Virtualization and Containerization of Application Infrastructure: A Comparison

Mathijs Jeroen Scheepers
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.j.scheepers@student.utwente.nl

ABSTRACT

Modern cloud infrastructure uses virtualization to isolate applications, optimize the utilization of hardware resources and provide operational flexibility. However, conventional virtualization comes at the cost of resource overhead. Container-based virtualization could be an alternative as it potentially reduces overhead and thus improves the utilization of datacenters. This paper presents the results of a marco-benchmark performance comparison between the two implementations of these technologies, namely Xen and LXC, as well as a discussion on their operational flexibility.

Keywords

Hypervisor, Virtualization, Cloud computing, Application infrastructure, LXC, Xen, Container-based virtualization

1. INTRODUCTION

According to Zhang et al. [20] virtualization technology is an essential part of modern cloud infrastructure, such as Amazon's Elastic Compute Cloud (EC2) and Google's App Engine. These days, most cloud computing datacenters run hypervisors on top of their physical machines. A hypervisor is a piece of computer software that creates and runs virtual machines. With these hypervisors, and the virtual machines that run on them, system administrators are able to optimize the use of available physical resources and confine individual parts of application infrastructure. A typical setup is displayed schematically in Figure 1. With the use of virtualization, resources can be consumed more effectively than conventional bare-metal setups, which use physical machines for isolating different parts of application infrastructure. Still efficiency could be increased even further. A hypervisor will run multiple kernels on a single physical machine, therefore the isolation of applications and processes is expensive. Mills [14] stated that 1,500 terawatt-hours of power per year is used to power cloud computing datacenters, that is about 10% of the worlds energy consumption, and this number is climbing. If compute resources could be used more efficiently that could have a big impact.

The cloud computing paradigm, as described by Buyya et al. [7], expects hypervisors to provide isolation and portability. The Xen [4] hypervisor is a popular technology and widely used at the moment.

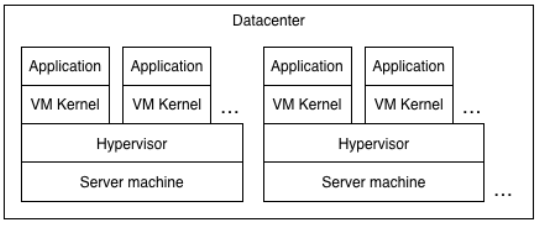

Figure 1. A schematic overview of virtual machines in a datacenter.

With recent developments around Docker [2] and LXC [3] there now seems to be a viable alternative to the hypervisor and traditional virtualization for application infrastructures. Linux Containers (LXC) is a kernel technology that is able to run a multitude of processes, each in their own isolated environment. This technique is called *container-based virtualization*. Docker is a tool that makes it easy to package an application and all of its dependencies into such containers. Merkel [13] explains that "Docker is ... the lightweight and nimble cousin of virtual machines".

There is a school of thought, popular within the Linux community, that claims that hypervisors originally were developed due to the Linux kernel's inability to provide superior resource isolation and effective scalability [11]. The container could be the solution.

The multiple kernels running on a hypervisor use a rather large fraction of the machines physical resources. LXC does not seem to have this problem. Combined with the tooling Docker provides, they provide the flexibility a modern system administrator expects, like easy provisioning and image construction. The way LXC isolates processes could reduce overhead on major software deployments in deployment time, application portability as well as physical resource usage. With a kernel feature, LXC is able to isolate processes and allocate resources without the use of hardware emulation. The technology is leveraged by the Docker and CoreOS [1] software, which enables the creation of complex and portable application infrastructures. Where Docker provides LXC with the deployment tooling it needs, CoreOS provides the underlying host operating system and makes it possible to setup a cluster of machines on which containers can be managed and migrated.

By using a single kernel per bare-metal machine, container-based virtualization could shift the cloud paradigm away from hypervisor-based virtual machines.

21st Twente Student Conference on IT June 23rd, 2014, Enschede, The Netherlands.

Table 1. Virtualization technologies

Full	Para	Container-based
KVM	Xen	LXC
	VMWare	OpenVZ
	VirtualBox	VServer
	UML	

LXC differs in a lot of ways from the traditional hypervisor. This paper will focus on two differences: physical resource impact and operational flexibility. This paper is structured as follows:

In Section 2 we will elaborate on the working of the Xen hypervisor, LXC, Docker and CoreOS. Next, in Section 3 we discuss related work and the contribution of this paper.

