future internet

MDPI

Article

Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge

Hamish Sturley [1,†], Augustin Fournier [2,†], Andoni Salcedo-Navarro [3,†], Miguel Garcia-Pineda [3,†] and Jaume Segura-Garcia [3,*,†]

[1] School of Computing, Engineering and Physical-Science, University of the West of Scotland, Paisley Campus, Paisley PA1 2BE, UK; b00336796@studentmail.uws.ac.uk
[2] Télécom SudParis/Orange, 91000 Evry, France; augustin.fournier2701@gmail.com
[3] Computer Science Dpt–ETSE, Universitat de València, 46100 Burjassot, Spain; andoni.salcedo@uv.es (A.S.-N.); miguel.garcia-pineda@uv.es (M.G.-P.)
* Correspondence: jaume.segura@uv.es; Tel.: +34-963-543-730
† These authors contributed equally to this work.

Abstract: With the emergence of containerization 10 years ago, we saw a compact, convenient and portable way of running apps directly concurrently with virtualization. The major difference is in the architecture. Containers share the same kernel as the guest and then do not virtualize low-layer components like the Central Processing Unit (CPU). On the one hand, they are lighter and more flexible than virtual machines (VMs). On the other hand, VMs can more precisely meet the low-layer needs and are completely autonomous systems. Nowadays, what is the best architecture to use to develop an application? In this paper, we will study the two main virtual methods of deploying this. We will compare both methods on several criteria: compatibility based on user experience and the ease of installation/deployment, scalability based on the automatic elasticity facing the workload and energy efficiency in terms of energy and computer resources. After the tests, we conclude that the containerization option is the most ecologically advantageous option in terms of energy consumption.

Keywords: containerization stack; edge computing; orchestration; performance tests; virtualization stack

Citation: Sturley, H.; Fournier, A.; Salcedo-Navarro, A.; Garcia-Pineda, M.; Segura-Garcia, J. Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge. *Future Internet* **2024**, *16*, 427. https://doi.org/10.3390/fi16110427

Academic Editors: Dimitrios Dechouniotis and Ioannis Dimolitsas

Received: 1 October 2024
Revised: 12 November 2024
Accepted: 18 November 2024
Published: 19 November 2024

1. Introduction

As digital transformation accelerates across various sectors, the adoption of advanced technologies such as e-Government, Industry 4.0, Agriculture 5.0 and Smart Cities is becoming increasingly widespread. These technologies, powered by the development of artificial intelligence (AI) and other emerging tools, offer unprecedented opportunities for institutions, governments and organizations to enhance efficiency, reduce operational costs, and achieve higher levels of automation [1]. The shift towards these technologies necessitates a critical examination of the underlying infrastructure, particularly in how software applications are deployed and managed to meet the growing demands of modern systems.

In the context of the Computing Continuum, which spans from Cloud to Edge/Fog environments, the deployment of applications requires a careful selection of the appropriate hosting environment [2]. The decision between virtualization and containerization, two prevalent technologies in this space, plays a pivotal role in determining the performance, scalability and efficiency of these deployments [3,4]. Virtualization, a well-established technology, allows multiple operating systems to run concurrently on a single physical machine by abstracting the hardware layer. In contrast, containerization offers a more lightweight approach by isolating applications within containers that share the host operating system's kernel, leading to faster deployment times and more efficient resource utilization.

As organizations increasingly shift towards Edge Computing—where data processing occurs closer to the source rather than in centralized cloud servers—the choice between virtualization and containerization becomes even more critical. Edge environments often come with constrained resources and specific Quality of Service (QoS) requirements, making it imperative to choose the right deployment strategy [5]. Virtual machines (VMs) provide robust isolation and security, which are essential in many Edge applications. However, they can also introduce significant overhead, potentially impacting performance in resource-limited settings. On the other hand, containers, while more resource-efficient, may pose challenges related to security and orchestration in diverse and distributed Edge environments [6].

The complexities involved in deploying applications across the Computing Continuum highlight the need for a comprehensive comparative analysis of virtualization and containerization technologies [2]. This paper aims to address this need by evaluating various virtualization and containerization stacks to determine the optimal environment for hosting deployments and the applications they support. The analysis will consider key factors such as performance, scalability, security and ease of management, which are crucial for meeting the unique demands of Edge Computing.

In addition to the technical evaluation, this study will also explore the broader implications of these technologies for the future of application deployment in Edge environments. As the Edge becomes an increasingly vital part of the Computing Continuum, understanding the tradeoffs between virtualization and containerization will be essential for organizations seeking to leverage these technologies effectively.

Ultimately, the findings of this research will contribute to the ongoing discourse on the best practices for deploying applications in the Computing Continuum, with a particular focus on Edge Computing. As the digital landscape continues to evolve, the ability to make informed decisions about deployment strategies will be key to unlocking the full potential of emerging technologies and ensuring their successful integration into the fabric of modern infrastructure.

Another possible research motivation in the study of the comparison between virtualization and containerization concerns the architecture of the CPU within the context of Edge Computing. Depending on the use of Intel (x86_64) or ARM architectures [7], the set of instructions used can be different. When using reduced instruction sets (RISC) in ARM and low-cost devices for Edge Computing [8], the advantages in performance, regarding the tradeoff between energy consumption and computational power, can be achieved efficiently when the proper use case is selected.

This work is presented as follows: after this introductory section, the section on related works explains some of the studies that establish the state of the art for this issue. Section 3 is focused on the materials and methodology and explains the tools, software and hardware used for the application tests with different architecture combinations. Section 4 will explain the results of the tests with different architectures, which will be discussed and concluded in Section 5.

2. State of the Art

The recent shift to Edge Computing models has placed a greater emphasis on cheaper and more power-efficient systems that are not well addressed by x86/64 architectures. Instead, many Edge Computing solutions are turning to alternative architectures, such as ARM-based processors, which are known for their lower power consumption and cost-effectiveness. These architectures are better suited for distributed, resource-constrained environments where Edge devices operate with limited power and processing capabilities. As Edge Computing continues to grow, there is an increasing demand for customized, specialized hardware that can meet these performance and energy efficiency needs, driving innovation beyond traditional x86/64 systems. This shift is helping industries deploy intelligent applications at the Edge, closer to the source of data, with reduced latency and improved scalability.

Different authors have used differing approaches to the analysis of the different options offered by virtualization and containerization for the Computing Continuum, from the Cloud to User Equipment (UE), passing through the Edge. In [9], the authors provide a systematic review of container virtualization in real-time environments in Industry 4.0, analyzing 37 papers on real-time constraints, task latency and container platforms for cyber–physical systems. Also, in [10], the authors address the need for a systematic analysis of containerization, focusing on its role as a lightweight alternative to virtual machines for deploying modern applications in diverse environments. They taxonomically classify existing research on the performance comparison between hypervisors and containers and analyze container orchestration and performance monitoring tools. In [11], the authors conduct a systematic literature review to identify the most popular container technologies and trends in their research, facilitating more informed decision-making. In [4], the authors perform a systematic mapping of research on these core technologies, revealing that most articles focus on deployment through validation and solution research, while fewer papers address orchestration, particularly in philosophical and model-related contexts. The findings highlight gaps in the literature, offering valuable insights for researchers, industries and service providers.

