2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

Time provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform

Bruno Xavier
Pontifícia Universidade Católica
do Rio Grande do Sul (PUCRS)
Email: bruno.xavier@acad.pucrs.br

Tiago Ferreto
Pontifícia Universidade Católica
do Rio Grande do Sul (PUCRS)
Email: tiago.ferreto@pucrs.br

Luis Jersak
Pontifícia Universidade Católica
do Rio Grande do Sul (PUCRS)
Email: luis.jersak@acad.pucrs.br

***Abstract*—Unikernels are a promising alternative for application deployment in cloud platforms. They comprise a very small footprint, providing better deployment agility and portability among virtualization platforms. Similar to Linux containers, they are a lightweight alternative for deploying distributed applications based on microservices. However, the comparison of unikernels with other virtualization options regarding the concurrent provisioning of instances, as in the case of microservices-based applications, is still lacking. This paper provides an evaluation of KVM (Virtual Machines), Docker (Containers), and OSv (Unikernel), when provisioning multiple instances concurrently in an OpenStack cloud platform. We confirmed that OSv outperforms the other options and also identified opportunities for optimization.**

I. Introduction

The agility on systems provisioning has been a subject of study with the rise of cloud computing. Applications, once designed in a monolithic way on top of dedicated machines, are now decomposed into smaller, specialized and distributed services. This approach aims to deliver the benefits of elasticity, portability, scalability and isolation faults. Furthermore, the provided facilities of cloud computing have increased the ephemeral property of systems in a way that applications are in constant change, which implies in a systematic deployment of new images throughout virtualized environments.

At the same time, lightweight virtualization systems, such as Linux containers [1] and Unikernels [2], have become alternatives to traditional operating systems running on top of hypervisors [3]. The former, popularized by frameworks like Docker [4], is already widely supported on private and public cloud platforms. The latter is a recent architecture model that still requires more studies. Unikernels are self-contained instances in which kernel and application are compiled together in the same address space. By compiling only the abstractions needed for the application purpose, Unikernels remove unnecessary components resulting in smaller images with faster boot time.

In this work, we evaluated the provisioning time using OSv [5](Unikernel), Docker (Container), and KVM (Virtual Machine) on top of an OpenStack cloud platform [6]. OpenStack is currently one of the most popular platforms used for both private and public clouds. Through the utilization of the Rally benchmark tool [7] and OSProfiler library [8], we conducted the spawning of several instances to investigate the impact on provisioning time under parallel conditions. We choose the OSv project as the Unikernel representative, since it supports different hypervisors and language runtimes, enabling a fairer comparison to Docker and KVM.

Study developed by the Research Group of the PDTI 01/2015, financed by Dell Computers of Brazil Ltd. with resources of Brazilian Law 8.248/91.

II. Background

This section presents an overview on the virtualization platforms used in our evaluation: KVM, Docker and OSv.

A. KVM (Kernel-based Virtual Machine)

KVM (Kernel-based Virtual Machine) is an open source solution that converts Linux in a Type 1 hypervisor. Due to the utilization of full virtualization and hardware assist, KVM enables running VMs with unmodified guest operating systems. KVM is implemented as a loadable kernel module, reducing the hypervisor size significantly through the reutilization of a large extent of Linux kernel facilities. In KVM, every VM is implemented as a regular Linux process.

B. Docker

Docker is an open-source platform for the deployment and management of Linux Containers. In Docker, containers are built on top of images decoupled in a multi-layer filesystem model. This architecture consists on multiple levels, one upon another in a design that resembles a version control system. It uses a technique called copy-on-write [9]. From the point of view of a container, all layers are unified in a single vision. However, the write operation is allowed only on the top layer.

C. OSv

OSv is an open-source unikernel framework, designed specially for deployment flexibility. It supports different hypervisors, such as KVM, Xen, VMWare and VirtualBox, as opposed to MirageOS and ClickOS, which only support the Xen hypervisor. OSv is compatible to most pre-compiled codes of the Linux OS, due to the utilization of a dynamic linker to translate POSIX calls. It also provides execution environments for different languages, such as Java, C and Ruby, which accelerates application development.

