Analysis of Virtualization Performance on Resource Efficiency using Containers and Unikernel

2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD) | 979-8-3503-6845-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICOABCD63526.2024.10704518

Aidil Arifin Nizar
Department of Informatics
Telkom University
Bandung, Indonesia
bhedil@student.telkomuniversity.ac.id

Siti Amatullah Karimah
Department of Informatics
Telkom University
Bandung, Indonesia
karimahsiti@telkomuniversity.ac.id

Erwid Musthofa Jadied
Department of Informatics
Telkom University
Bandung, Indonesia
jadied@telkomuniversity.ac.id

***Abstract*—Applications on a computer require resources and robust security measures to prevent attacks. In environments where multiple applications run simultaneously, isolation becomes essential, and the computer must efficiently manage resources for these isolated applications. Virtualization technology addresses this need by providing virtual resources, enabling the operation of virtual computers without requiring additional physical hardware. This approach enhances flexibility, efficiency, and secure isolation in managing computing resources. This research investigates the performance differences between container-based and unikernel-based approaches to virtualization, specifically on the Google Cloud Platform (GCP). The study is conducted in two stages: general tests and special tests. General tests involve measuring the image size and boot time, both critical factors in optimizing system performance and resource allocation. Special tests assess the system's performance under load, focusing on CPU and memory utilization by generating simulated requests to an Nginx web server using the wrk benchmarking tool, which simulates multiple concurrent clients to create a realistic workload. The results reveal significant advantages of the unikernel approach. The Nginx image size in a unikernel environment is about 1 MB, making it 57 times smaller than in a containerized environment. Additionally, unikernel boot time is 169 times faster. Furthermore, the unikernel demonstrates 22% more efficient CPU utilization and 0.6% better memory efficiency compared to containers, highlighting its potential for improved performance in specific use cases.**

Keywords—cloud, container, unikernel, performance, virtualization

I. INTRODUCTION

Rapid technological growth has led individuals to recognize the necessity of understanding and mastering technology. In the digital age, the internet has become the primary repository of information. It is therefore inevitable that the internet will become an integral part of every individual's life. Anyone who wishes to explore the digital realm can use a browser as a platform. The browser acts as a client, facilitating the retrieval and presentation of information from the internet. However, the browser obtains this information from a technology known as a web server.

Web-based applications are dependent on the web server for the retrieval and display of information to the client, namely the browser. The advent of 5G telecommunications technology has the potential to transform the traditional infrastructure of mobile communication networks, including 4G, 3G, and 2G. These networks were designed with a primary focus on high data transmission speeds. In contrast, 5G places greater emphasis on factors such as low latency, energy efficiency, and heterogeneity [1]. The processing of such large amounts of data must be conducted in real-time and requires a significant investment of resources [2].

The application can be utilized in its general form with the assistance of a computer that remains operational to transmit data responses to the client. It should be noted that the process itself requires a certain level of computing power and hardware specifications. The advent of cloud computing has removed the need for application developers to concern themselves with the operational costs of hardware specifications. However, developers are afforded the flexibility to select hardware configurations that are compatible with the computing requirements of applications developed for widespread usage. In the context of cloud computing, the concept of "pay-as-you-go" enables developers to consider the cost of computing resources in a more flexible manner [3].

Moreover, this study highlights the potential implications of virtualization techniques using container and unikernel. Virtualization is the process of creating virtual computers in several logical computer forms, with each logical computer having the capacity to operate with a different operating system [4]. The technique employs the resources of a single physical machine, yet it is capable of creating a distinct and isolated environment by partitioning the resources of the physical machine to facilitate the execution of independent applications or operating systems. This approach offers enhanced efficiency in the utilization of computing resources, an isolated environment for processing, and greater flexibility in the management of operating systems and applications.

In computing, a "container" represents a technology that wraps an application in one package, or alternatively, can be referred to as "containerd," which is filled with application programs, dependencies, and so forth [5]. Unikernel is a lightweight machine image that aims to reduce the memory footprint and image size by integrating the application code and its dependencies into single bootable binary images [6]. In this research, the containerization technology is utilized due to the popularity of Docker and the potential for Unikernel to offer enhanced computing performance and resource efficiency, which is comparable to that of container [7].