Regarding Cloud-centered applications for these technologies, in [12], the authors compare the performance of three Cloud architectures: Infrastructure as a Service (IaaS) , Platform as a Service (PaaS) and Software as a Service (SaaS), focusing on the differences between containerization and virtualization. The findings indicate that while containers generally perform better in SaaS and PaaS environments, the performance metrics for IaaS are nearly the same for both methods. This research aims to help users make informed decisions about their Cloud deployment strategies by providing clear, quantifiable data on these differences. Also, in [3], the authors examine the performance differences between these technologies across IaaS, PaaS and SaaS Cloud architectures. The findings indicate that containers generally offer better performance for SaaS and PaaS, while the performance for IaaS is nearly identical between the two methods, providing valuable insights for users deciding between them.

In [13], the authors focus their interest on interactive training simulations, which are used across various industries and challenge traditional computing networks due to their high-performance demands. They examine how to extend traditional high-performance computing (HPC) environments into Cloud-based services to support multiple interactive simulations while maintaining performance. Their work evaluates four HPC load-balancing techniques—virtualization, software containers and clustering—to optimize the scheduling and execution of these simulations, finding that the choice of technique should be based on cluster resources, job competition and software requirements.

Regarding Internet of Things (IoT) applications requiring high computing power and real-time processing, in [6], the authors propose to face these challenges with low-cost Small Board Computers (SBCs) by leveraging container-based technologies and fog computing to enhance device collaboration, scalability and load balancing. They compare the container orchestration platforms Docker Swarm and Kubernetes on SBC clusters, finding that Docker Swarm provides superior performance across various topologies and cluster configurations.

In [2], the authors face the problem of container and microservice management platforms like Kubernetes, which are crucial for Cloud computing but struggle with heterogeneous computing environments, where nodes have varied characteristics. They propose to replace Kubernetes' native scheduler with a customizable one that better accommodates the diverse needs of a Continuum Computing environment. Testing a batch-based scheduling approach on virtual machines demonstrates improved performance over Kubernetes' pod-by-pod method, highlighting the limitations of default scheduling in supporting complex deployment requirements.

In [14], the author exposes in a seminar a comparison between hypervisor-based and container-based virtualization, highlighting the rise in popularity of container-based virtualization due to Docker, a tool that simplifies container management. While hypervisor-based

virtualization has been a dominant method for decades, container-based virtualization offers distinct advantages, leading to its increased use. The paper details the benefits and tradeoffs of both technologies, with a particular focus on container-based virtualization and Docker. It also addresses the security risks associated with containerization and provides a summary of related work in the field.

Different authors have tackled the integration of virtualization/containerization with Edge Computing. In [15], the authors investigate different works on Cloud and Edge Computing paradigms and their virtualization at the Edge. They also provide a taxonomy based on the state of the art for virtualized resources. In [16], the author analyzes the application of the concept of device virtualization in IoT/Edge Computing and carries out performance studies using single-board computers (SBCs) to efficiently deploy different low-power virtualized devices (i.e., families of Raspberry Pi and Odroid) in terms of performance when handling container-virtualized instances. Also, in [17], the authors focus their efforts in evaluating a methodology based on the study of containerization-based Edge–Cloud computing infrastructures' performance. They saw that containerization at the Edge does not introduce noticeable performance degradation in terms of communication, computing and intelligence capabilities, making it a promising technology for the Edge–Cloud computing paradigm, although they claim that there is still room for improvement, especially in time-critical industrial applications. According to them, this evaluation methodology can be used as a reference for practitioners in Edge–Cloud computing to obtain a client-perspective overview of system performance.

The massive amount of data generation today challenges current network architectures, which struggle with bandwidth, latency and privacy issues. From the Edge Computing perspective, these problems are addressed by enabling local data processing, while Network Function Virtualization (NFV) simplifies service creation by virtualizing traditional network functions. In [5], the authors compare containers and virtual machines through the implementation of a unified management and orchestration platform (MANO) for Virtualized Everything Functions (VxFs) using LXD for containers and KVM for virtual machines on a Raspberry Pi 4. The results indicate that containerization is not always suitable alone and is best used as a complementary approach to full virtualization.

3. Materials and Methods

In this section, we outline the tools and procedures used for the different experiments conducted to check the performance of the different approaches selected. First, we describe the software tools used, and later, we describe the hardware options. Finally, the tests deployed define a comparative framework between both approaches.

3.1. Stress-ng

To guarantee a consistent and dependable benchmark, it was essential to choose a tool that could apply a controlled and reproducible workload to the system. For this purpose, the `stress-ng` tool, short for Stress Next Generation, was selected. Stress-ng [18] is a powerful, Linux-based benchmarking and stress-testing tool specifically developed to evaluate the performance and stability of various hardware components under extreme conditions. It is widely used in scenarios where assessing the resilience and capability of computer systems is critical, such as in Fog (Fog Computing) and Edge Computing environments, where devices often operate under varying loads and conditions.

Stress-ng offers an extensive range of tests that can apply loads to different parts of a system, including the CPU, memory, input/output (I/O) subsystems, and more. By simulating intense workloads, Stress-ng helps to identify potential weaknesses or bottlenecks within these subsystems. This is particularly important in Fog and Edge Computing, where hardware is often deployed in remote or resource-constrained environments, requiring systems to be both robust and reliable.

The tool's metrics provide detailed insights into how a system performs under pressure. For instance, it can measure the CPU's ability to handle intensive computational

tasks, the memory's capacity to manage large datasets, and the I/O subsystem's efficiency in processing data transfers. These metrics are crucial for understanding overall performance and identifying any areas that may require optimization. Additionally, Stress-ng can simulate hardware failures or resource exhaustion scenarios, allowing engineers to test a system's response to such conditions and ensure that it can recover gracefully or maintain functionality. This tool provides multiple options when configuring the input and output settings for the planned architecture.

The `--metrics` option displays the total number of "Bogo" (bogus) operations performed in the stress processes, but these are not reliable indicators of performance or throughput and are not intended for benchmarking. Instead, they provide a useful way to observe system behavior under different types of load.

Stress-ng uses an unconventional metric called "Bogo Ops", which measures the number of iterations the stressor completes during a run, indicating the overall amount of "work" done in Bogo operations (Bogo Ops). Bogo Ops provide a general idea of system performance under load conditions but are not intended to serve as precise benchmarking metrics. The other metrics used within this `--metrics` option are real time, usr time, sys time, Bogo Ops/s (real time) and Bogo Ops/s (usr + sys time). These metrics are explained in Table 1.