978-1-5090-2453-7/16 $31.00 © 2016 IEEE
DOI 10.1109/CCGrid.2016.86

D. Related Work

Several studies investigate the provisioning time on cloud computing. Most of them, with comparisons between private platforms or public providers. Moreover, some studies propose optimizations throughout the provisioning workflow.

In [10], the authors provide a comparison between OpenStack and OpenNebula. An evaluation is done considering provision of ephemeral instances, provisioning with additional volumes, and provisioning without caching inside the compute nodes. The study decoupled the provisioning time in the following steps: request time, schedule time, platform communication overhead and instance creation time.

Another evaluation [11] does a comparison among three public cloud providers considering different resource configurations of virtual machines: image size and instance type. In addition, the work investigated other conditions such as: provisioning in a particular time of the day and in different data center locations. Since the research was made on cloud providers, the hypervisor time was abstracted.

In [12], among several performance aspects, the provisioning time is measured across distinct open source platforms (Eucalyptus and OpenNebula), along with Amazon EC2. In this case, Xen was chosen as the hypervisor for the open-source platforms. The benchmarks were based on a workload spread in a set of services on top of virtual machines to simulate the Wikipedia service.

In [13], an experiment is done with the simultaneous provisioning of 512 virtual machines on top of OpenNebula. The study was focused on the optimization of OpenNebula in terms of provisioning time.

III. Evaluation

In OpenStack, an instance is considered ready when it reaches a running state. This status differs for Docker and regular virtual machines. In Docker it represents a container in execution, and in KVM, it is a virtual machine ready to launch the operating system. Therefore, we chose to decompose the provisioning time in the following stages:

1) **Instance and Operating System startup**: Regarding OSv/KVM and Linux/KVM, this stage is separated in virtual machine and operating system startup. As for Docker, it is the time for the container process to get started;
2) **Image creation**: Comprises the copy of the image from the Glance repository and its preparation inside the compute node. For OSv/KVM and Linux/KVM, the image is converted to the *RAW* format, resized to the desired flavor and converted again to the QCOW2 format, which represents the final disk image. Docker has to import the image and generate the metadata based on the backend layered filesystem model;
3) **Openstack Overhead**: Considers the overhead generated by the platform with internal communication.

We chose the Rally Benchmark tool [7] to spawn the instances. In order to measure the individual steps, internal calls were traced with the OSProfiler library [8]. Nevertheless, not all OpenStack projects have included the OSProfiler in their current versions, which led us to write and apply patches inside Nova and Neutron projects. The results were grouped in three scenarios, which denotes the concurrent provisioning of 10, 20 and 30 instances. In this way, we could analyse the impact of the concurrency growth on a particular stage.

Our test environment is formed by two nodes. One as the controller and the other one as a compute node. The machines configuration is described in Table 1. A gigabit switch interconnects both machines, providing a theorical throughput of 1Gbps. On both nodes, we have installed OpenStack version Kilo 2015.1.1 on top of a Linux Ubuntu 14.04. Docker (version 1.7.1) and the nova-docker driver were also installed on the compute node. We have left OpenStack in the default configuration except for the parameter *max_concurrent_builds*. It was necessary to increase it to 30, since OpenStack originally limits the concurrent provisioning for 10 instances.

Role	CPU	Memory	Virtual Cores
Controller node	Intel(R) Xeon(R) CPU E5-2650 2.00GHz	64GB	32
Compute node	Intel(R) Xeon(R) CPU E5-2650 2.60GHz	128G	32

Table 1. Machine Specifications

Three images were built and sent to Glance. One in the Docker TAR format and two in the QCOW2 format for both KVM and OSv. We did not consider the variation in image size as a factor for analysis as previous studies have already investigated it [11], [12]. The images properties are shown in Table 2. Regarding the available resources on the compute node and the size of the images, a flavor was added to support the simultaneous provisioning up to 30 instances: 512MB of memory, 4GB for the disk and 1 vCPU.