The issue addressed in this research is the assessment of the effectiveness of virtualization technology in the context of container and unikernel systems. The limitations of this research pertain to the limited resources available in cloud servers utilizing shared-core CPUs, which are employed to simulate traffic services on a small scale. The case study employed in this research serves to compare the performance of container and unikernel with that of general tests and special tests. General tests employ parameters such as the size of the image utilized and the speed of boot time. Special tests utilize a load testing approach, whereby numerous clients are concurrently requested to measure CPU utilization and memory usage. The problem that this research aims to address is determining which virtualization technology is more

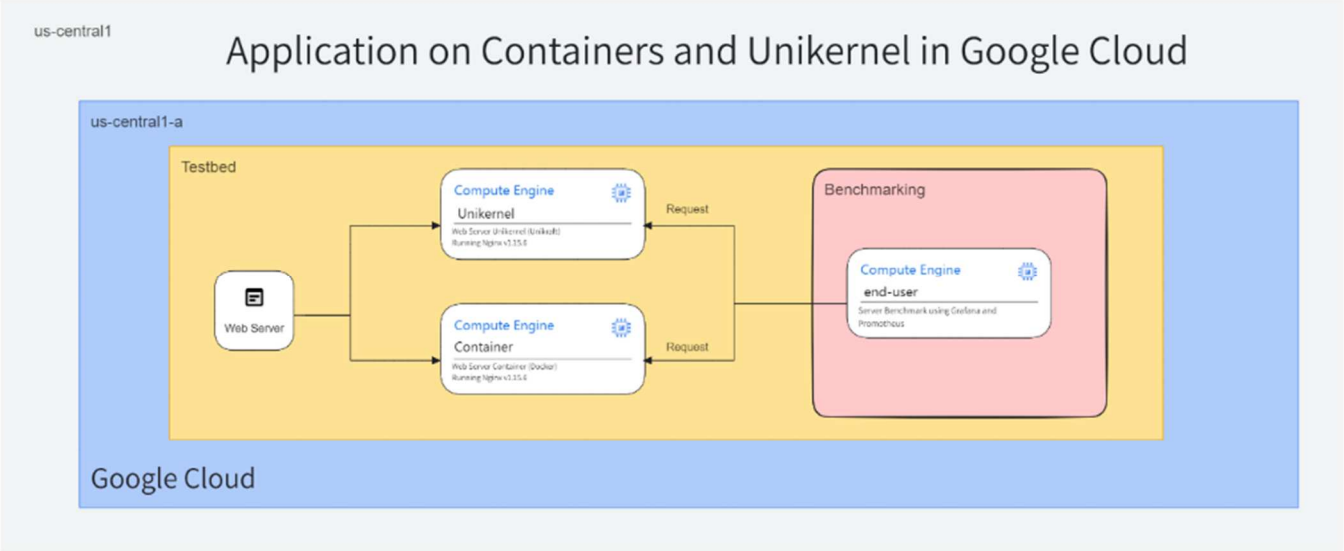

Fig. 1. Web Server Application Architecture using Compute Engine.

effective and optimal for use based on performance testing. This research will also explain how to measure the performance of the two virtualization technologies and why the test uses two separate parameters, namely general tests and special tests.

The aim of this research is to analyze the advantages and disadvantages of different container and unikernel approaches to managing computing resources. This will enable the identification of the most optimal virtualization technology, based on the results of performance testing with defined parameters. These include general tests (image size and boot time) and special tests (requests per second to the web server based on the number of clients using simultaneously or concurrency to measure memory and CPU utilization on the cloud-based server used). The research will focus on virtualization in containers and unikernel.

II. Related Studies

In 2017, Plauth, Feinbube, and Polze stated that, in terms of application throughput, most unikernels perform at least on par or even better than containers. They also noted that containers are not immune to overhead associated with network performance, which may make unikernels a more suitable choice in cases where throughput is not a primary concern, such as when latency, protocol overhead, or processing that occurs during data transfer are not significant factors. These are just some of the aspects that demonstrate that while containers have a high level of feasibility, unikernels are on the verge of becoming a viable alternative [8].