Table 1. Other metrics explained within stress-ng.

Metric	Explanation
real time (s)	The average wall clock duration (in seconds) of the stressor is the total wall clock time for all instances of that stressor divided by the number of stressors run.
usr time (s)	The usr time is the total user time (in seconds) consumed by running all instances of the stressor.
sys time (s)	The sys time is the total system time (in seconds) consumed by running all instances of the stressor.
Bogo Ops/s (real time)	The total Bogo operations per second based on wall clock time, which reflects the apparent runtime, increases as more processors distribute the workload, reducing wall clock time and boosting the apparent Bogo Ops rate of the system.
Bogo Ops/s (usr + sys time)	The total Bogo operations per second based on cumulative user and system time represent the real Bogo Ops rate of the system, accounting for the actual execution time across all processors, which generally decreases with more concurrent stressors due to contention on cache, memory, execution units, buses and I/O devices.

The use of the package `stress-ng` is not advised for precise and conventional measurements; it is only recommended to measure tendencies. The metrics used within this tool have no transformative operation to any other real metric, such as throughput, delay or any other. We used it as a comparative tool for the different experiments planned.

3.2. Top System Monitor

The `top` system monitor [19] is a powerful and widely used tool in Linux-based operating systems, providing real-time insights into system performance and resource utilization. By default, top tracks a variety of key metrics, including per-core CPU usage, memory consumption, load averages and detailed information about all running tasks and processes. The interface is dynamic, updating continuously to reflect the current state of the system, which allows users to observe how resources are being allocated and identify potential bottlenecks or issues in real time. This makes top an invaluable tool for system administrators and developers who need to monitor system health, troubleshoot performance problems or manage resources effectively.

In our planned tests, we utilized top to capture detailed performance data, which were recorded as JSON objects for later analysis. By doing so, we can systematically track how the system behaves under the stress conditions we impose, including how resources are distributed across multiple CPU cores and how memory usage fluctuates. This approach

allows us to gather precise and structured data, which can be analyzed to understand the impact of different stressors on system performance. The JSON format also facilitates easy integration with other tools and workflows, enabling a more comprehensive assessment of the performance outcomes from our tests. This data-driven approach enhances the accuracy of our performance evaluations, providing a solid foundation for optimizing system configurations or identifying areas for improvement.

3.3. Power Meter Data Logger

For the performance evaluation, aside from the usual machine-based metrics, we measured the sustainability of the system in terms of power consumption. To this end, we selected a power meter logger, which was the UM24C [20] power meter data logger, due to its broad sensor suite. This logger can measure the supply voltage, supply current, supply power, cable resistance, charged capacity, temperature, USB data line voltage and USB charging mode. Table 2 shows the main operational ranges for this power meter. Logging is typically provided over the Bluetooth interface and managed through an app provided by the manufacturer; however, this was replaced with the `rdumtool` [21] command line tool, exposing sensor values as JSON objects in order to improve ease of processing.

Table 2. The main features of the um24c power monitor as stated by the manufacturer [20].

Sensor	Range	Resolution	Accuracy
Supply V	4.50–24.00 V	0.01 V	$\pm(0.2\% + 1digit)$
Supply A	0.000–3.000 A	0.001 A	$\pm(0.8\% + 3digits)$
Cable resistance	1.5–9999.9 Ω		
Temperature	−10 °C–100 °C		±3 °C

3.4. Raspberry Pi Testbed Setup

Due to the rise in ARM-based Edge Computing infrastructure, an ARM host was selected for testing to best determine results for all computing paradigms. The Raspberry Pi 4B+ was chosen as a testbed to compute resources due to the greater availability of high-accuracy low-voltage meters encouraging the selection of devices in the 5 V range. The host Raspberry Pi (rpi) was additionally subjected to tests to provide a baseline for performance, with an Operating System (OS), which was a 64-bit Rasbian 12 desktop.

The following containers and virtual machines were created in order to test the performance of each:

1. **Kubernetes and Docker:** The Kubernetes test environment was set up as a single-node Kubernetes cluster using the minikube Kubernetes stack, the Podman container runtime and a single 64-bit Debian 12 container for running the tests. The docker test environment was set up as a 64-bit `linuxserver debian 12` ssh container with stress-ng installed on the standard docker engine with Docker Compose orchestration.
2. **x86 and aarch64:** The x86 VM was run as a 64-bit QEMU virtual machine with `qemu-sysem-x86_64`, and Debian 12 was used for the tests.x86 was selected for emulation in the ARM environment to demonstrate the need for effective virtualization methods in foreign instruction sets due to the rise in ARM-based Edge Computing resources. The ARM VM was run as a 64-bit QEMU virtual machine with `qemu-system-aarch64` with the virtual machine type, the kernel virtual machine extension or KVM, allowing for the direct throughput of quest operations, which were used to best represent the capabilities of virtual machines. The arm64 version of Debian 12 was used for the tests (used as a Raspberry Pi VM).

3.5. x86 Testbed Setup

When building the test setup, it was determined that using a virtualized environment would reduce labor and complexity requirements when testing the system, ensure parity

between the systems being tested and enable the sharing of computing resources. To this end, the Proxmox virtual environment was selected for use as a host for the systems evaluated.

The Proxmox virtual environment (PVE) is an enterprise-grade layer 2 hypervisor based on the Debian kernel. The main strengths of the PVE include a highly active community with excellent community and developer support, a highly capable webUI environment and support for advanced virtual storage.

Within the PVE, all evaluated systems were provided with the same hardware: 2 cores from both CPUs for a total of 4 threads, 8 Gigabytes of RAM, 100 Gigabytes of hard drive disk space and access to both the host's network and an internal virtual network, both provided through virtualized switches.

Figure 1 depicts a global scheme for the stress tests carried out in this work. These tests will also serve as a basis for the pointcloud generation use case that will be explained later.

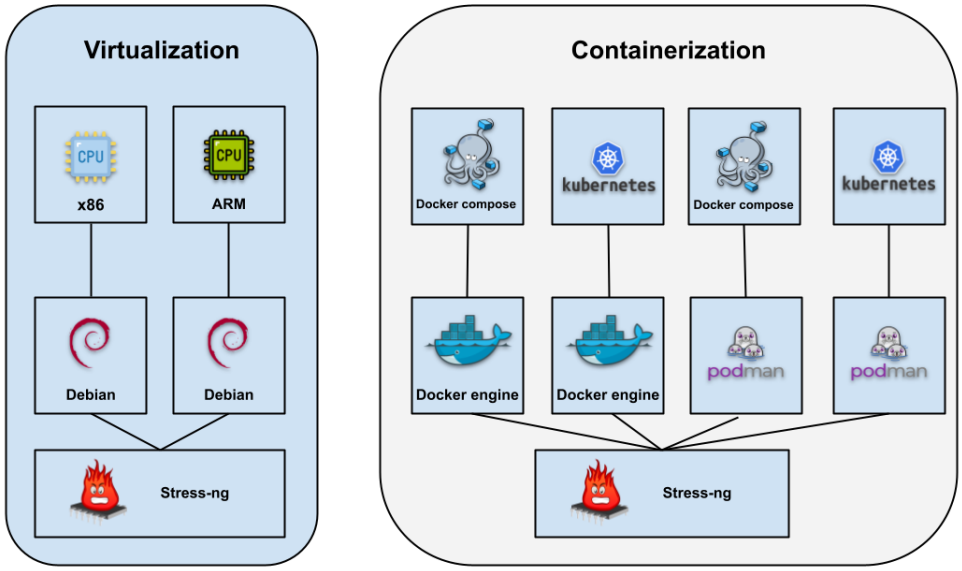

Figure 1. Global scheme of the tests.