We will compare the physical performance of a single machine, running the same application using two different isolation techniques. Namely, isolation through virtualization and through containerization. Do containers really have a performance benefit, and if so, how significant is that benefit? These questions will be answered in Section 4 by analyzing the results of several benchmarks.

There are still several research challenges when talking about cloud computing. Among others, improving automated service provisioning, machine migrations and server consolidation [20]. In Section 5, we discuss these and show how Xen and Docker are able to help with these challenges and how their solutions differ.

Finally, in Section 6 the results of the performance comparison as well as the operational comparison will be discussed and related.

2. BACKGROUND

Modern application infrastructure techniques and methodologies incentivize an accelerated adoption of cloud computing technologies as well as various virtualization technologies. For example the DevOps [10] software development methodology and techniques that require scriptable infrastructure.

The virtualization technologies that have emerged mostly focus on the Linux kernel and can be split up into three categories: full-virtualization, para-virtualization and container-based virtualization. Para-virtualization modifies the kernel of virtual machines slightly to optimize for performance in the virtual environment. Full-virtualization does not require kernel adjustments. Container-based virtualization does not use a kernel at all.

Table 1 shows a selection of various technologies and their categorization. These are the kind of technologies used in IaaS (infrastructure as a service) solutions and PaaS (platform as a service) solutions like Amazone Elastic Compute Cloud, Google App Engine, DotCloud and Open Shift. Since application infrastructure can be diverse, there is no single best solution for all of these services. Rather, each service or application has its own specific requirements.

We will be comparing two different technologies with a different architectures: Xen, a para-virtualization hypervisor and LXC, a container-based isolation linux kernel feature. In this section we will briefly explain both architectures. In the following sections we will go in depth on their performance and operational flexibility.

2.1 Xen

The Xen hypervisor is based on para-virtualization. A

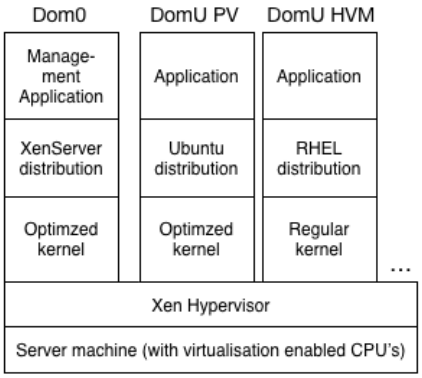

Figure 2. A schematic overview of a machine running the Xen hypervisor.

virtual machine on the Xen hypervisor could run a modified kernel in order to provide better performance and reduce overhead. The hypervisor is installed directly into the bootloader.

The virtual machines, running on top of the hypervisor, are called domains or guests. A special domain, called *domain0*, controls the system (Dom0). This domain has the capability to setup the environment. It could contain tools for the setup of networking, provisioning of new virtual machines and migrating them.

The other domains are what is called underprivileged to domain0. Therefore they are called *DomU*. These DomU domains can either be para-virtualized (PV) or hardware-assisted (HVM). The PV-domains require a optimized kernel, whereas the HVM-domains require no kernel modification but do require x86's virtualization support (Intel VT-X, AMD-SVM). This architecture support is not required when running a PV virtual machine.

Since Xen only provides the hypervisor technology, we still need a management operating system to be installed on Dom0. The XenServer is an implementation for the Dom0 management system. It provides extended tooling to provision, manage, monitor and migrate virtual machines. This is the domain on which XenServer could be installed. With domain0 being a virtual environment, a XenServer installation is itself running on a virtual machine. Figure 2 shows a schematic overview of a machine running the Xen hypervisor with XenServer installed on Dom0.

Xen has been in development for more than 12 years and thus can be considered a mature technology. Xen technology is widely used, for example by Amazone Web Services, Google, Rackspace, Oracle, Cisco and Citrix [5].

2.2 LXC

Linux Containers (LXC) provides lightweight operating system virtualization and is relatively new to the other technologies listed in Table 1. Unlike Xen, LXC does not require hardware architecture support. LXC is the successor of VServer and OpenVZ, other container-based virtualization technologies.