A related study in 2020 by Kuo, Williams, Koller, and Mohan proposed that unikernels appeared to be a lightweight alternative to traditional or more commonly accepted virtual machines before unikernels came into the spotlight. The Lupine system, configured to perform as a unikernel, demonstrated superior performance in all categories, including image size, boot time, and application performance, when compared to other unikernels [9].

In 2018, Goethals, Sebrechts, Atrey, Volkaert, and Filip proposed that OSv is a stable and reliable unikernel for testing REST services and in-application processing workloads with request latency parameters, when compared to other unikernels, Rumprun and UniK. OSv supports a range of programming languages, including C++, Go, Python, and Java. Additionally, it has been observed that unikernels outperform containers in terms of response time. However, regarding memory usage, unikernels consume more than containers due to the additional overhead introduced by the kernel. In contrast, containers utilize the kernel from the host OS [10].

A similar study by Mavridis and Karatza (2019) concluded that lightweight virtualization technologies such as containers and unikernels can be considered alternatives to traditional virtual machines (VMs). The study employed tests involving the running of basic HTTP web servers in containers and unikernels, with the objective of measuring their performance in terms of the number of requests per second and their latency, based on the number and type of concurrent clients. The results of the tests conducted revealed that containers exhibited a higher request rate compared to unikernel. In the case of latency, unikernels such as MirageOS, IncludeOS, and Rumprun exhibited significantly lower and more stable results compared to containers. Among the other unikernels, MirageOS demonstrated superior performance, lower latency, and lower memory usage [7].

A similar study conducted by Kuenzer, Bădoiu, Lefeuvre, Santhanam, Jung, Gain, Soldani, Lupu, Teodorescu, Raducanu, Banu, Mathy, Deaconescu, Raiciu, and Huici in 2021 found that Unikraft exhibited superior performance, up to 1.7x to 2.7x, compared to using Linux as a guest. In their research, they indicated that Unikraft has a faster boot time, lower memory usage, and a smaller image size compared to Docker Container, Rumprun, Hermitux, Lupine, OSv, and Linux MicroVM [11].

In this study, the focus is on the performance testing of virtualization in cloud-based virtual machines. The gap in the previous studies lies in their exclusive focus on unikernel. These prior studies primarily examined unikernel running directly on the kernel, often referred to as specialized kernels [11]. This study specifically targets the area of resource usage within virtual machines and the unikernel in virtual machines, an aspect that has not been thoroughly explored. The testing parameters are divided into two categories: general tests and special tests. The general performance tests, derived from several referenced studies, include boot time and image size. The special tests represent a novel aspect of this research, measuring the CPU and memory utilization of the Virtual Machines. These special tests employ a load testing approach

by sending concurrent requests from multiple clients. The expected outcome is that unikernel will demonstrate better or superior performance compared to container, as observed in previous studies, even though unikernel run on top of a guest OS.

III. Architecture

A. System Architecture

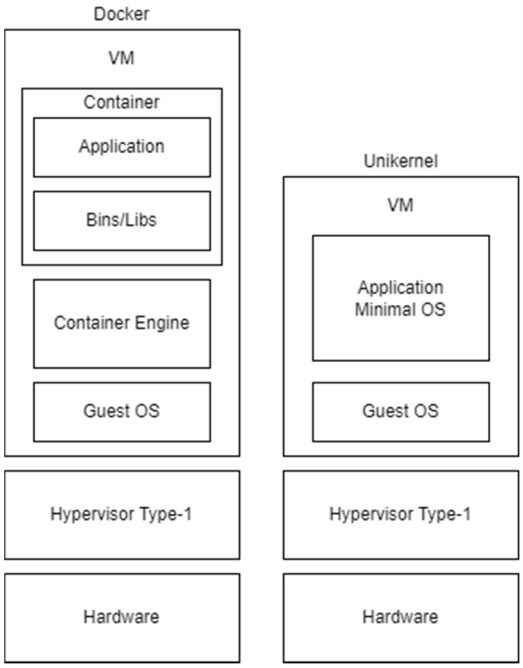

Fig. 2. Container and Unikernel Architecture in VM.

