2019 Sixth International Conference on Software Defined Systems (SDS)

Lightweight Virtualization Approaches for Software-Defined Systems and Cloud Computing: An Evaluation of Unikernels and Containers

Ilias Mavridis
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
imavridis@csd.auth.gr

Helen Karatza
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
karatza@csd.auth.gr

***Abstract*—Software defined systems use virtualization technologies to provide an abstraction of the hardware infrastructure at different layers. Ultimately, the adoption of software defined systems in all cloud infrastructure components will lead to Software Defined Cloud Computing. Nevertheless, virtualization has already been used for years and is a key element of cloud computing. Traditionally, virtual machines are deployed in cloud infrastructure and used to execute applications on common operating systems. New lightweight virtualization technologies, such as containers and unikernels, appeared later to improve resource efficiency and facilitate the decomposition of big monolithic applications into multiple, smaller services. In this work, we present and empirically evaluate four popular unikernel technologies, Docker containers and Docker LinuxKit. We deployed containers both on bare metal and on virtual machines. To fairly evaluate their performance, we created similar applications for unikernels and containers. Additionally, we deployed full-fledged database applications ported on both virtualization technologies. Although in bibliography there are a few studies which compare unikernels and containers, in our study for the first time, we provide a comprehensive performance evaluation of clean-slate and legacy unikernels, Docker containers and Docker LinuxKit.**

***Index Terms*—Cloud, Software Defined Systems, Virtualization, Unikernels, Containers**

I. Introduction

Cloud computing is a well-established but constantly evolving technology. Virtualization is a fundamental element of cloud computing. It allows cloud users to share the same underlying infrastructure, as isolated software abstractions. It enhances portability, flexibility and scalability on cloud, while it improves the resource utilization and reduces the cost and energy consumption of the cloud infrastructure. Software Defined Systems (SDS) also use virtualization to provide a software abstraction for different subsystems including Software Defined Networking (SDN) [1], Software Defined Storage (SDStorage), Software Defined Servers (Virtualization) etc. [2].

The key objective of SDS is to provide a unified abstraction of different underlying systems as a single comprehensive system or resource pools [2], [3]. As far as cloud is concerned, the ultimate goal is to create a Software-Defined Cloud (SDCloud), where all the physical resources will be virtualized and controlled via software [4]. The core of any SDCloud is considered to be SDCompute [2]. SDCompute may employ any virtualization technology such as XEN or KVM hypervisors. New lightweight virtualization technologies could become part of SDCloud and co-exist or even replace traditional full-fledged VMs.

Containers and unikernels are two of the most dominant lightweight virtualization technologies. Containers are a more mature technology, compared to unikernels and offered as a product by various cloud providers. They share the same operating system of a host machine. Their main drawback is weak isolation. On the other hand, although they are not a new idea, unikernels, have recently gained momentum as a new virtualization approach in cloud computing. A unikernel is an immutable, single address space machine image, which can be deployed by a hypervisor and contains a library operating system. Unikernels are much smaller and have less overhead compared to full-flegded VMs, they also provide higher isolation compared to containers.

In this work, we present and evaluate the performance of four of the most wide-spread unikernels, two for each main category. We select OSv and Rumprun as legacy unikernels and MirageOS and IncludeOS as clean slate unikernels [4]. We also recognize Docker, as the dominant container solution and we deploy Docker containers on bare metal and on VMs [5]. Finally, we evaluate LinuxKit [6], which is a toolkit for building lean, customized OSes for containers. To provide a fair evaluation between all these different technologies, we create a custom http web server application for each system and we also evaluate the performance of SQL database application ported to containers and unikernels. Although in recent bibliography there are a few studies which present and compare lightweight virtualization technologies, for the first time, in this work we empirically evaluate the performance of unikernels of both categories, Docker containers and LinuxKit. We also present our experience with the available tools and the development process for all the aforementioned virtualization technologies.

The remainder of this paper is organized as follows. Section

978-1-7281-0722-6/19/$31.00 ©2019 IEEE

II provides an overview of the related research. Section III provides a background on light-weight virtualization technologies and tools. In Section IV the experimental results are demonstrated and finally, concluding remarks are presented in Section V.

II. Related Work

In recent years, there has been an increasing interest in lightweight virtualization technologies. Already, containers are a vital element of modern cloud computing and play an important role in emerging concepts, such as microservices. Unikernels later appeared as a more secure technology compared to containers, with less overhead compared to traditional VMs. In current literature, there is an increasing number of studies which propose different implementations of unikernels, compare and evaluate their performance and present, well-suited to unikernels, use cases.

In [7], Goethals et al. compared the performance and memory consumption of OSv and Docker containers. Although they did consider Rumprun and Unik, due to several issues, they eventually evaluated only OSv unikernels. They conducted various experiments with REST services and heavy processing workloads, written in Java, Go, and Python, on both OSv and Docker. For single-threated REST service stress tests, they found that unikernels perform better than their corresponding containers. For heavy workloads, almost all unikernels performed equally as their container counterparts. Concerning memory, unikernels consume significantly more than containers.

Xavier et al. [8], experimentally evaluated the time of provisioning multiple KVM, Docker and OSv instances concurrently, on an OpenStack cloud platform. They found that, for their experimental scenarios, OSv outperformed the other virtualization options. Moreover, they recognized the cloud platform as a factor that significantly affects the overall provisioning time.

The authors of [9], used different benchmarks to evaluate KVM, Docker, LXC and OSv in terms of processing, storage, memory and network. Although OSv values are missing on several experimental results, overall, they found that containers generally achieved better performance compared to other virtualization technologies. However they pointed out that containers do not offer strong security.