3.6. x86 Tests

As previously mentioned, the term Bogo Ops is short for "bogus operations", and it is a metric used by stress-ng to indicate the number of operations performed by the stress test within a given time frame, which for us is 1 min and 10 min.

While Bogo Ops are not a standardized measurement like FLOPS (for floating-point operations) or IOPS (for disk), they give a rough idea of performance by showing how many iterations of a given task have been completed. The higher the Bogo Ops value, the more operations the system was able to perform during the stress test, which suggests better performance for that particular component.

When running CPU stress tests with `stress-ng`, the tool runs different types of computational workloads (like integer calculations, floating-point math, or cryptographic operations). The Bogo Ops metric in CPU tests indicates how many iterations of these workloads the CPU was able to complete.

In memory stress tests, stress-ng performs memory-intensive operations like allocating, copying, reading and writing data to and from memory. Here, the Bogo Ops metric reflects how many memory-related operations (e.g., memory reads, writes or allocations) were completed. This metric is useful for testing how well the system handles memory-intensive workloads and can reveal memory bottlenecks.

For I/O disk tests, stress-ng performs read/write operations on files to measure the disk's I/O performance. The Bogo Ops metric in I/O tests represents the number of I/O operations completed, such as file reads, writes and seeks or other disk-related operations. A large number of Bogo Ops in this context suggests that the disk subsystem can handle a

larger volume of I/O operations in the given time. We performed each test three times and obtained the average value in order to have relevant values.

Kubernetes and Docker Engine are very common in companies with complicated and scalable networks. These tools enable automated scalability management, load balancing and container failure management for a complex application. The goal is to test the different cases in a simple architecture with similar solutions to quantify performance. Compared to Docker Engine, Podman does not use a centralized daemon, which allows for lighter container management, does not use root connection for security, and natively supports pod management. Podman is therefore a legitimate competitor to Docker Engine as a runtime container. For orchestrators, Kubernetes is designed to handle complex applications on a large scale and is suitable for production environments, so it has plenty of features. Docker Compose is designed for a simple application consisting of a single node. It does not have features such as scaling (too light), fault tolerance and advanced volume management, making it much easier to configure. Later on, we will describe Docker Compose as a container manager, but as having a simple command utility for the sake of simplicity.

For each container–runtime and container–manager coupling, we obtained specific metrics:

- Duration of the test: this was measured to see the influence of time on efficiency.
- Bogo Ops: This is a non-official metric used to measure the number of operations carried out by the hardware. We will use this to compare the efficiency of each method.
- User time: this is the time taken by the CPU to execute code from you or your apps.
- System time: this is the time taken by the CPU to manage and interact with other hardware components (I/O, network, memory, etc.).

3.7. Raspberry Pi Tests

In order to determine the power efficiency of the evaluated automated power meter in the different virtual environments, observations were generated via a bash script [22]. Tests were run 5 times for each virtual environment, and then Bogo Ops in a 60 s window per watt second were used to determine a value for relative power efficiency.

Due to issues with the rdumtool and maximum recording length, the measured watt hours were compared to a correspondingly adjusted Bogo Ops value.

3.8. Use Case: Dynamic Pointcloud Generation with OpenDroneMap

After testing the different architectures, we used the best option for a specific use case: pointcloud processing. Pointcloud processing and management have specific requirements for computing. Similarly to [23], we found that a good software option for processing is Open Drone Map (ODM) [24]. This is an open framework used to generate and process maps and pointclouds from Unmanned Aerial Vehicles (UAVs) [25].

In this use case, we will demonstrate how Docker and Kubernetes dynamically manage and scale pointcloud generation. Drone images are transmitted to a Raspberry Pi for preprocessing, which then requests a GPU-accelerated transformation matrix from an Edge server to align spectral bands. Kubernetes orchestrates Docker containers in the Cloud, distributing the computational load for efficient pointcloud generation. The system architecture is shown in Figure 2.

The workflow starts with the drone capturing images and transmitting them to a Raspberry Pi, which preprocesses the images for pointcloud generation. Upon receiving the first image, the Raspberry Pi requests a transformation matrix from the Edge server to align the images from different spectral bands. The Edge server, using GPU acceleration, computes the matrix, ensuring all images share the same perspective and reducing computation time.

After obtaining the transformation matrix, the Raspberry Pi extracts metadata from the images to apply vignetting corrections. It then aligns the images using the transformation matrix, ensuring consistency across all spectral bands. Once the images are preprocessed and aligned, the Raspberry Pi sends them to the Cloud environment for further processing. In the Cloud, Kubernetes orchestrates the deployment of ODM instances within Docker

containers. Each instance is responsible for processing a specific spectral index, such as the NDVI, or band.

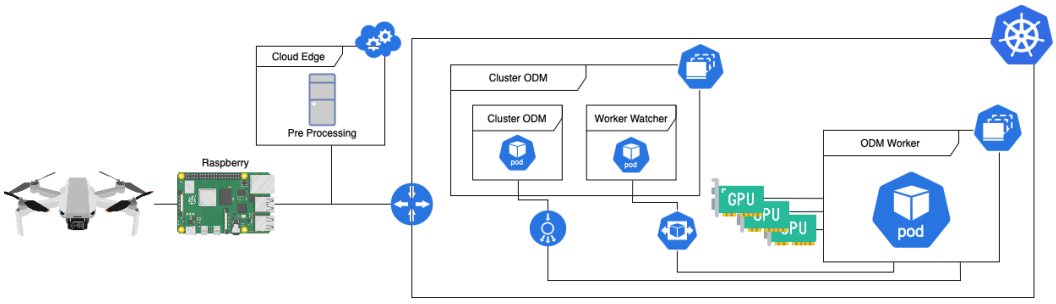

Figure 2. System architecture for pointcloud generation using distributed computing with Raspberry Pi, Edge server and Kubernetes-orchestrated Cloud environment.

The system scales dynamically by adjusting the number of ODM worker instances based on the number of spectral indices to be computed in parallel. This scalability is achieved through Kubernetes' ability to manage resources and deploy additional instances as needed. A Watcher Service operates within the Cloud environment to monitor the availability of ODM instances. It ensures that new instances are registered with ClusterODM, which functions as a load balancer, distributing tasks among the available workers.

