEMATEL Method," *Communications in Statistics - Simulation and Computation*, vol. 45, no. 3, pp. 781–790, Mar. 2016, publisher: Taylor & Francis, doi: 10.1080/03610918.2013.875564.

[62] Docker, "Docker Desktop vs DIY with Docker Engine," 2022, [Accessed: 2024-12-18]. [Online]. Available: https://www.docker.com/wp-content/uploads/2022/03/Docker_Desktop_vs_DIY_datasheet_v2.pdf

[63] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: Toward an Open-source Solution for Cloud Computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, Oct. 2012, doi: 10.5120/8738-2991.

[64] K. Pepple, *Deploying OpenStack*. "O'Reilly Media, Inc.", Aug. 2011, google-Books-ID: Q4uCFZqbRNkC.

[65] O. Legun, "oleglegun/geekbench-avg-scores: Calculate average Geekbench scores," [Accessed: 2025-05-20]. [Online]. Available: https://github.com/oleglegun/geekbench-avg-scores/tree/master

[66] Geekbench, "Geekbench 6 Benchmark Internals," May 2024, [Accessed: 2025-05-22]. [Online]. Available: https://www.geekbench.com/doc/geekbench6-benchmark-internals.pdf

[67] M. Nurhasanah, S. Ameliasari, A. I. Iskandar, and F. Andria, "Strategy For Strengthening MSMEs Capabilities : Implementation Of Digital Marketing As A Means Of Promotion And Commercialization Of Business Products In The Kencana Village Of Bogor City," *International Journal of Business, Economics, and Social Development*, vol. 4, no. 4, pp. 310–320, Nov. 2023, doi: 10.46336/ijbesd.v4i4.537.

[68] Leaflet, "Leaflet — an open-source JavaScript library for interactive maps," [Accessed: 2025-05-26]. [Online]. Available: https://leafletjs.com/

[69] SpareCores, "hpc7a.48xlarge by Amazon Web Services - Spare Cores," [Accessed: 2025-05-17]. [Online]. Available: https://sparecores.com/server/aws/hpc7a.48xlarge

[70] M. McInnes, "HBv4 size series - Azure Virtual Machines," 2025, [Accessed: 2025-05-20]. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/high-performance-compute/hbv4-series

[71] SpareCores, "h3-standard-88 by Google Cloud Platform - Spare Cores," [Accessed: 2025-05-20]. [Online]. Available: https://sparecores.com/server/gcp/h3-standard-88

[72] Oracle, "Compute Shapes," [Accessed: 2024-12-16]. [Online]. Available: https://docs.oracle.com/en-us/iaas/Content/Compute/References/computeshapes.htm

[73] FCT, "FICHA TÉCNICA DAS PLATAFORMAS COMPUTACIONAIS," 2024, [Accessed: 2024-12-27].

[74] M. Tyson, "Ampere's Altra Max 80 Core Arm CPU Gets Benchmarked, Delidded, Measured," Jul. 2022, [Accessed: 2025-05-20]. [Online]. Available: https://www.tomshardware.com/news/ampere-altra-max-80-ccore-arm-delidded

[75] Geekbench, "Amazon EC2 hpc6a.48xlarge - Geekbench," Jan. 2022, [Accessed: 2024-12-16]. [Online]. Available: https://browser.geekbench.com/v5/cpu/12156639

[76] SpareCores, "hpc6id.32xlarge by Amazon Web Services - Spare Cores," [Accessed: 2025-05-17]. [Online]. Available: https://sparecores.com/server/aws/hpc6id.32xlarge

[77] M. McInnes, "HBv3 size series - Azure Virtual Machines," Oct. 2024, [Accessed: 2024-12-16]. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/high-performance-compute/hbv3-series

[78] ——, "HC size series - Azure Virtual Machines," [Accessed: 2025-05-20]. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/high-performance-compute/hc-series

[79] SpareCores, "c2d-highcpu-32 by Google Cloud Platform - Spare Cores," [Accessed: 2025-05-17]. [Online]. Available: https://sparecores.com/server/gcp/c2d-highcpu-32