In [10], Acharya et al. indicated virtualization as a key element of new networking solutions, such as SDN and Network Functions Virtualization (NFV). Similar to [9], they experimentally evaluated the CPU, memory and Input/Output (I/O) performance of KVM, Docker, rkt, Rumprun and OSv, on both x86 and ARMv8 platforms. However, Rumprun and OSv unikernels are only evaluated on x86 platform. The overall results showed that containers are the fastest virtualization technology followed by unikernels and KVM respectively.

The performance and resource utilization of KVM, Docker and IncludeOS are compared on both Intel and AMD servers in [11]. The author, based on various TCP web-service and UDP network performance experiments, concluded that Docker performed better and consumed fewer resources compared to the most unstable IncludeOS. However, IncludeOS performed better and consumed fewer resources compared to Ubuntu KVM VM.

In [12], Enberg used Netperf and Memcached to evaluate the performance of KVM, Docker and OSv unikernels for raw networking and network intensive application. He noted that, in general, containers are the fastest virtualization technology for network intensive applications and unikernels are the fastest hypervisor-based technology.

Plauth et al. [13] deployed Nginx servers and Redis applications on Ubuntu, LXD, Docker, Rumprun and OSv, running natively or on KVM and XEN VMs. They also deployed an http-conduit server on MirageOS, although a comparison between a conduit server and the heavily optimized Nginx server is unequal. They conducted various experiments and measured different metrics. Concerning application throughput, they found that most unikernels performed at least equally well as or even better than containers. Also, they found that whereas Docker achieved the shortest startup times, unikernels have tiny image sizes.

In [4], a tool chain called PHP2UNI is presented, which builds IncludeOS and Rump Kernels unikernel images from PHP files. The authors found that PHP2Uni-IncludeOS had the lowest memory usage, then followed HHVM, Docker, the comparable solution running directly above the OS and last rump-based solutions, which consumed much more memory. As far as performance is concerned, for two different uses cases of high throughput and server-side computation needed, PHP2UNI achieved the lowest response times.

A lightweight and flexible runtime, for efficient mobile offloading in Mobile Edge Computing or Mobile Fog Computing was presented in [14]. The runtime was called Android Unikernel and followed the idea of supporting various applications in one unikernel (Rich-Unikernel). The authors evaluated the proposed runtime with four application benchmarks and observed that it had lower boot-up time, memory footprint, image size and energy consumption, compared to Android VM and Android container.

Finally, a Software-Defined Security strategy for cloud, based on unikernels, has been proposed in [15]. The authors described a framework which supports on-the-fly generation of unikernel images, which integrate protection mechanisms through their source code and as boot arguments. They evaluated the performance of the proposed strategy, by deploying an http web server, rather than regular Virtualbox VMs and Docker containers. They concluded that unikernels are well suited to minimize risk exposure with relatively limited costs.

III. Background

A. Virtualization Technologies

Virtualization was originally introduced by IBM in 1960 and the first virtual machine (VM) was created seven years later [16]. Nowadays, modern VMs follow the same principles. A hypervisor (also called virtual machine monitor VMM) creates and manages VMs and allows multiple OSes to run

on the same underlying hardware. The machine, in which the hypervisor is running, is called host machine and each VM running on a host machine is called ”guest machine”.

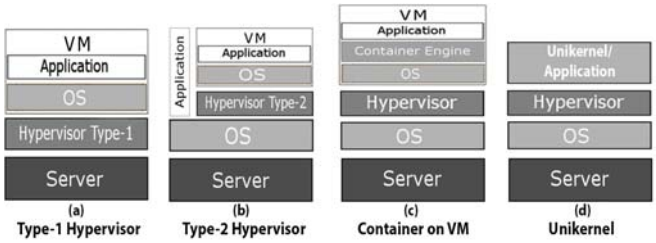

Fig. 1: Virtualization technologies and methods.

1) Virtual Machines: There are two types of hypervisors, Type-1 (Fig. 1 a) and Type-2 (Fig. 1 b). Type-1 hypervisors, also called ”bare-metal hypervisors”, are directly installed on the hardware, whereas Type-2 hypervisors, also called hosted hypervisors, run on top of the host OS. Each VM runs its own OS and is isolated from the others. Several VMs can coexist simultaneously on the same physical machine and be managed by the same hypervisor. Due to its nature, a VM can easily migrate and be duplicated among physical servers, something that simplifies systems maintenance and failure recovery.

Although in our study we evaluated the lightweight virtualization technologies of containers and unikernels, we used KVM (Kernel-based Virtual Machine) to run Docker containers on top of VMs (Fig. 1 c). Running containers on top of VMs is a technique which is mainly used to drastically improve containers' isolation [5]. KVM is open source software and it is not clear whether it is a Type-1 or Type-2 hypervisor, because it loads a kernel module into Linux kernel and transforms the host into a hypervisor. Every VM is treated by the host OS as a regular Linux process. KVM uses a modified QEMU (Quick Emulator), which uses CPU extensions, to emulate all the resources of the VMs and supports paravirtualization through Virtio [17].

2) Containers: Containers, in contrast to VMs, are a virtualization technology which does not require a hypervisor. Containers operate at the OS level and are managed by a container engine. They are more lightweight and can be created significantly faster than VMs, mainly because they do not run an instance of a complete OS. Containers share the host's OS kernel and run multiple isolated processes over the same host [5]. To control resource consumption and to provide isolation, containers rely on control groups (cgroups) and namespaces features of Linux kernel. Containers' main advantages, compared to VMs, are low overhead and fast boot process, however, containers do not excel at providing isolation as VMs do.