Virtualization	Format	Image Size
KVM	QCOW2	1.2G
Docker	TAR	342M
OSv	QCOW2	88M

Table 2. Images Properties

The first evaluation comprises the operating system/container startup and the instance creation times. As mentioned earlier, Docker does not have the latter, since the spawning is already the API call to start the container process. Figure 1 shows the results. Concerning the Operating System/Container startup times, OSv/KVM and Docker presented almost the same means for all scenarios, below one second. Linux/KVM, on the other hand, showed a notable growth, increasing by 92% and 48% with 20 and 30 instances respectively. Regarding the variation, both Docker and Linux/KVM increased the distance between the first and the last instance/container (Figure 2). The internal synchronism inside the Docker Engine implied in the serialisation of the load. Therefore, the addition of concurrency will always increase the overall time, albeit this fact should only be relevant to a larger number of containers. There is no difference in the instance creation times for Linux/KVM and OSv/KVM since the image has the same format and the size had no influence in the scenarios.

As mentioned before, the image creation step differs for Docker and KVM (QCOW2) images. Concerning KVM, none of the image measures was impacted by the concurrency, yet the times are presented in Table 3. The copy from Glance repository is the only factor which will impact in this stage, depending on the image size. On the other hand, containers showed a degradation. Despite the image copy from Glance, which did not change, the parallel pressure on the same image increased the times, presenting a variation between the first and the last started container. This behaviour is explained by the decompression of the images to check its metadata before the import, causing a serialisation managed by the Docker API (Figure 3).

Virtualization	Copy from Glance	Convertion	Resize	Disk Creation
OSv/KVM	1000ms	300ms	400ms	200ms
Linux/KVM	12000ms	300ms	400ms	200ms

Table 3. OSv and KVM Image Times

Figure 4 shows the overhead generated by the platform across the concurrency scenarios. We defined overhead as the operations executed by OpenStack till the spawning begins. It comprises the internal HTTP communication among the services; RPC calls within the same service and database calls. In this sense, the platform has a significant impact on the overall provisioning time. Despite the aspects that make either Docker as OSv faster than regular virtual machines, they still depend on the internal cloud mechanisms to get the instances/containers ready. Like the instance creation stage, both OSv/KVM and Linux/KVM show the same means, since they rely on the same provisioning *driver*. Docker presented less overhead, although both drivers were impacted by the concurrency, increasing the times from 12 to 25 seconds, and from 7 to 18 seconds, for KVM and Docker respectively.

Figure 5 depicts the overall provisioning times, covering the request till the total workload of instances/containers become ready in OpenStack. This stage does not include the image copy from the Glance repository, which is strictly related to the image size. Moreover, the operating system overhead was suppressed, that can strongly vary concerning Linux/KVM instances and somewhat with Docker containers, depending on the concurrency load. OSv/KVM, on the other hand, as presented in the first evaluation, has no variation in the operating system startup, therefore, outperforming Linux/KVM instances. Concerning Docker, the provisioning is mainly impacted throughout the image import.