The basic principle of a container is that it allows for processes and their resources to be isolated without any hardware emulation or hardware requirements. Containers provide a sort of virtualization platform where every container can run their own operating system but share the kernel. So each container has their own filesystem and

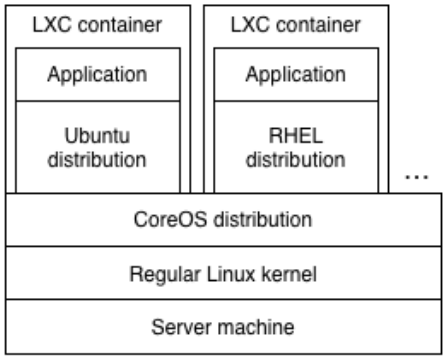

Figure 3. A schematic overview of a machine running the CoreOS and LXC containers.

network stack, and every container can run its own Linux distribution. For example, a CoreOS host can run Ubuntu, RHEL, Debian, Arch and even other CoreOS containers simultaneously. These abstractions make a container behave like virtual machine with a separate filesystem, networking and other operating system resources. But really they are not, since there is no hardware emulation taking place. Figure 3 shows a schematic overview of a machine running the CoreOS and two LXC containers.

Isolation is an important aspect of containers, it is provided through Linux *cgroups* and *namespaces*. Namespaces are used to isolate resources like; the filesystem, networking, user management and process ids. Cgroups are used for resource allocation and management. For example with a cgroup the amount of memory, a container can use, can be limited. Cgroups are regular Linux process groups so they can run next to any host OS processes. One important difference in resource allocation between LXC and hypervisors is that CPU resources can not be allocated on a per core basis, rather one should specify a priority.

2.2.1 Docker

Docker is a tool that makes it easy to package an application and all of its dependencies into a container. It does this by providing a toolset and an unified API for managing kernel-level technologies, such as LXC containers, cgroups and a copy-on-write filesystem.

Docker relies on *AuFS* (Advanced Multi-Layered Unification Filesystem) as a filesystem for containers. AuFS is a layered filesystem that can transparently overlay one or more existing filesystems. AuFS allows Docker to use certain images as the basis for containers. For example, you might have an Ubuntu image that can be used as the basis for many different containers. Thanks to AuFS, only one copy of the Ubuntu image is required, which results in savings in storage space and memory consumption, as well as the faster deployment of containers. Another benefit of using AuFS is the ability to version images. Each new version is simply a *diff*[1] of changes from the previous version, effectively keeping image files to a minimum. This also means that there always is a complete audit trail of what has changed from one version of a container to another, just like version control systems used in software development[13].

2.2.2 CoreOS

CoreOS is a relatively new Linux distribution that has been architected to provide features needed to run large modern infrastructure stacks. The distribution provides a trimmed down Linux kernel to reduce as much overhead as possible.

CoreOS also provides the fleet and etcd tools with which a cluster could be setup to provide redundancy and failover.

2.2.3 Future

Both Docker and CoreOS are still in active development and have not reached a stable release. Both projects currently recommend *not* to use the systems in production environments.

LXC already has a stable release but new features are still in development. There are plans to add new namespaces to enhance isolation and security. These include a security namespaces, device namespaces and time namespaces. Especially the time namespace is an interesting development, since it will allow for live host migrations.

Combining LXC with Docker and CoreOS, the whole package provides a lightweight, clean, full featured base layer for isolating application infrastructure.

3. RELATED WORK

Virtualization was first introduced in the 1960s by researchers at IBM. The system, IBM developed, has evolved and is currently still being used in their z/VM hypervisor for the IBM System Z mainframe.

Ever since IBM first introduced the idea, virtualization has been a well-covered research topic, especially virtualization for the x86 architecture. There have been several papers comparing various virtualization technologies:

Quétier et al. [15] compared VServer container technology with Xen, UML and VMWare on their ability to scale and provide resource isolation. They found that VMWare and UML have strong limitations with respect to overhead and performance isolation. They also found that Xen suffers from slow inter-virtual machine communication performance.

Che et al. [8] compared micro- and macro-performance of the Xen, KVM and OpenVZ technology. OpenVZ is a predecessor of LXC and is also based around container-based isolation.

Wang and Ng [17] presented a measurement study to characterize the impact of virtualization on the networking performance of the Amazon Elastic Cloud Computing, which uses Xen. It was found that even when the network was lightly used, virtualization could introduce significant delay variation and throughput instability.