In our study, we focus on Docker as it is an open source and widely used container platform [18]. Docker also provides Docker Hub, which allows to easily share and manage Docker containers among users. In 2017, Docker announced LinuxKit [19], a toolkit to build minimal Linux subsystems that only include exactly the components which the runtime platform requires [20]. LinuxKit images can be deployed by various hypervisors, cloud providers and on x86_64 and arm64 bare metal. In our experiments, we run LinuxKit images with QEMU.

3) Unikernels: Unikernels are actually specialized, single-address, immutable, minimal VMs, built by using library OSes [21]. While the kernel size of traditional, general-purpose operating systems is steadily getting bigger and more complicated the unikernels approach is based on the concept of library operating systems [22]. Unikernels contain a single application, they do not have separate user and kernel address space and hold only the code and libraries required to run the application they are built for (Fig. 1 d). Also, because unikernels are built to run only a specific application, which is hard coded into the image, they have small file size and footprint, minimal attack surface and booting time as low as 10 ms, according to IBM [21], [23]. This almost instant boot time of unikernels is exploited by Jitsu (Just-in-Time Summoning of Unikernels), which is presented in [24]. Jitsu is actually a DNS server which launches the unikernel that will service an address, when it receives the DNS lookup request for that address.

Traditional VMs, unikernels and containers, provide different levels of performance and isolation, have different footprints, boot times, density and maturity. VMs have been used in production for many years, they provide hardware level isolation, however they are usually as big as hundreds of MegaBytes to GigaBytes, have relatively high resource consumption and significant boot times. Containers have high density, lower resource consumption and boot times with their main drawback to be poor isolation. Unikernels, although under development, promise hardware level isolation and higher performance than containers. On the other hand, unikernels have limitations resulting from their own design and even when they are eventually production ready, they will not be a panacea.

4) Unikernels and tools: There is already a big number of available unikernel solutions and probably more will appear going forward [25]. We can classify unikernels in two general categories, clean-slate and legacy. Clean-slate unikernels follow a minimalistic approach and are usually language specific. Clean-slate unikernels require re-implemented OS functions and more effort to develop applications. However, they produce smaller images with lower resource requirements compared to legacy unikernels [25]. On the other hand, legacy unikernels are POSIX compatible and re-use existing libraries. These unikernels have larger images and higher resource requirements but are used to easily port applications to unikernels, by cross-compiling existing applications [7]. In our study we evaluated the clean-slate MirageOS and IncludeOS and compared their performance to OSv and Rumprun legacy unikernels.

MirageOS is one of the first library operating systems used to construct unikernels [22]. It follows a clean-slate approach, can be deployed on XEN, KVM and lightweight hypervisors,

TABLE I: Unikernel Tools.

Unikernels and Tools	Languages	Hypervisors and Cloud
OSv	Java, Node, Ruby, Scala, JavaSCript, C, C++	KVM, XEN, VirtualBox, Vmware, ESXi, Amazon EC2, Google Cloud
Rumprun	C, C++, Erlan, Go, Java, JavaScript, Node.js, Python, Ruby, Rust	KVM, XEN
IncludeOS	C++	KVM/QEMU, VirtualBox, hvt, ESXi, Google Cloud, Bochs, OpenStack
MirageOS	OCaml	KVM/QEMU, Xen, hvt
Unik	Node.js, Go, Java, C/C++, Python, OCaml	AWS, QEMU, VirtualBox, PhotonController, XEN, hvt, vSphere, OpenStack, Google Cloud

as well as on Amazon EC2 and Google Compute Engine [26]. MirageOS unikernel images are written in OCaml and there are currently available almost 100 MirageOS libraries [27]. MirageOS-based solutions are proposed in [15], [21], [24], [28], [29] and [30].

Rumprun uses the kernel-quality drivers of rump kernels to build unikernels [31]. Rump kernels are originated from NetBSD, which was specifically designed to be ported to as many hardware platforms as possible [32]. Rumprun supports various languages (Table I) and allows porting POSIX software to unikernel [33]. Rumprun unikernels can run on Xen and KVM hypervisors, as well as on bare metal. In [34], Rumprun unikernel was used as a building block for multiserver systems and in [35] the authors implemented a Rumprun-based Edge-hosted Personal Service.

The OSv cloud-focused unikernel is presented in [36]. It provides execution environments for many languages and supports different hypervisors and cloud platforms (Table I). OSv provides a more general-purpose unikernel base and produces large unikernel images which can be used to run any application that can run in a single process [32]. In [37], there is a repository with many popular applications already ported to OSv and ready to be deployed. In [38], the authors presented an architecture for provisioning OSv-based Unikernels in OpenStack.

IncludeOS is still (v0.12 as of February 2019) an under development library operating system for writing unikernel applications in C++ [39]. IncludeOS has many features in common with MirageOS. However, it supports more hypervisors and cloud providers compared to MirageOS and provides an orchestration tool which is called mothership [40].

Unik is not a unikernel or a library operating system, but a tool for compiling and deploying unikernels [41]. Unik tried to make the development of unikernels as easy as Docker did for containers. It supports Rumprun, OSv, IncludeOS and MirageOS unikernels and many backends (Table I). UniK is fully controllable through a REST API, supports integration with orchestration tools such as Kubernetes and promises a high degree of pluggability and scalability [32].