By distributing computational tasks across multiple ODM workers, the system significantly accelerates the pointcloud generation process. The parallel processing of different spectral indices reduces the time required to obtain results. This architecture demonstrates the effectiveness of combining Kubernetes and Docker to manage complex workloads in a Cloud environment.

4. Results

4.1. Tests Results

For the tests explained in Sections 3.6 and 3.7, the average values obtained are depicted in Table 3. In this figure, we see that the payload (in terms of "Bogo Ops") for longer processes (10 min) is greater (and proportional to that of shorter processes (1 min)). The speed or apparent runtime (in terms of "Bogo Ops/s" in real time) seems to be nearly constant in both processes (the longer and shorter ones), and the runtime for the cumulative user and system time has the same behavior but on a lower scale because these do not take into account as many operations.

Table 3. Averages for tests carried out with CPU using `stress-ng` in Section 3.6.

Duration (min)	Manager	Container Runtime	Bogo Ops	Bogo Ops/s	Usr + Sys Bogo Ops/s
10	Docker Compose	Podman	311,7331.0	5195.5	1300.4
10	Docker Compose	Docker Engine	3,089,738.0	5149.6	1288.6
10	-	Docker Engine	3,093,922.3	5156.5	1290.2
10	-	Podman	3,050,764.7	5084.6	1272.4
10	Kubernetes	Podman	3,027,897.3	5046.5	1279.8
10	Kubernetes	Docker Engine	2,984,890.0	4974.8	1265.3
1	Docker Compose	Docker Engine	308,970.3	5149.4	1288.8
1	Docker Compose	Podman	309,507.7	5158.4	1293.0
1	-	Docker Engine	310,324.0	5171.9	1294.4
1	-	Podman	306,818.3	5113.5	1281.2
1	Kubernetes	Podman	299,595.3	4993.2	1267.0
1	Kubernetes	Docker Engine	296,143.0	4935.6	1254.3

For the tests described in Section 3.7, the following data are presented. Of note are the highly similar rpi, Docker and aarch64 vm measurements, demonstrating the high efficiency of containers, as well as the suitability of virtualization stacks with the appropriate extensions. On the other hand, the apparent lack of performance in Kubernetes hosted in Podman environments seems to indicate inefficiencies in the container runtime or high overheads. Finally, the extremely poor performance of the x86 virtual machine shows the difficulties in emulating legacy hardware on modern instruction set paradigms. Another interesting finding, however, is uncovered when evaluating the stress reports from the x86 host. It was discovered that completed add operations were found to be roughly equal to divide operations, as without full CPU parity, no virtual acceleration was being undertaken at all, indicating a possible area of improvement for virtualization speed when similar instructions are available.

Figures 3–7 show the percentage of additional performance based on the solution that had the lowest percentage. We consider the numbers of `usr + sys Bogo Ops/s` and `Bogo Ops/s` for the task itself to have the most reliable graph possible. The purpose of these results is to identify trends, not to consider specific measures as they are not standardized. They allow us to get an idea of performance and resource management. As expected, the Kubernetes container manager solution is less effective in our scenario because we will deploy only a few containers over a short period of time. There is a difference in CPU usage performance of 5%, which can be significant in terms of time and resources. It is difficult to come to a conclusion about runtime containers here. Although the use of Docker Compose should not affect the performance results of the containers, there is a difference in the results obtained between using Docker Compose and using the runtime container alone. This difference may be due to inaccurate measurements and other factors. The measurement and explanation of these differences may be addressed in a future study.

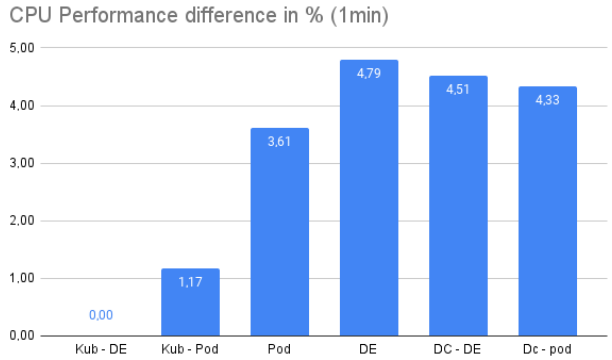

Figure 3. CPU performance difference in percent for 1 min.

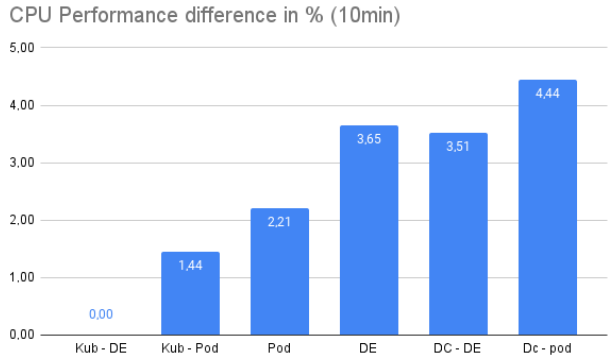

Figure 4. CPU performance difference in percent for 10 min.

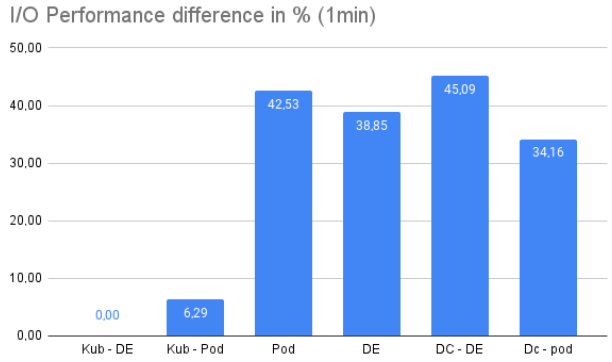

Figure 5. I/O disk performance difference in percent for 1 min.

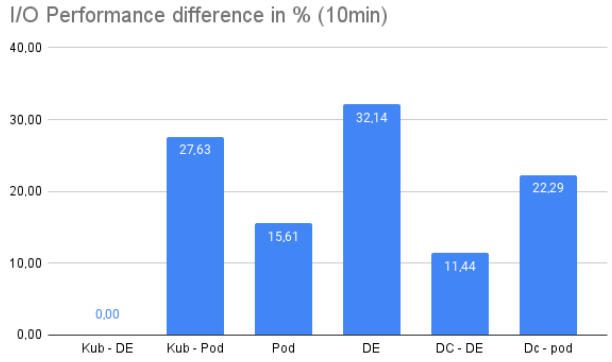

Figure 6. I/O disk performance difference in percent for 10 min.