Bardac et al. [6] used LXC to deploy a large scale peer-to-peer BitTorrent network. Host resource analysis and swarm performance analysis were performed for multiple swarm configurations. The experiment allowed the identification of several correlations between virtualization parameters, such as the influence of uplink traffic shaping on download capacity and the relation between host switching capacity and CPU utilization.

Younge et al. [19] evaluated virtualization technologies in the context of HPC (High performance computing): Xen, KVM and VirtualBox were compared. It was concluded that KVM is the best overall choice for HPC since the researchers found that KVM performed significantly better than Xen and VirtualBox in the HPCC[12] and SPEC[9] benchmarks.

[1] `Diff` is a file comparison utility that outputs the differences between two files

Xavier et al. [18] did similar research on virtualization in HPC environments. However, he focused solely on container-based technologies: LXC, OpenVZ and VServer. The study found that the resource isolation features in container-based systems are not mature, yet. Performance on memory and network isolation was poor, CPU on the other hand was isolated well. Disk I/O performance was not measured.

Sampathkumar [16] did a comprehensive study where he compared LXC with Xen as well as KVM. This was done in order to find the optimal technology to be used in the *Intelligent River middleware system*. He did several micro-benchmarks measuring performance in CPU, memory and disk I/O. However, networking was outside of the scope of his research, as was the use of deployment and management software. All the software was installed with Ubuntu as the host operating system. One could argue that this does not do the specific technologies justice, since every technology runs better on a Linux distribution tailored to its needs. The research found that Xen was far better on isolating resources but at the cost of adding overhead. LXC, on the other hand, was more performant when looking at disk I/O, RAM as well as CPU. In the end, the advice was in favor of LXC technology.

3.1 Contribution

Most of these studies draw their conclusions from micro-benchmarks and focus on the core of the virtualization technologies—like the hypervisor. Micro-benchmarks are benchmarks which focus on a single isolated component. For example the time the PHP interpreter takes to execute a certain algorithm. Macro-benchmarks on the other hand are benchmarks that focus on interconnected components, like an application's infrastructure as a whole.

The real difference in virtualization technologies can often be found in the way virtual machines communicate with one another, and how load on a specific virtual machine influences the other. The performance benchmarks described in this paper are macro-benchmarks. We look at network latency when virtual machines are communicating with one another and take the deployment infrastructure into account.

Virtual machines do not live on their own, they live within application infrastructure and perform their designated task. This infrastructure could built on a cluster of physical machines for example all running CoreOS or XenServer. These low level software implementations are the core on which a cloud computing datacenter is built. The development of Docker has made the use of LXC considerably easier.

We will measure the performance of both technologies running on software tailored to their needs and we will discuss operational flexibility, which is essential to system administrators. This is different from previous work since we look at the application infrastructure as a whole and take tooling into account. This instead of performing micro-benchmarks, which are performed on host operating systems which are not used in real production environments, since they are not tailored to the specific technology.

4. PERFORMANCE COMPARISON

In order to compare Xen with LXC we have setup XenServer 6.2 and CoreOS 324.3.0 with Docker 0.11.1 on two identical machines. They are equipped with 4GB of RAM and an Intel Xeon Quad core CPU with Intel VT-X virtualization support. Both will run two Ubuntu 12.04 virtual machines or containers.

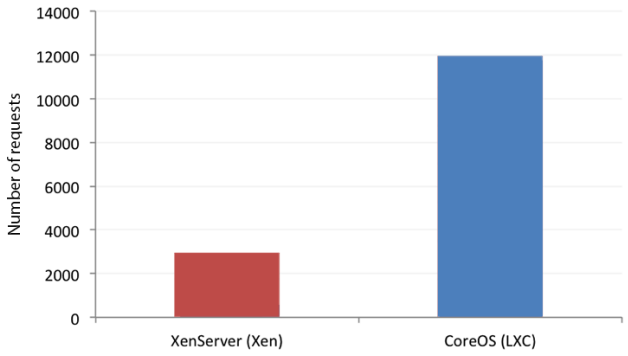

Figure 4. The total amount of requests processed within 800 seconds. (More is better)