Solo5 has originally started from IBM Research and is a unikernel base [42]. Solo5 is actually the lowest layer of the unikernel which interacts with the hypervisor [43]. Solo5 enabled MirageOS unikernels to run on either Linux KVM/QEMU or on a specialized unikernel monitor which is originally called ukvm. The ukvm monitor was presented in [24] and it is a minimal specialized monitor, which only contains what the unikernel needs, both in terms of interface and implementation.. According to [23], ukvm is less than 5% of the code size of a traditional monitor, and due to its minimalism, it has higher isolation, lower complexity and faster booting time for unikernels. Ukvm, since Solo5 version 0.4.0, is called Solo5-hvt (hardware virtualized tender) and is referred to as tender and not as monitor anymore, thus it thereafter supports additional tenders, such as sandboxed process tender (spt) [44]. Since 2017 Solo5 also supports IncludeOS unikernels.

IV. Evaluation

In this section, we present the hardware and software setup of our system, the applications which were deployed in all visualization technologies, the benchmarks for both http web server and SQL server applications and discuss the experimental results.

A. Experimental Setup

All the experiments were executed on a 3.7 GHz Intel CORE I3-6100 processor with 8 GB of DDR4 DIMM RAM, 240GB Western Digital SSD and 100 Mbps Ethernet. A connected, second machine was used as client, to stress the server and record the experimental results. The server was running Ubuntu Server 16.04.3 LTS. In cases where traditional VMs were deployed, the guest machine was running the lightweight, general purpose OS, Alpine 3.9.0 Virtual edition.

We used QEMU 2.5.0 for VMs and Docker 5:18.09 for containers, on both Ubuntu and Alpine. LinuxKit images created with LinuxKit 0.6+ and unikernel images were deployed with OSv 0.52.0, the latest available version of Rumprun in January 2019, IncludeOS 0.14.0 and MirageOS 3.4.0.

For the SQL database application, we selected MySQL 5.6.38 and the MyISAM storage engine, since they were available and stable in all virtualization technologies, except for LinuxKit in which we necessarily used the Alpine equivalent to MySQL 5.6.38, MariaDB v10.1.32. For the http web server use case, we developed a simple http web server in compatible programming languages. In every case we used the default Linux network stack, without adjusting any of its predefined variables.

Every VM and container was limited to use one CPU core and 1GB of RAM. The benchmarks used to measure the performance of the two use cases were sysbench with the OLTP (online transaction processing) component [45] for MySQL and wrk [46] to measure the performance of the http web server applications.

B. Http Web Server

1) Http web server application images development: Widely used web server applications, such as nginx [47], have already been ported to legacy unikernels, including OSv and Rumprun. However, because porting this kind of applications to clean-slate unikernels requires a significant amount of effort, there are no equivalent MirageOS and IncludeOS implementations available.

To provide a fair comparison among all virtualization approaches, we developed a basic http web server application for each case. Our application was built to respond to http requests and to send the requested webpage to clients. If the requested webpage does not exist, the http web server returns an error message. In our evaluation, we used a static html webpage.

To create the unikernel applications, first we used the tools provided by each unikernel ecosystem and second, we developed again the same unikernels with Unik. Although we addressed many issues, we managed to create and run all four different unikernels with Unik and Virtualbox hypervisor. Unik has some very interesting capabilities, nevertheless it is still in experimental phase. We may note that, in its current stage, Unik is far from providing a Docker-like and easy way of building unikernels.

First, we wrote the http web server application for the MirageOS unikernel. MirageOS consists of OCaml libraries which link with a runtime to form a unikernel [48]. These libraries are managed via the OPAM package manager [49]. Ocaml is a general-purpose programming language with an emphasis on expressiveness and safety [50]. MirageOS also provides a repository with example codes in [51]. We adapted the tutorial code of [51] to create our application on OCaml. Before building the unikernel, we configured it with boot target hvt and then we built and ran the unikernel image with hvt. The MirageOS unikernel used a bridged TAP virtual network interface.

For the second clean-slate unikernel, IncludeOS, we wrote a similar C++ version of the previous OCaml http web server application. In IncludeOS main github repository [52], there are also source code examples of simple applications, which we adapted to create our own application. IncludeOS is the second unikernel, which is compatible with hvt. In this case as well, we built and ran the IncludeOS unikernel with a bridged TAP virtual network interface.

OSv and Rumprun unikernels, in contrast to MirageOS and IncludeOS, support a wide range of programming languages (Table I). Also, both of them maintain on-line repositories with useful examples and applications for different languages [37], [53]. We selected Java to develop our application for both legacy unikernels. We also used two additional tools. Apache Maven [54] automation tool to build the Java application and Capstan [55] to run OSv. Our experimental setup for these unikernels, was similar to the previous ones, but in these cases we deployed them with QEMU and not with hvt.

Finally, on Docker and LinuxKit, we used the same Docker container image. We developed an http web server in C++ and created a Docker image. With this image, we created a LinuxKit image. We ran Docker containers with CPU and memory limitations, with their default bridges. For LinuxKit images, we applied the same source limitations and ran them with QEMU and bridged TAP virtual network. In the case of Docker containers running on top of a VM, we used KVM/QEMU to deploy Alpine VM with bridged TAP virtual network interface and limited resources to 1 CPU Core and 1GB ram. We selected Alpine OS as guest OS, mainly because of its minimal design and high performance.