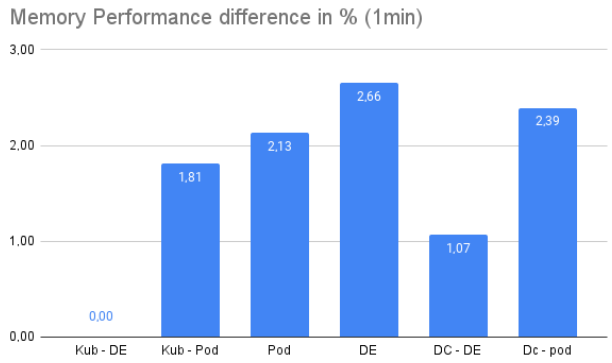

Figure 7. Memory performance difference in percent for 1 min.

For the CPU performance graph over 10 min (see Figure 4), a similar trend is obtained. It should be noted that the performance difference between Kubernetes and other solutions decreased, showing a trend that suggests that Kubernetes can perform well in the long term. With these graphs, we can see that the use of Docker Compose with Podman performs better on all levels in the context of using only a few containers over a moderate period of time. For the I/O disk performance graphs over 1 min (see Figure 5), we can see that Docker Compose acts as a better container manager in our case, being 40% more efficient on average. This can be explained because Podman does not use a daemon for managing containers. Thus, there is no overload with this and with complex network services. In addition, Kubernetes is very complex in terms of components; it needs a Kuberlet, an API server, a controller manager, etc. For this reason, Kubernetes and Docker Engine are less likely to have better results than Docker Compose and Podman.

The results for the I/O disk performance graph for ten minutes (see Figure 6) show the same trend as for CPU performance; Kubernetes' results are still worse than the others,

but they significantly increased. This trend supports the hypothesis about the long-term efficiency of Kubernetes.

For the memory part (see Figure 7), the difference is less significant than CPU consumption or I/O operations. This difference can be explained by the same argument used for the I/O disk performance: the complexity of Kubernetes is more resource-consuming than the lightweight container manager Docker Compose.

We can then recommend using Docker Compose and Podman in an environment that requires resource efficiency on a one-time basis and not for a long-term service. For example, to train artificial intelligence (AI) on a model with large amounts of data, we can use Docker Compose and Podman. In the case of a complex architecture with a need for redundancy and error management or a service/application, we recommend Kubernetes, which will meet these needs adequately.

Figure 8 summarizes the whole range of tests performed in the different virtualized and containerized architectures. This figure shows that the use of Kubernetes with the Podman backend demonstrates the advantage of utilizing Kubernetes with the more popular Docker Engine; furthermore, the true extent of x86's virtualization inefficiency is revealed.

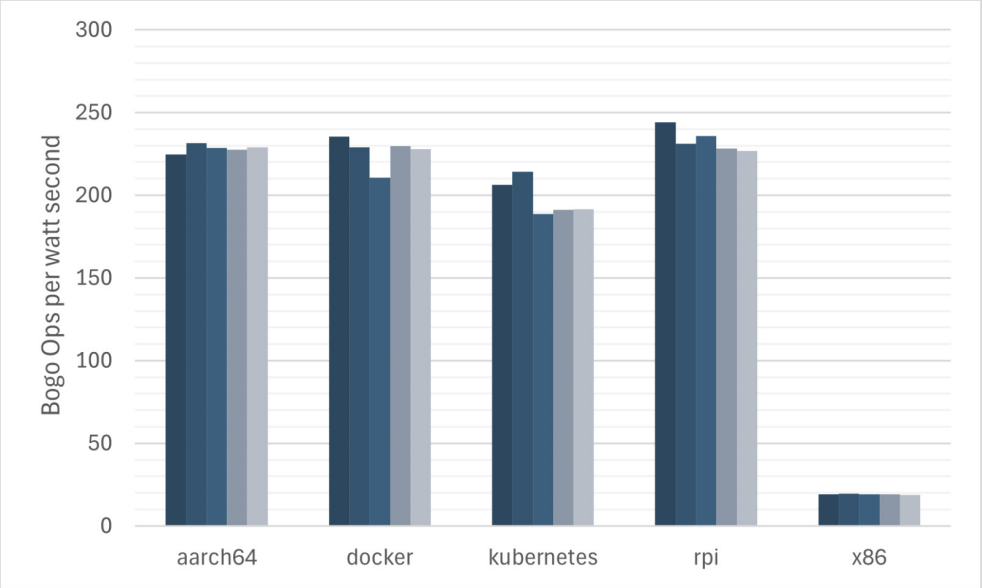

Figure 8. Comparison of all 25 tests (5 min) performed on the Raspberry Pi testbed.

Figure 9 summarizes the same set of tests as before but on a 20 min timescale instead of 5 min. This figure shows that the power efficiency of Kubernetes exceeds that of Docker when run for longer periods.

The widespread adoption of Kubernetes has fostered an extensive ecosystem of Kubernetes-compatible containers and tools, along with a supportive community of developers and users. This environment provides organizations with a powerful suite of tools, which encourages both new and existing users to engage with Kubernetes-based solutions. As a result, individuals and organizations often prefer Kubernetes-compatible solutions, even when theoretically superior alternatives may be available.

Although Kubernetes was originally designed for clustered deployments, single-node Kubernetes cluster solutions have gained substantial popularity due to their reduced resource requirements and improved performance. For instance, Canonical's MicroK8s operates as a single-node cluster by default, catering to small production environments and development needs. Similarly, Rancher Labs' K3s is optimized as a single-node solution specifically for Edge Computing and IoT applications. This trend highlights the flexibility and adaptability of the Kubernetes ecosystem in addressing modern deployment requirements.

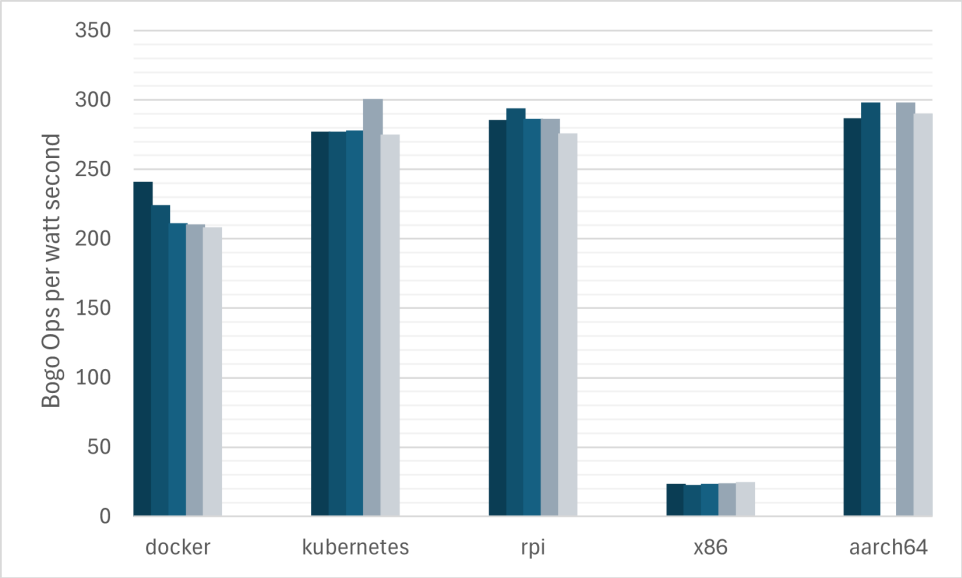

Figure 9. Comparison of all 24 tests (20 min) performed on the Raspberry Pi testbed.