As illustrated in Figure 2, a type-1 hypervisor is embedded in bare metal using a cloud-based server in Google Cloud with the assistance of KVM (Kernel-based Virtual Machine), which serves as a type-1 hypervisor [12]. The virtualization runs on a virtual machine with the guest operating system being Ubuntu 22.04 LTS. The container that will be utilized in the testing phase is Docker and unikernel using Unikraft. Unikraft is a newly developed micro-library operating system that completely modularizes core OS functions, making it simple to customize the unikernel by including only the necessary components. It also offers a suite of composable, performance-focused APIs. Unikraft offers high performance and it's easy to use for developers [11]. Unikraft is run using QEMU, a type-2 hypervisor [2], to facilitate the background operation of applications.

Figure 1 illustrates the cloud architecture using Google Cloud. The virtual machine instance acts as a unikernel and container performance testing site. In the architecture, the deployments are applied monolithically to the prebuilt nginx v1.15.6 application, which serves as a web server application to perform general and special tests. Moreover, the instance designated as a benchmarking server end-user will transmit HTTP requests to the instances that function as unikernel and container for load testing. The objective is to evaluate the performance of each web server deployed, monitoring CPU and memory utilization. This will be achieved through Grafana as a metric visualization and Prometheus as a tool for collecting metric data with the help of the node exporter.

B. Testbed

Each instance of the software has the same specifications, ensuring that the results of the performance testing are relevant and comparable, as shown in Table I. The CPU model used in the virtual machine is the Intel® Xeon® CPU, with a clock speed of 2.20 GHz. The machine configuration in the VM instance in Compute Engine is the E2 type with the shared-core type. The E2 shared-core type is a cost-effective option that employs a

TABLE I. Virtual Machine Specifications

Attributes	Specifications
Region	us-central1
Zone	us-central1-a
Machine Configuration	E2-medium (2vCPUs, 1 core, 4GB memory)
OS	Ubuntu 22.04 LTS
Disk Type	Balanced Persistent Disk
Architecture	x86/64

virtio memory balloon device mechanism, enabling flexible resource management based on demand. It is well-suited to low-workload scenarios. The E2 shared-core type utilizes two concurrently running vCPUs, which are shared on a single physical core for a brief period. Each vCPU can utilize up to 100% of the CPU time within a short duration before returning from the allotted time limit. Tests were conducted using E2-medium shared-core to determine the results of VM performance with low specifications and a cost-effective configuration. In E2-medium, it supports two vCPUs, with each CPU processing time allocated to 50% of a fractional vCPU, resulting in a total of 100% of the CPU processing time being used [13].

C. Test Scenario

Testing is divided into two stages, namely using general tests and special tests.

a) General Test

```
#!/bin/bash

#Calculate Boot Time for Unikernel
IMAGE="unikraft.org/nginx:1.15"
PORTS="8080:80"

measure_boot_time() {
    START_TIME=$(date +%s%N)
    kraft run -p $PORTS $IMAGE &
    KRAFT_PID=$!
    END_TIME=$(date +%s%N)
    TIME_TAKEN=$((END_TIME - START_TIME))
    echo "Boot time: $TIME_TAKEN nanoseconds"
    kill -SIGTERM $KRAFT_PID
}
measure_boot_time
```

Fig. 3. Unikernel Boot Time Script.

The test compares the image size and boot time speed of each instance, carried out during both creation and execution phases. The image used is Nginx, version 1.15.6, in the form of a prebuilt image pulled from the Unikraft repository and Docker Hub. This image is run in both unikernel and

```
#!/bin/bash

#Calculate Boot Time for Container
IMAGE="nginx:1.15.6"
PORTS="8080:80"
measure_boot_time() {
    START_TIME=$(date +%s%N)
    docker run --name webserver -p $PORTS -d $IMAGE > /dev/null
    END_TIME=$(date +%s%N)
    TIME_TAKEN=$((END_TIME - START_TIME))
    echo "Boot time: $TIME_TAKEN nanoseconds"
}
measure_boot_time
```

Fig. 4. Container Boot Time Script.