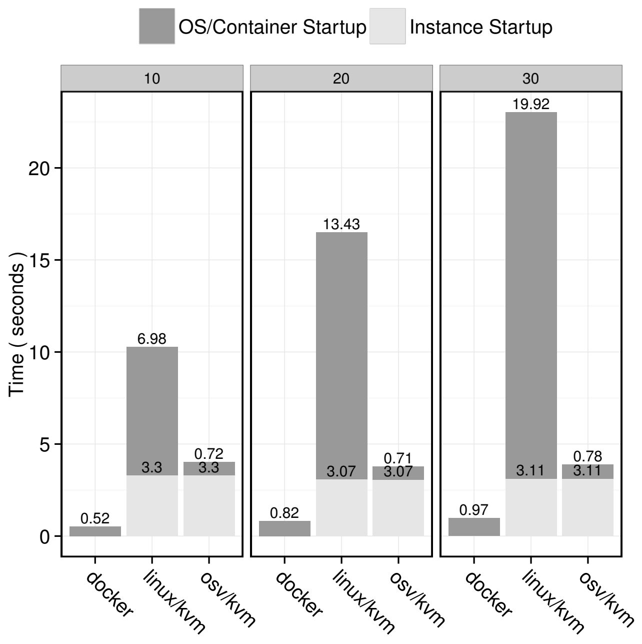
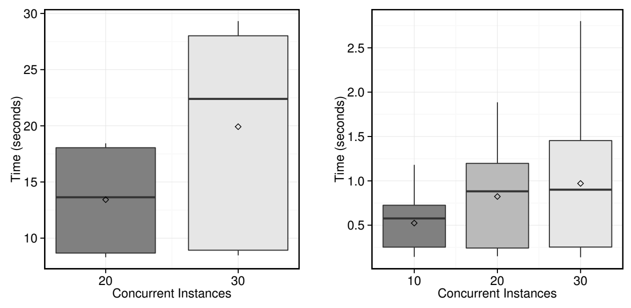


Figure 1: Instance and Operating System/Container Startup for 10, 20 and 30 instances



(a) Linux/KVM OS startup variation (b) Container startup variation

Figure 2: Boot time variation for Linux/KVM and Docker

IV. Conclusions and Future Work

In this work we have conducted an experiment using Docker, Linux/KVM and OSv/KVM to evaluate the provisioning time within these virtualization systems on an OpenStack cloud platform. In this context, we have decomposed each stage of a conventional provisioning pipeline to identify the impact of each step on the technologies. In our results, OSv outperforms Docker and Linux/KVM in the circumstances we propose. It involves the initial provisioning of instances in an environment with absence of caching. In a future work, we intend to extend the study to different conditions. As Docker has a multi-layered file system, it allows the incremental deployment in case the compute node already cached its