The comparison between the Kubernetes experiment using the previous version and the new version (i.e., k3s) on a single node shows a difference of only 1.4%. Therefore, we consider the results of the new experiments to be compatible with those of the previous version. Also, as shown in Figure 10, the efficiency enhancement between the one-node and two-node options is around 8%.

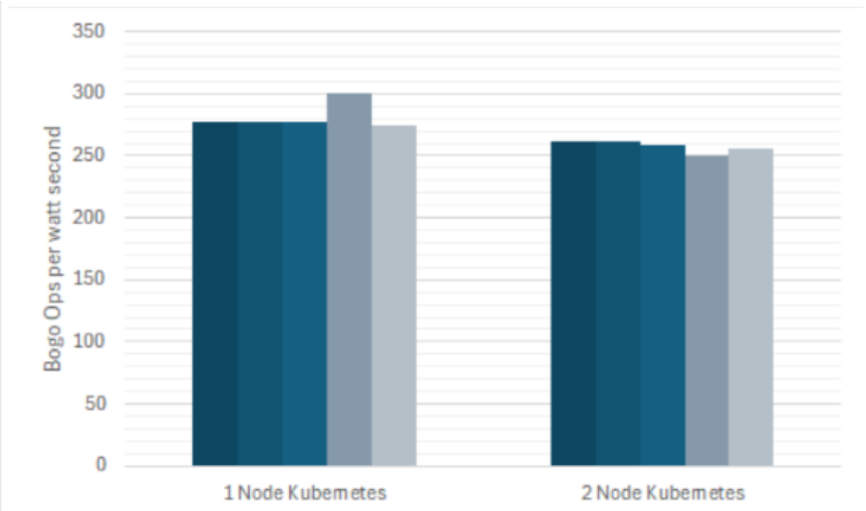

Figure 10. Tests with 2-node cluster compared to previous 1-node architecture.

4.2. Use Case Results

From the previous analysis, we found that the worst option was the use of the Kubernetes and Docker Engine stack, which in the different comparison test figures, with results given as percentages (Figures 3–7), is the reference baseline, as it underwent no improvement in performance. Our purpose here is to check this option for a use case with intensive processing.

To evaluate the performance and scalability of the use case given in Section 3.8, we conducted an experiment using a drone flight that captured 48 images. The data were processed using four ODM nodes configured in a cluster, with each node equipped with four virtual CPUs (vCPUs). The hardware specifications for the ODM nodes were based on

the recommendations provided in the official documentation (https://docs.opendronemap.org/installation/ (Visited on 29 September 2024)).

The results of the experiment are summarized in Table 4. The table presents the average preprocessing time per image, the average time taken to create each transformation matrix, and the total time required to generate the pointclouds for the RGB, Red Edge (RE), Red (R) and Green (G) spectral bands.

Table 4. Performance metrics for use case given in Section 3.8.

Metric	Average Time (s)	RGB (s)	RE (s)	R (s)	G (s)
Preprocessing Time	3.67	–	–	–	–
Warp Matrix Creation	24.72	–	–	–	–
Pointcloud Time	–	912.28	868.09	875.95	858.71

The experimental results, summarized in Table 4, highlight the effectiveness of our distributed computing approach for pointcloud generation. The average preprocessing time per image was 3.67 s, and the creation of the warp matrix took an average of 24.72 s. We processed the pointclouds for the four spectral bands in parallel. The processing times were 912.28 s for RGB, 868.09 s for RE, 875.95 s for R and 858.71 s for G. This parallelization significantly accelerated the overall processing workflow, demonstrating that distributing computational tasks across multiple nodes reduces processing time and enhances efficiency in generating pointclouds from drone images.

5. Discussion and Conclusions

5.1. Power Inefficiencies

The data presented in Table 3 indicate a clear need to optimize both container and virtualization environments on ARM Edge devices, as the range of watt-to-operation requirements will place a greater demand on both power requirements, potentially affecting battery life, and heat output.

Moreover, this also demonstrates the similarities between containerization and virtualization environments when considering power consumption, allowing greater freedom for software deployment.

5.2. Comparison Between Virtual Machines and Containers

Both methods of virtualization present some advantages. On the one hand, VMs can provide a graphical user interface that is very useful for monitoring and testing without dealing with the network. Furthermore, VMs emulate low-layer components, allowing you to tune them to fit your needs. Thus, you can make VMs for use with real stuff like USB sticks, which is an impossible thing to do with containers.

On the other hand, you have containers. These involve high-layer virtualization, with only the packages you need and your code embedded. Therefore, containers are light and can be modified quickly due to the share of the host's OS. Containers can easily be made lightweight and compatible. The community has a lot of container images with software already built in. Developers can just pull the image and get rid of the prerequisites, accelerating software development and app deployment. Containers are developed in a static context. The files and apps within a container are not meant to change. Developing and testing an app or some software meant to be tested in a dynamical context in a container may cause compatibility or security issues.

So, regarding the use of virtual machines or containers, although the solution is not straightforward, we can say that it depends on the requirements, but we can conclude that a good method of virtualization is crucial for the success of a project. To separate apps on the same host without overloading the host with a virtual OS, containers are a must. Containers should be used when flexibility and resource efficiency are critical, especially in resource-constrained environments such as Edge Computing.

As much as containers share the OS of the host, some cases are not compatible with granularity in hardware devices, complete isolation, specific OS configurations and customer requirements. Thus, the use of VMs would be relevant in this case.

As a general conclusion, we can see that even in the worst case of the different stack tests (using the Kubernetes and Docker Engine stack, which is a common option for companies), the performance can be meaningful, although the computing power is quite low.

For future work, we will further study different use cases and the analysis of performance in bigger hardware architectures for Edge or Cloud. Also, different tests of these technologies on security issues in Edge environments will be performed in a later work for deploying services in distributed and resource-limited environments.

Furthermore, we will explore the performance impact arising from the differences in system requirements between virtualization and containerization when executing a greater variety of different parallel tasks at the Edge.