2) Performance evaluation of http web server applications: Table II presents four different metrics, for requests per second and latency, for the http web server application, for different virtualization technologies and number of concurrent clients. For every case we can see the average and max value of requests per second and latency, as well as the standard deviation and the percentage of values appeared in +/- standard deviation. If we observe the requests per second values, we can easily recognize that Rumprun unikernels achieved significantly lower performance, compared to the other virtualization technologies. Compared to OSv unikernel, Rumprun achieved in average 18 times fewer http requests per second and has twice as high latency in average.

To understand why Rumpruns performance is so low, we have to keep in mind that the http web server application mainly uses the TCP/IP network stack of the system. Also, worth noting is that Rumprun is using the unmodified TCP/IP network stack of NetBSD. In a similar to ours experimental scenario, the authors of [10] measured the bandwidth of the TCP network protocol for Docker, rkt, KVM, OSv and Rumprum. They stressed the system with the Iperf network benchmarking tool and, for their experimental setup, they found that Rumprun achieved 32 times lower network performance compared to OSv. To further investigate how this significant performance difference is affected by the type of http web server application, we deployed the highly optimized nginx web server on Rumprun and OSv. The results, which are not presented in this work due to space limitations, followed a similar pattern to the previous ones. In average, nginx running on OSv achieved 5 times more requests/sec compared to Rumprun.

Concerning unikernels, OSv achieved the highest average number of requests per second and then MirageOS follows with less than 2.5% difference in the average number of requests per second in total, for different number of clients. However, OSv presents almost 3 times higher standard deviation compared to MirageOS. Docker on bare metal achieved the highest average number of requests per second for all different experimental configurations, then OSv follows, LinuxKit, MirageOS, Docker on VM-Virtio, Docker on VM, IncludeOS and finally Rumprun with important difference in its performance. For more than 20 concurrent clients, OSv achieved in average 2.75% higher number of requests per second compared to Docker. MirageOS, for even higher number of concurrent clients (90 - 100) achieved equal and even higher number of requests per second compared to Docker.

In the second half of Table II we can see the latency metrics,

TABLE II: Http Web Server.