container environments to test the speed of the boot time. The boot time testing procedure is initiated automatically via a bash script, ensuring minimal manual intervention. As shown in Figures 3 and 4, the script utilizes the date command and variables for the prebuilt images. It can also be observed that for unikernel, a termination signal is required to stop the process, as the image does not run in the background. In contrast, Docker can be configured to run in the background. The script used for Docker also considers the state from initialization until the container is fully running.

b) Special Test

This test is designed to assess the specific performance of a system against the overall instance performance, which is measured in terms of CPU and memory utilization metrics. The approach employed is that of a load testing methodology. In the context of information technology, load testing is the process of evaluating the system's capacity to withstand a peak load. Peak load is defined as the maximum number of users accessing the system concurrently at any given time [14].

The test is facilitated by a tool called wrk, which is an HTTP benchmarking tool that is capable of generating substantial loads when executed on a single multi-core CPU. This tool employs a multithreaded design and scalable event notification systems, such as epoll and kqueue, to achieve its functionality [15].

The collection of metric data on CPU and memory utilization is facilitated by the use of Prometheus and Node Exporter, which are exporters that assist in the collection of metrics from resources within the hardware that are targeted for analysis, namely VM instances of Container and Unikernel [16, 17].

These metrics then serve to inform the creation of visualisation dashboards, which are made available for use through the Grafana platform. This enables data visualisation [18]. The tool is installed on an end-user virtual machine instance that serves as a benchmarking server. The metrics were employed to ascertain the resource efficiency in computing each instance in HTTP requests received.

IV. Experimental Result

A. General Test

TABLE II. NGINX Image Size Comparison

Virtualization	Size
Container	109 MB
Unikernel	1.9 MB

The size of each image can be seen in Table 2. The image size testing of nginx images within each unikernel and container is a prebuilt image that has been provided or built without the necessity for us to create the image from scratch in order to use it directly. In the case of a unikernel, the image is in binary form, obtained from the unikraft repository and inspected using the du -s --si command with megabytes format. In contrast, for container that use Docker, the image is obtained from Docker Hub. Table 2 illustrates the size of each image in container and unikernel.

TABLE III. Boot Time Comparison

Virtualization	Boot Time (ms)
Container	508.355146
Unikernel	3.187648

In addition, it is necessary to determine the time required to boot using the prebuilt image in each virtualization environment. To this end, a bash script was developed that utilizes the date command and measures the time in nanoseconds. Subsequently, the time required to boot each container and unikernel was converted to milliseconds (ms). The results of this analysis are presented in Table 3.

B. Special Test

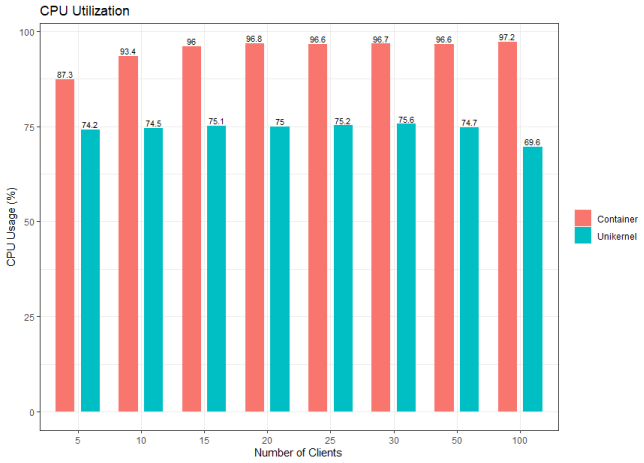

Fig. 5. CPU Utilization Comparison Results.

The results of the CPU utilization test for each container and unikernel are presented in Figure 5. The test employs a tool wrk, configured with a 30-second interval, a single thread, and clients executing in an increasing concurrency environment. The command is executed on the benchmark server, which represents the end-user.