Author Contributions: Conceptualization, J.S.-G. and M.G.-P.; methodology, J.S.-G., H.S. and A.F.; software, H.S., A.F. and A.S.-N.; investigation, H.S., A.F. and A.S.-N.; resources, J.S.-G. and M.G.-P.; writing—original draft preparation, J.S.-G.; writing—review and editing, H.S., A.F., A.S.-N. and M.G.-P.; funding acquisition, J.S.-G. and M.G.-P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Spanish Ministry of Science and Innovation/Spanish Research Agency (MCIN/AEI) within the project Agriculture 6.0 with reference TED2021-131040B-C33 and the project ECO4RUPA with reference PID2021-126823OB-I00, funded by MCIN/AEI/10.13039/501100011033 and by the European Union "NextGenerationEU"/PRTR. Also, the Spanish Ministry of Universities for the grant number PRX22/000503. Also, we thank the Generalitat Valenciana for the grant CIBEST/2023/101 and the grant CIAEST/2022/91.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank the Spanish Ministry of Science and Innovation/Spanish Research Agency (MCIN/AEI) and the Generalitat Valenciana for the funding provided for the development of this research. Also, we thank Juan Gutierrez and Santiago Felici for their advice. Further thanks go to the academic and catering staff of the Escola Tècnica Superior d'Enginyeria for their support during the initial phases of the project.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Javaid, M.; Haleem, A.; Singh, R.P.; Suman, R. Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study. *J. Ind. Integr. Manag.* **2022**, *07*, 83–111. [CrossRef]
2. Robles-Enciso, A.; Skarmeta, A.F. Adapting Containerized Workloads for the Continuum Computing. *IEEE Access* **2024**, *12*, 104102–104114. [CrossRef]
3. M, A.; Dinkar, A.; Mouli, S.C.; B, S.; Deshpande, A.A. Comparison of Containerization and Virtualization in Cloud Architectures. In Proceedings of the 2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 9–11 July 2021; pp. 1–5. [CrossRef]
4. Odun-Ayo, I.; Geteloma, V.; Eweoya, I.; Ahuja, R. Virtualization, Containerization, Composition, and Orchestration of Cloud Computing Services. In Proceedings of the Computational Science and Its Applications—ICCSA 2019, Saint Petersburg, Russia, 1–4 July 2019; Misra, S., Gervasi, O., Murgante, B., Stankova, E., Korkhov, V., Torre, C., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Tarantino, E., Eds.; Springer: Cham, Switzerland, 2019; pp. 403–417.
5. da Cunha, H.G.V.O.; Moreira, R.; de Oliveira Silva, F. A Comparative Study Between Containerization and Full-Virtualization of Virtualized Everything Functions in Edge Computing. In *Proceedings of the Advanced Information Networking and Applications*; Barolli, L., Woungang, I., Enokido, T., Eds.; Springer: Cham, Switzerland, 2021; pp. 771–782.
6. Fayos-Jordan, R.; Felici-Castell, S.; Segura-Garcia, J.; Lopez-Ballester, J.; Cobos, M. Performance comparison of container orchestration platforms with low cost devices in the fog, assisting Internet of Things applications. *J. Netw. Comput. Appl.* **2020**, *169*, 102788. [CrossRef]
7. Ouro, P.; Lopez-Novoa, U.; Guest, M.F. On the performance of a highly-scalable Computational Fluid Dynamics code on AMD, ARM and Intel processor-based HPC systems. *Comput. Phys. Commun.* **2021**, *269*, 108105. [CrossRef]
8. Tso, F.P.; White, D.R.; Jouet, S.; Singer, J.; Pezaros, D.P. The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures. In Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, Philadelphia, PA, USA, 8–11 July 2013; pp. 108–112. [CrossRef]

9. Queiroz, R.; Cruz, T.; Mendes, J.; Sousa, P.; Simões, P. Container-based Virtualization for Real-time Industrial Systems—A Systematic Review. *ACM Comput. Surv.* **2023**, *56*, 59. [CrossRef]
10. Bhardwaj, A.; Krishna, C.R. Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey. *Arab. J. Sci. Eng.* **2021**, *46*, 8585–8601. [CrossRef]
11. Silva, V.G.d.; Kirikova, M.; Alksnis, G. Containers for Virtualization: An Overview. *Appl. Comput. Syst.* **2018**, *23*, 21–27. [CrossRef]
12. Abuabdo, A.; Al-Sharif, Z.A. Virtualization vs. Containerization: Towards a Multithreaded Performance Evaluation Approach. In Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 3–7 November 2019; pp. 1–6. [CrossRef]
13. Mondesire, S.C.; Angelopoulou, A.; Sirigampola, S.; Goldiez, B. Combining virtualization and containerization to support interactive games and simulations on the cloud. *Simul. Model. Pract. Theory* **2019**, *93*, 233–244. [CrossRef]
14. Eder, M. Hypervisor- vs. Container-based Virtualization. Seminar Future Internet WS2015/16—Network Architectures and Services. July 2016. Available online: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2016-07-1/NET-2016-07-1_01.pdf (accessed on 28 August 2024).
15. Mansouri, Y.; Babar, M.A. A review of edge computing: Features and resource virtualization. *J. Parallel Distrib. Comput.* **2021**, *150*, 155–183. [CrossRef]
16. Morabito, R. Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation. *IEEE Access* **2017**, *5*, 8835–8850. [CrossRef]
17. Liu, Y.; Lan, D.; Pang, Z.; Karlsson, M.; Gong, S. Performance Evaluation of Containerization in Edge-Cloud Computing Stacks for Industrial Applications: A Client Perspective. *IEEE Open J. Ind. Electron. Soc.* **2021**, *2*, 153–168. [CrossRef]
18. King, C.I. Stress-ng. 2024. Available online: https://github.com/ColinIanKing/stress-ng (accessed on 24 August 2024).
19. Murray, C. Linux Handbook. Available online: https://linuxhandbook.com/top-command/ (accessed on 28 August 2024).
20. Ruiden, K. UM42C USB Power Meter Manual. Available online: https://fccid.io/2A5Y7-UM34C/User-Manual/User-manual-5807006.pdf (accessed on 28 August 2024).
21. Finnie, R. rdumtool—RDTech UM24C/UM25C/UM34C Bluetooth Interface Tool. Available online: https://github.com/smandon/rdumtool (accessed on 28 August 2024).
22. Undack. Containerpaper Repository. Available online: https://github.com/Undack/Containerpaper (accessed on 27 September 2024).
23. Catala-Roman, P.; Navarro, E.A.; Segura-Garcia, J.; Garcia-Pineda, M. Harnessing Digital Twins for Agriculture 5.0: A Comparative Analysis of 3D Point Cloud Tools. *Appl. Sci.* **2024**, *14*, 1709. [CrossRef]
24. OpenDroneMap. 2024. Available online: https://www.opendronemap.org/ (accessed on 20 August 2024).
25. Authors, O. ODM—A Command Line Toolkit to Generate Maps, Point Clouds, 3D Models and DEMs from Drone, Balloon or Kite Images. OpenDroneMap/ODM GitHub Page. 2020. Available online: https://github.com/OpenDroneMap/ODM (accessed on 20 August 2024).