		Container on Bare Metal	Container on VM	Container on VM-Virtio	LinuxKit	OSv	Rumprun	IncludeOS	MirageOS	Container on Bare Metal	Container on VM	Container on VM-Virtio	LinuxKit	OSv	Rumprun	IncludeOS	MirageOS
		Reqs/Sec								Latency (ms)							
1 client	Average	1210	1200	1410	1340	0.5	224.73	741.2	890	0.89	0.59	0.49	0.52	954.29	4	0.96	94.85
	Max	1330	1300	1480	1510	10	232	1100	1510	62.09	8.26	8.96	8.36	1010	9.79	46.33	1080
	Stdev	218.55	60.93	129.53	198.21	2.24	5.97	380.69	446.2	3.57	0.32	0.15	0.27	214.93	0.199	2.47	222.05
	+/- Stdev	95.92%	71.29%	97.41%	83.00%	95.00%	85.00%	74.37%	66.67%	98.94%	98.86%	99.39%	96.51%	95.00%	96.56%	98.16%	89.58%
10 clients	Average	4890	4000	3910	4370	3530	247.58	2930	4000	1.96	3.56	1.44	3.9	3.52	21.33	1.83	2.61
	Max	5040	4760	5020	5030	3950	262	3520	4770	207.43	207.67	206	417.65	92.17	43.87	19.91	22.34
	Stdev	406.91	360.23	491.92	476.62	706.26	7.68	359.08	416.54	10.73	18.04	7.61	25.17	5.42	9.03	1.24	1.49
	+/- Stdev	96.04%	69.31%	81.50%	53.00%	87.00%	88.00%	75.50%	75.00%	98.95%	97.87%	99.61%	98.20%	96.94%	58.16%	88.97%	92.40%
20 clients	Average	4850	3880	4390	4570	4910	246.46	3190	4380	2.82	6.97	3.37	6.89	5.13	32.8	3.02	4.57
	Max	5030	4690	5030	5040	5460	260	3690	4860	412.09	414.33	414.3	846.83	125.58	63.07	20.07	22.27
	Stdev	502.15	401.37	649.83	511.09	1140	6.87	304.75	337.32	15.26	28.42	20.69	42.56	7.56	16.27	1.71	1.41
	+/- Stdev	91.00%	72.00%	72.00%	79.00%	86.50%	92.00%	75.00%	71.00%	98.89%	95.51%	98.36%	97.32%	96.39%	58.55%	77.31%	88.14%
30 clients	Average	4880	4000	4500	4610	4950	248.1	3170	4550	9.96	20.37	3.59	29.92	7.32	37.05	4.32	6.56
	Max	5030	4740	5030	5050	5470	262	3800	4970	823.98	1690	416.32	1670	209.11	70.13	30.43	21.47
	Stdev	261.14	399.77	578.12	708.12	1130	7.84	382.33	324.36	58.87	97.14	20.24	131.86	9.55	16.32	2.59	1.39
	+/- Stdev	88.00%	73.00%	73.00%	80.00%	87.50%	72.00%	74.69%	82.00%	96.98%	94.73%	98.13%	94.76%	96.27%	60.82%	78.39%	87.80%
40 clients	Average	4710	4070	4570	4640	4930	243.25	3150	4600	11.88	12.21	6.43	34.1	9.74	45.48	5.63	8.64
	Max	5050	4790	5030	5050	5460	252	3910	4990	833.4	864.03	833.24	1730	303.06	69.48	40.79	24.21
	Stdev	561.95	334.99	555.06	743.04	1180	7.08	459.9	269.31	61.24	50.74	41.81	159.52	12.94	14.32	3.47	1.63
	+/- Stdev	87.00%	69.00%	77.50%	87.23%	86.50%	83.00%	74.37%	70.00%	95.95%	94.22%	97.54%	95.54%	96.35%	67.70%	79.47%	85.71%
50 clients	Average	4820	3890	4420	4790	4920	248.68	3370	4660	7.39	23.66	6.05	36.5	12.09	36.7	6.36	10.66
	Max	5030	4700	5030	5050	5630	262	3920	5090	833.53	1740	832.25	1730	452.02	65.67	30.42	31.08
	Stdev	403.69	390.98	566.92	717.27	1180	9.14	290.23	330.49	36.41	120.6	40.37	158.53	16.35	15.74	3.35	1.74
	+/- Stdev	90.00%	74.00%	69.50%	92.55%	86.00%	82.00%	79.40%	78.00%	96.48%	95.65%	97.68%	95.10%	96.49%	60.05%	70.99%	85.55%
60 clients	Average	4810	3890	4540	4600	4930	244.74	3380	4730	8.37	17.71	4.18	32.68	15.22	37.02	7.7	12.63
	Max	5030	4400	5030	5040	5460	270	3980	5080	841.26	1720	416.27	1710	616.23	70.71	41.79	33.34
	Stdev	421.35	331.3	511.95	727.79	1180	10.21	337.29	251.74	45.03	92.54	23.55	142.54	27.02	15.07	4.16	1.67
	+/- Stdev	87.00%	70.00%	80.50%	94.68%	86.50%	84.00%	76.88%	79.00%	96.56%	95.22%	97.84%	94.82%	97.26%	62.44%	70.94%	82.96%
70 clients	Average	4790	3920	4410	4750	4950	247.32	3400	4700	7.17	29.1	14.63	28.2	18.01	42.1	8.34	14.78
	Max	5070	4800	5020	5040	5460	262	3920	5180	826.7	1670	1670	1730	716.73	69.35	35.56	32.87
	Stdev	478.16	437.48	575.64	860	1160	15.68	319.54	341.67	33.93	125.45	95.55	130.67	34.12	12.93	4.29	2.25
	+/- Stdev	90.00%	70.00%	68.50%	95.51%	85.50%	94.00%	82.41%	81.00%	96.32%	94.49%	96.81%	95.35%	97.47%	69.05%	67.90%	83.83%
80 clients	Average	4860	3950	4370	4740	4980	241.84	3350	4760	24.83	31.32	14.87	16.34	21.45	47.79	8.35	16.73
	Max	5070	4800	5040	5030	5460	252	3880	5300	1670	1730	1660	1710	870.97	76.14	43.76	42.54
	Stdev	560.01	357.27	602.5	672.8	1150	12.93	387.05	321.52	117.74	145.07	91.6	103.22	45.57	11.31	4.47	2.13
	+/- Stdev	95.74%	76.00%	71.50%	91.49%	87.50%	92.00%	81.91%	90.00%	95.19%	95.53%	96.46%	96.86%	97.59%	68.88%	70.13%	86.26%
90 clients	Average	4740	3820	4510	4790	4910	236.32	3370	4760	19.58	32.76	14.55	27.19	21.82	31.84	8.07	18.85
	Max	5030	4610	5030	5050	5460	252	4000	5400	1670	1720	1690	1730	816.91	75.21	39.79	66.29
	Stdev	627.82	384.07	468.94	632.88	1210	16.29	273.78	359.1	89.86	129.98	88.27	132.99	33.78	16.19	4.25	2.72
	+/- Stdev	92.55%	68.00%	66.50%	91.49%	86.50%	86.00%	74.87%	73.00%	95.03%	94.15%	96.31%	95.59%	97.25%	59.92%	69.37%	87.45%
100 clients	Average	4780	3830	4590	4690	4950	240.87	3480	4780	18.25	25.16	13.83	20.86	28.65	39.59	8.36	20.74
	Max	5040	4640	5030	5070	5460	260	3890	5160	1660	1740	1670	1740	1190	73.85	44.39	43.13
	Stdev	352.29	298.72	716.7	920	1150	12.92	383.75	320.8	80.29	117.5	78.13	117.58	67.42	15.54	4.52	2.37
	+/- Stdev	87.00%	71.00%	87.37%	93.26%	86.00%	89.00%	82.88%	84.00%	93.97%	94.65%	96.30%	96.44%	97.68%	62.75%	70.72%	81.72%

for the same experiments. The highest average latency values achieved by Rumprun and the lowest by IncludeOS. It is worth mentioning that the overall max latency values for MirageOS, IncludeOS and Rumprun were ranged between 40 and 80 milliseconds whereas, for the other virtualization technologies exceeded one second. Moreover, we can observe that for these three unikernels, the overall standard deviation of latency is very low. MirageOS running on ukvm achieved the lowest latency and only 5% lower average request rate from Docker on bare metal.

TABLE III: Memory usage in MB per unikernel.

	OSV	Rumprun	IncludeOS	MirageOS
Memory consumption (Mbytes)	185	97	16	7

Additionally, for this experimental scenario, we used the ”free -m” Linux command, to measure the amount of memory that each unikernel needs in order to operate. The values presented in Table III, are in alignment with our initial claims, that clean-slate unikernels, such as MirageOS and IncludeOS, have lower resource consumption than legacy unikernels.