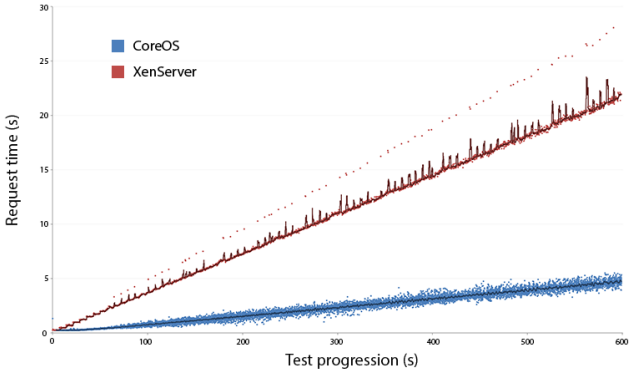

Figure 5. Progression of the request time for the first 600 seconds. (Less is better)

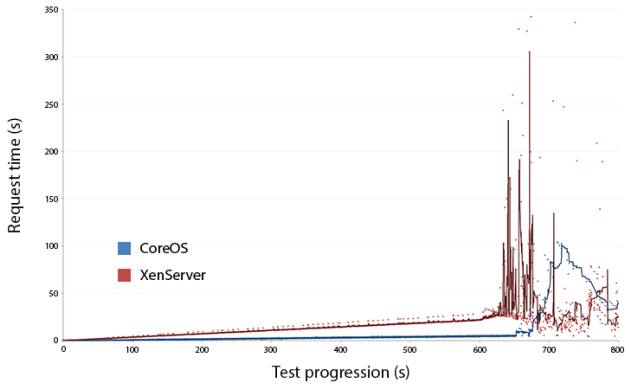

Figure 6. Progression of the request time for 800 seconds. (Less is better)

The virtual machines running on the XenServer host use a para-virtualized kernel and have XenServer Tools installed for further optimization through drivers especially designed for running on Xen. On Xen both virtual machines get access to two CPU cores.

The first virtual machine gets access to 2 GB of memory and runs Apache 2.2, PHP 5.3 and WordPress 3.9. It functions as an application server. The second virtual machine gets access to 1 GB of memory and runs MySQL 5.5 with a database filled with the default sample content WordPress provides. This machine functions as database server. With these technologies we run an installation of WordPress on the LAMP application stack with separate application and database servers.

Two separate benchmarks have been performed, the first benchmark focused on the application's performance when it is used by an increasing number of users. The second benchmark focused specifically on the interaction between the two machines.

For the first benchmark, we used JMeter to generate a large number of simultaneous requests. Using `top`[2], NewRelic server monitoring and XenServer software the performance on the host system will be monitored.

4.1 Application benchmark

The application test is a macro-benchmark. Our application setup will present a WordPress blog filled with the default sample content provided with the installation. When a request is made to the blog, the WordPress software needs to fetch data from the database and return with a response. We will use JMeter to send an increasing number of simultaneous requests. We use an increasing number to know on with number of simultaneous requests the server runs out of memory and start experiencing severe performance problems.

Within 800 seconds JMeter attempted to perform as many requests as it could, with an increasing amount of concurrent requests. Only requests which resulted in a successful response were counted. At $t = 0ms$ the testing software started with 1 concurrent request and would send a new request once the previous finished. The number of concurrent requests was increased linearly, until at 720 seconds it reached 100 concurrent requests.

One of the purposes of this benchmark was to check when significant performance loss, due to resource shortage, would happen.

Figure 4 shows the number of requests that were successful within 800 seconds. The figure shows that CoreOS was able to process far more requests within the 800 seconds, more than four times as many as Xen. This was unexpected since Sampathkumar [16] showed, in his micro-benchmarks, that LXC outperformed Xen by 7% not by 306%. This difference could be attributed to the different ways CPU isolation is handled, where Xen isolates per CPU core, LXC uses isolation based on cgroup priority. With this LXC could be able to use the available CPU resources more effectively and we think this could be the cause of this difference.

Figure 5 shows that Xen takes more time to process a single request. The trend line drawn in-between the data points takes the average of 30 data points. A flatter line suggests a more consistent performance. So Figure 5 shows that Xen does not perform as consistently as LXC, even

[2] `Top` is an activity monitor utility for Unix, similar to the Windows task manager.