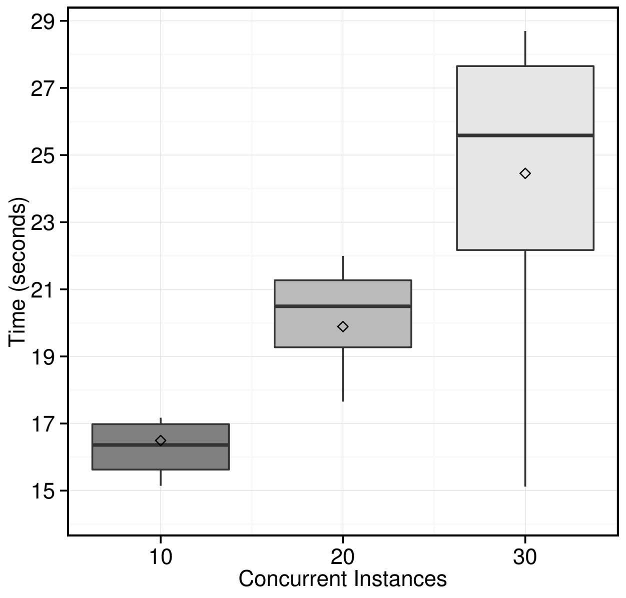


Figure 3: Docker import time variation

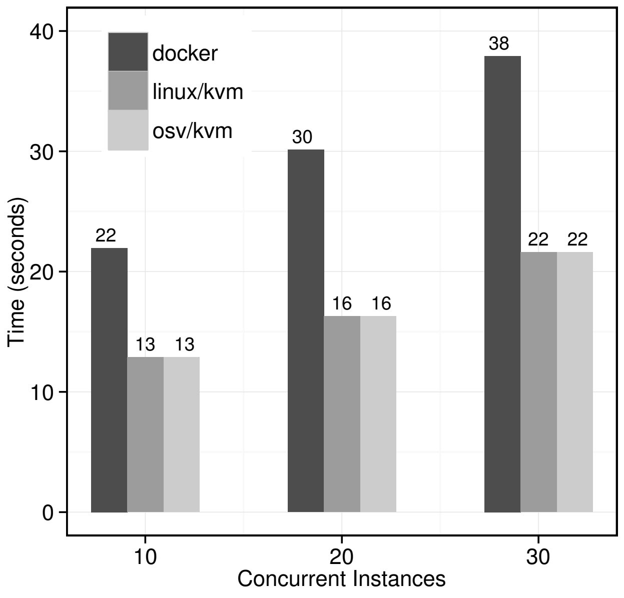


Figure 5: OpenStack full workload times

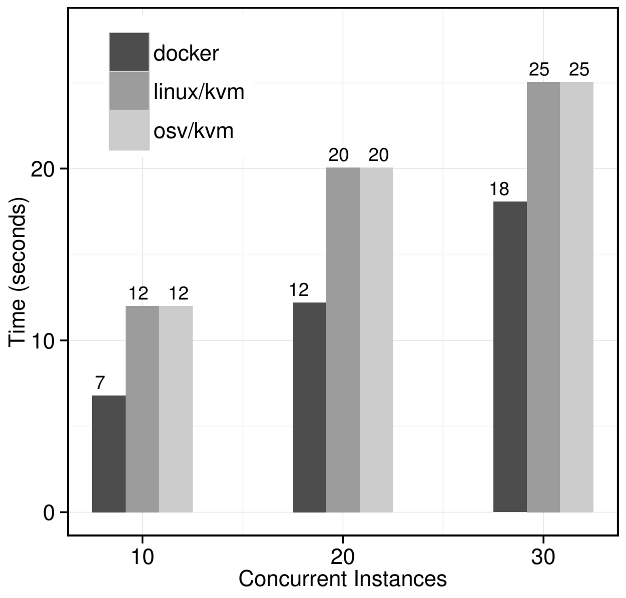


Figure 4: OpenStack communication overhead

underlying layers. On the other hand, OSv has to be fully recompiled and re-provisioned. Hence, a new experiment may be done with the provisioning of just the changed layers concerning Docker. The work was also based on the same image spawned by multiple instances. In this sense, a further investigation will be done with a full stack application.

Despite the focus on the virtualization types, we observed a significant impact caused by the cloud platform in the overhead of the overall provisioning time. Therefore, this work may contribute to future studies considering different IaaS environments and optimizations in OpenStack to improve the deployment time of lightweight virtualization systems.

References

[1] LXC. Acessed on: 20/03/2015. [Online]. Available: https://linuxcontainers.org

[2] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, *Unikernels: library operating systems for the cloud*. ACM, May 2013, vol. 41.

[3] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974.

[4] What is Docker? Acessed on: 20/03/2015. [Online]. Available: https://www.docker.com/whatisdocker

[5] A. Kivity, D. Laor, G. Costa, P. Enberg, and N. Har'El, "OSv—Optimizing the Operating System for Virtual Machines," *2014 USENIX Annual ...*, 2014.

[6] OpenStack. Acessed on: 20/03/2015. [Online]. Available: https://www.openstack.org

[7] OpenStack Rally. Acessed on: 20/07/2015. [Online]. Available: https://wiki.openstack.org/wiki/Rally

[8] OSProfiler. Acessed on: 20/07/2015. [Online]. Available: https://github.com/stackforge/osprofiler

[9] S. Kasampalis, "Copy on write based file systems performance analysis and implementation."

[10] G. Carrozza, L. Battaglia, V. Manetti, A. Marotta, R. Canonico, and S. Avallone, "On the Evaluation of VM Provisioning Time in Cloud Platforms for Mission-Critical Infrastructures," *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pp. 802–810, 2014.

[11] M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 423–430.

[12] Y. Ueda and T. Nakatani, "Performance variations of two open-source cloud platforms," *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pp. 1–10, 2010.

[13] K. Razavi, S. Costache, A. Gardiman, K. Verstoep, and T. Kielmann, "Scaling VM Deployment in an Open Source Cloud Stack," in *ScienceCloud '15: Proceedings of the 6th Workshop on Scientific Cloud Computing*. ACM Request Permissions, Jun. 2015, pp. 3–10.