C. SQL Database

In the second series of our experiments, we deployed MySQL server applications with Docker containers, OSv and Rumprun unikernels and last LinuKit. In contrast to the http web server experiments, in which we developed the http web server applications, in this series we used, already ported to containers and unikernels MySQL server applications. In Docker Hub there are more than 2 million public images available to download, whereas there are more than 36 commonly used applications ported to Rumprun and more than 105 application ported to OSv. Our experience with OSv and Rumprun, showed that not only there are more OSv ported applications than Rumprun, but also that OSv applications are more functional. For example, MySQL application on Rumprun supports only MyISAM storage engine.

We initially used sysbench OLTP to populate our database with 500.000 rows of random data. Then, we quantified the performances of each virtualization solution for both read/write and read-only requests. In all cases we used the same MySQL configuration file. Fig. 2 illustrates the requests per second recorded values for read/write and read-only requests. In case of read/write requests, sysbench selects queries from five SELECT queries, two UPDATE queries, a DELETE query and an INSERT query. As we noticed in our results, the ratio was about 65% reads, 25% writes and 10% other queries.

As it would be expected, read-only requests had the highest request/sec rate, compared to more complicated read/write requests, in all cases. By observing Fig. 2, we recognize that Rumprun achieved by far the lowest performance. In average containers running on bare metal, LinuxKit and VM achieved higher requests/sec rate and lower average time for request (Fig. 2, Table IV). For this experimental scenario, OSv achieved in average 27% fewer requests/sec compared to Docker on bare metal. Worth noting is that, although read-only requests to container on bare metal and on paravirtualized VM had almost the same performance, there is a much bigger difference for read-write requests between them. Most probably this happens because the hypervisor used buffering. Regarding latency, except for Rumprun, there is no big difference for the other virtualization technologies.

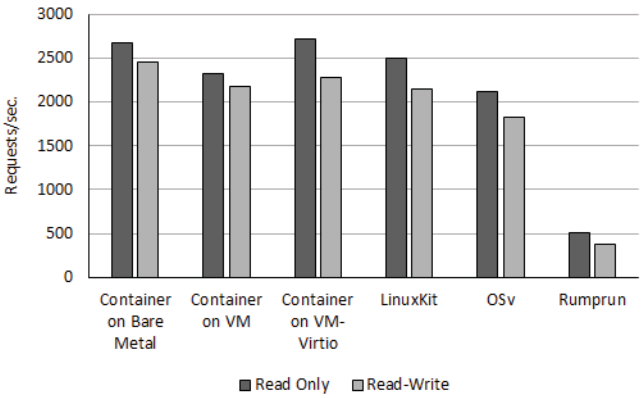

Fig. 2: SQL server performance.

TABLE IV: SQL server latency in ms.

	Read Only			Read-Write		
	Avg	Max	95%	Avg	Max	95%
Container on Bare Metal	5.22	40.63	6.7	5.7	25.14	6.97
Container on VM	6.02	81.23	9.61	8.73	74.92	11.73
Container on VM-Virtio	5.13	63.95	6.58	8.31	223.75	13.74
LinuxKit	5.59	21.15	6.05	8.84	79.37	10.77
OSv	6.61	68.46	7.65	10.4	68.49	16.32
Rumprun	27.59	78.61	28.68	49.82	233.06	138.71

V. Conclusions

Towards Software Defined Cloud Computing, virtualization technologies do play an important role. Lightweight virtualization technologies, such as containers and unikernels, provide an interesting alternative to traditional VMs. Containers have already been used in cloud for several years. Lately, the old idea of unikernels tries to find a new application in the cloud environment.

In this study, for the first time, we used the well-known Docker container engine to deploy containers in 3 different ways and we compared their performance to 4 different unikernel solutions. We deployed Containers on bare metal, containers on minimalistic KVM VMs and Linuxkit. We developed a simple http web server and we also used SQL database application images, to compare the performance of containers to both clean-slate and legacy unikernels.

Our experimental evaluation showed that in almost all cases containers had higher request rate compared to unikernels.

However, OSv and MirageOS achieved comparable performance to containers and in some cases marginally higher than them. As for latency, MirageOs, IncludeOS and Rumprun showed significant lower and more stable values compared to containers. Between unikernels, MirageOS showed high performance, the lowest latency and the lowest memory usage, with its main drawback to be that it only supports OCaml. On the other hand, OSv supports many languages, has a public directory of already ported applications, achieved lower latency than containers and comparable performance but significantly higher memory utilization.

In our evaluation we tried different tools through the development process. Maven and Capstan help us run OSv unikernels. We employed Unik to develop and run, all the four kinds of unikernels. Unik, as most unikernel technologies, is currently under development and even if it is very promising, it is far from fulfilling its goals. Our empirical evaluation shows that each technology provides benefits but also has drawbacks at different levels. Each of these lightweight virtualization technologies can be used appropriately in different cases.