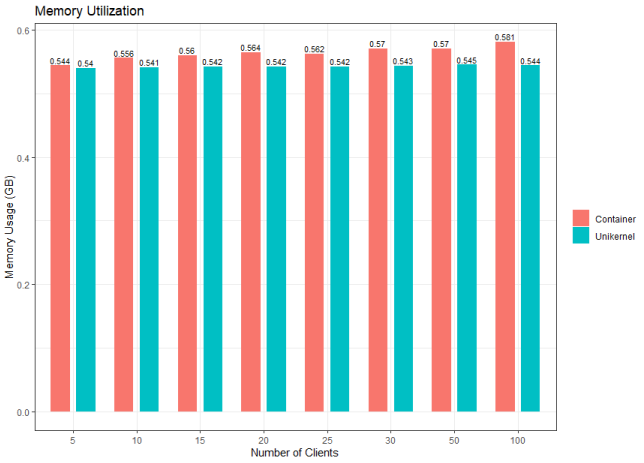

Fig. 6. Memory Utilization Comparison Results.

The results of the memory utilization test for each container and unikernel can be seen in Figure 6. Additionally, the test employs the wrk tool with the identical command utilized during the CPU utilization testing in the benchmark server, which is the end-user.

C. Analysis Results

Table 2 presents the results of an evaluation of the nginx image size test for container and unikernel. From the table, it is evident that the prebuilt images resulted in notable differences in image size. The image utilized in the container has a greater result than that of the unikernel.

The size of the image affects the performance of the boot time, which is based on the results of Table 3. The unikernel requires approximately 3 ms, while the container requires up to 508 ms. Finally, these results demonstrate that, despite the use of a guest OS by the unikernel, virtualization offers an effective and efficient performance compared to container.

Moreover, the use of a unikernel is also more efficient in terms of resource utilization, particularly in regard to CPU and memory usage within a virtual machine environment. Figure 5 illustrates that, in terms of CPU utilization, the performance of the unikernel is considerably superior to that of container in the face of high load requests, even when the number of clients reaches 50 and 100. This is evidenced by a notable decrease in utilization. Although the initial low-load request is caused by fixed overhead [19], it can be demonstrated that the unikernel is capable of overcoming heavy processing workloads [10]. Nevertheless, since the unikernel still shares the kernel with the guest OS, there is minimal overhead compared to a traditional unikernel, which can eliminate the overhead associated with context switching. This does not negate the potential for resource efficiency exhibited by the unikernel, which is characterised by minimal design overheads, such as a smaller image size and faster boot time in comparison to container [20].

The test employs a tool, wrk, for benchmarking purposes. This tool utilizes a single thread for 30 seconds, with an increasing number of clients making concurrent requests. The utilization of a single thread is attributable to the constrained resources of the shared-core e-2 medium VM instance employed [13], which precludes the necessity for context switching overhead and ensures the acquisition of consistent performance outcomes by means of thread management during testing [21, 22].

This also applies to the memory utilization efficiency generated by the unikernel, as illustrated in Figure 6. It can be observed that the memory utilization remains stable and relatively low in comparison to the significantly higher container utilization.

An intriguing observation is that the performance benefits of a unikernel operating system do not diminish when it is deployed on a guest operating system. Chen's [20] research indicates that the image size is approximately 5 MB, and the boot time is less than 50 ms for the unikernel. The image size and boot time obtained are consistent with the reference values presented in the study. This finding is consistent with the container in the research conducted by Chen [20], which had an image size of approximately 50 MB and a boot time of 800–100 ms. Although the size of the Nginx image in the container is larger in this study, it is a prebuilt image.

V. Conclusion

The objective of this research is to gain insight into the relative merits and drawbacks of virtualization technologies, with a particular focus on container and unikernel. The aim is to assess the performance of these technologies through empirical testing. The superiority of the unikernel over container is evident, despite the latter's use of prebuilt images and execution on a Guest OS with a cloud platform. This encompasses all aspects of testing, including general test parameters and special tests that have been defined. The superior performance of the unikernel is evidenced by its smaller image size and faster boot time, which enables it to outperform container in terms of CPU and memory utilization on each server.

VI. Future works

As a recommendation for future research, the approach to simulating traffic services can be conducted on a larger scale and utilising a server with high specifications and a unikernel architecture that eliminates the guest OS and utilises only the kernel. It is to be hoped that the performance of the unikernel will be demonstrably more efficient than that of the container.

References

[1] R.-S. Schmoll, T. Fischer, H. Salah, and F. H. P. Fitzek, "Comparing and evaluating application-specific boot times of virtualized instances," in 2019 IEEE 2nd 5G World Forum (5GWF), 2019, pp. 602-606.