REFERENCES

[1] Michel, Oliver, and Eric Keller. "SDN in wide-area networks: A survey." 2017 Fourth International Conference on Software Defined Systems (SDS). IEEE, 2017.
[2] Jararweh, Yaser, et al. "Software defined cloud: Survey, system and evaluation." Future Generation Computer Systems 58 (2016): 56-74.
[3] Freris, Nikolaos M. "A software defined architecture for cyberphysical systems." 2017 Fourth International Conference on Software Defined Systems (SDS). IEEE, 2017.
[4] Pasquier, Thomas, David Eyers, and Jean Bacon. "PHP2Uni: Building Unikernels using Scripting Language Transpilation." 2017 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2017.
[5] Mavridis, Ilias, and Helen Karatza. "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing." Future Generation Computer Systems 94 (2019): 674-696.
[6] https://github.com/linuxkit/linuxkit
[7] Goethals, Tom, et al. "Unikernels vs Containers: An In-Depth Benchmarking Study in the context of Microservice Applications." 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2). IEEE, 2018.
[8] Xavier, Bruno, Tiago Ferreto, and Luis Jersak. "Time provisioning evaluation of KVM, Docker and unikernels in a cloud platform." 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2016.
[9] Morabito, Roberto, Jimmy Kjllman, and Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison." 2015 IEEE International Conference on Cloud Engineering. IEEE, 2015.
[10] Acharya, Ashijeet, et al. "A Performance Benchmarking Analysis of Hypervisors Containers and Unikernels on ARMv8 and x86 CPUs." 2018 European Conference on Networks and Communications (EuCNC). IEEE, 2018.
[11] Czipri, Tamas. A performance comparison of KVM, Docker and the IncludeOS Unikernel. MS thesis. 2016.
[12] Enberg, Pekka. "A Performance Evaluation of Hypervisor, Unikernel, and Container Network I/O Virtualization." (2016).
[13] Plauth, Max, Lena Feinbube, and Andreas Polze. "A performance evaluation of lightweight approaches to virtualization." Cloud Computing 2017 (2017): 14.
[14] Wu, Song, et al. "Android Unikernel: Gearing mobile code offloading towards edge computing." Future Generation Computer Systems 86 (2018): 694-703.
[15] Compasti, Maxime, et al. "Unikernel-based approach for software-defined security in cloud infrastructures." NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2018.
[16] https://www.ibm.com/developerworks/aix/library/au-aixhpvirtualization/
[17] https://wiki.archlinux.org/index.php/KVM
[18] https://www.docker.com
[19] https://blog.docker.com/2017/12/top-5-blogs-2017-linuxkit-toolkit-building-secure-lean-portable-linux-subsystems/
[20] https://www.phoronix.com/misc/linux-eoy2018/index.html
[21] Sfyrakis, Ioannis, and Thomas Gro. "UniGuard: Protecting Unikernels using Intel SGX." 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2018.
[22] Madhavapeddy, Anil, et al. "Unikernels: Library operating systems for the cloud." Acm Sigplan Notices 48.4 (2013): 461-472.
[23] Williams, Dan, and Ricardo Koller. "Unikernel monitors: extending minimalism outside of the box." 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16). 2016.
[24] Madhavapeddy, Anil, et al. "Jitsu: Just-in-time summoning of unikernels." 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). 2015.
[25] https://mjbright.github.io/Talks/2018-Jan-28_Devconf.cz_Unikernels/2018-Jan-28_Devconf.cz_Unikernels.pdf
[26] https://github.com/mirage/mirage
[27] http://unikernel.org/projects/
[28] Zhao, Jianxin, et al. "Data analytics service composition and deployment on edge devices." Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks. ACM, 2018.
[29] Cozzolino, Vittorio, Aaron Yi Ding, and Jrg Ott. "Fades: Fine-grained edge offloading with unikernels." Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems. ACM, 2017.
[30] Imada, Takayuki. "MirageOS Unikernel with Network Acceleration for IoT Cloud Environments." Proceedings of the 2018 2nd International Conference on Cloud and Big Data Computing. ACM, 2018.
[31] Kantee, Antti. "Flexible operating system internals: the design and implementation of the anykernel and rump kernels." (2012).
[32] Pavlicek, Russell C. Unikernels: Beyond Containers to the Next Generation of Cloud. O'Reilly Media, 2017.
[33] https://github.com/rumpkernel/rumprun
[34] Elphinstone, Kevin, et al. "A performance evaluation of rump kernels as a multi-server os building block on sel4." Proceedings of the 8th Asia-Pacific Workshop on Systems. ACM, 2017.
[35] Bai, Yongshu, Pengzhan Hao, and Yifan Zhang. "A case for web service bandwidth reduction on mobile devices with edge-hosted personal services." IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018.
[36] Kivity, Avi, et al. "OSvoptimizing the operating system for virtual machines." 2014 USENIX Annual Technical Conference (USENIXATC 14). 2014.
[37] https://github.com/cloudius-systems/osv-apps
[38] Knob, Luis Augusto Dias, Bruno Gomes Xavier, and Tiago Ferreto. "An Unikernels Provisioning Architecture for OpenStack." 2018 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2018.
[39] Bratterud, Alfred, et al. "IncludeOS: A minimal, resource efficient unikernel for cloud services." 2015 ieee 7th international conference on cloud computing technology and science (cloudcom). IEEE, 2015.
[40] http://www.includeos.com/mothership/
[41] https://github.com/solo-io/unik
[42] https://github.com/Solo5/solo5
[43] https://developer.ibm.com/open/projects/solo5-unikernel/
[44] https://github.com/Solo5/solo5/blob/master/docs/architecture.md
[45] https://github.com/akopytov/sysbench
[46] https://github.com/wg/wrk
[47] https://www.nginx.com/
[48] https://mirage.io/wiki/install
[49] https://opam.ocaml.org/
[50] http://www.ocaml.org/
[51] https://github.com/mirage/mirage-skeleton
[52] https://github.com/hioa-cs/IncludeOS
[53] https://github.com/rumpkernel/rumprun-packages
[54] https://maven.apache.org/what-is-maven.html
[55] https://github.com/cloudius-systems/capstan