[2] V. Aggarwal and B. Thangaraju, "Performance analysis of virtualization technologies in NFV and edge deployments," in 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2020, pp. 1-5.

[3] C. Wu, R. Buyya, and K. Ramamohanarao, "Cloud pricing models," ACM Computing Surveys, vol. 52, no. 6, pp. 1-36, Oct. 2019. doi:10.1145/3342103

[4] L. Chen, M. Xian, J. Liu, and H. Wang, "Research on virtualization security in cloud computing," IOP Conference Series: Materials Science and Engineering, vol. 806, no. 1, p. 012027, Apr. 2020. doi:10.1088/1757-899x/806/1/012027

[5] P.-J. Maenhaut, B. Volckaert, V. Ongenae, and F. De Turck, "Resource Management in a containerized cloud: Status and challenges," Journal of Network and Systems Management, vol. 28, no. 2, pp. 197-246, Nov. 2019. doi:10.1007/s10922-019-09504-0

[6] R. Behravesh, E. Coronado, and R. Riggio, "Performance evaluation on virtualization technologies for NFV deployment in 5G networks," 2019 IEEE Conference on Network Softwarization (NetSoft), Jun. 2019. doi:10.1109/netsoft.2019.8806664

[7] I. Mavridis and H. Karatza, "Lightweight virtualization approaches for software-defined systems and cloud computing: An evaluation of unikernels and containers," in 2019 Sixth International Conference on Software Defined Systems (SDS), 2019, pp. 171-178.

[8] M. Plauth, L. Feinbube, and A. Polze, "A performance evaluation of lightweight approaches to virtualization," Cloud Computing, vol. 2017, p. 14, 2017.

[9] H.-C. Kuo, D. Williams, R. Koller, and S. Mohan, "A Linux in unikernel clothing," in Proceedings of the Fifteenth European Conference on Computer Systems, New York, NY, USA: ACM, Apr. 2020, pp. 1-15. doi: 10.1145/3342195.3387526.

[10] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications," in 2018 IEEE 8th International Symposium on cloud and service computing (SC2), 2018, pp. 1-8.

[11] S. Kuenzer et al., "Unikraft: fast, specialized unikernels the easy way," in Proceedings of the Sixteenth European Conference on Computer Systems, 2021, pp. 376-394.

[12] "About nested virtualization | Compute Engine Documentation | google cloud," Google, https://cloud.google.com/compute/docs/instances/nested-virtualization/overview (accessed Jun. 15, 2024).

[13] "General-purpose machine family for Compute Engine | Compute Engine Documentation | Google Cloud - cloud.google.com."

[14] M. Arslan, U. Qamar, S. Hassan, and S. Ayub, "Automatic performance analysis of cloud based load testing of web-application & its comparison with traditional load testing," in 2015 6th IEEE

International Conference on Software Engineering and Service Science (ICSESS), 2015, pp. 140-144.

[15] "GitHub - wg/wrk: Modern HTTP benchmarking tool - github.com."

[16] Prometheus, "Prometheus - Monitoring system & time series database - prometheus.io."

[17] "GitHub - prometheus/node_exporter: Exporter for machine metrics - github.com."

[18] "Get started with Grafana Open Source | Grafana documentation - grafana.com."

[19] F. Moebius, T. Pfandzelter, and D. Bermbach, "Are Unikernels Ready for Serverless on the Edge?" arXiv preprint arXiv:2403.00515, 2024.

[20] S. Chen and M. Zhou, “Evolving container to Unikernel for edge computing and applications in Process Industry,” Processes, vol. 9, no. 2, p. 351, Feb. 2021. doi:10.3390/pr9020351

[21] S. K. Tesfatsion, C. Klein, and J. Tordsson, "Virtualization techniques compared: performance, resource, and power usage overheads in clouds," in Proceedings of the 2018 ACM/SPEC international conference on performance engineering, 2018, pp. 145-156.

[22] I. Molyneaux, The Art of Application Performance Testing: Help for Programmers and Quality Assurance. O'Reilly Media, Inc, 2009.