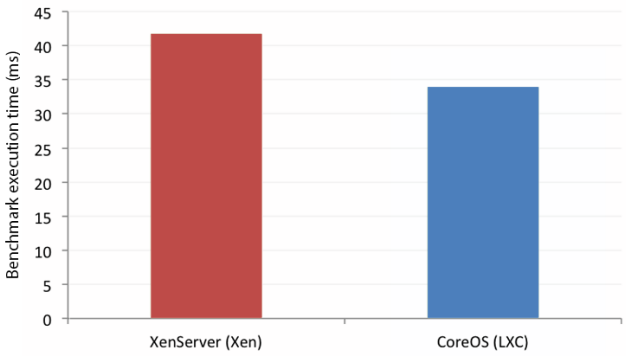

Figure 7. Time in ms to complete a one SQL SELECT query. (Less is better)

when it has physical memory available.

When the number of connections increased beyond what the server could handle with physical memory, the host starts swapping memory between its hard disk and physical memory—Figure 6 clearly shows this effect. After 610 seconds (85 concurrent connections) the Xen setup starts this process. And after 655 seconds (91 concurrent connections) the LXC setup runs out of memory. This means that the memory overhead Xen introduces, in respect to LXC, could be used to process 6 more concurrent requests with LXC.

Furthermore Figure 6 shows that LXC handles the thrashing more consistently. However, Xen is able to continue serving the responses faster and with less failed responses. Failed responses are identified using HTTP status codes, for example `502 Bad Gateway`. The downward slope after 700 seconds is caused by failed requests, since the graph only shows successful requests. After 707 seconds the application running on LXC starts to throw errors. This is expected since Sampathkumar [16] has already shown that Xen is considerably better at isolating than LXC. In particular, in situations where the required resources exceed the available ones.

In this benchmark we did not give the containers access to the surplus of memory, the difference in host operating system footprint provided. The basic footprint of a clean setup for XenServer used 906MB memory for running the domain0 virtual machine. If we compare this to 161MB footprint of a clean CoreOS installation, the difference would be 745MB, which could be used for handling additional requests.

4.2 Inter-virtual machine communication benchmark

In order to test inter-virtual machine communication with a real world application stack we performed two more benchmarks. The first was a PHP-script querying the database to test inter-virtual machine communication. The script was executed on a different virtual machine than the database. The results are shown in Figure 7. These show that LXC experiences less overhead when querying the database. This overhead consists of overhead in networking and CPU utilization, since these are the main resources consumed by running this benchmark. The results corresponds with the conclusion of Wang and Ng [17] which states that the virtual networking used with Xen introduces overhead.

A far more interesting benchmark is the performance of inter-virtual machine communication under stress. With

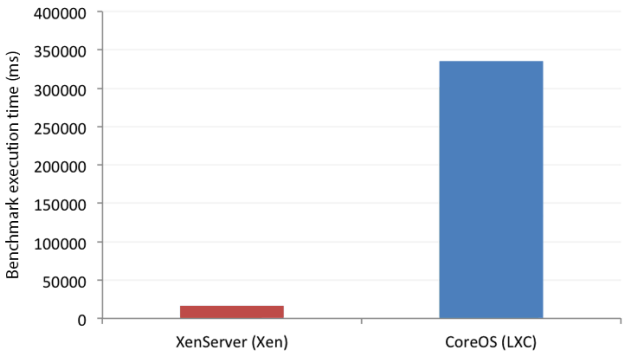

Figure 8. Time in ms to complete 10.000 SQL INSERT queries. (Less is better)

another PHP-script, we will be inserting randomly generated data into the database. The script was set up so that the generation of the data utilized all the available resources. Figure 8 shows that the same script took 16 seconds to complete on the Xen setup. While it took 335 seconds on the LXC setup. This clearly shows LXC's inability to successfully isolate resources.

With the results of these benchmarks we can confirm that the conclusions of Sampathkumar [16] and Wang and Ng [17] still hold in the environments tailored to the specific technologies. However, the difference in resource isolation ability and overhead reduction seem to be more extreme.

5. OPERATIONAL COMPARISON

While optimal performance and stability is a noble goal, we believe that ease of use and performant tooling is equally as important. Especially since the rise of the new DevOps [10] software development methodology, which encourages the use of scripting and automation when designing application infrastructure. The use of virtualization and the development of infrastructure APIs has enabled this.

Both LXC and Xen have various tooling options and could be used to automate deployment and design infrastructure. However, how these tools work differs. Table 2 shows how we have judged various components of the tooling. The table distinguishes between CoreOS, which uses Docker and LXC, XenServer and Amazone's EC2, which also uses Xen and provides virtual machines as a public cloud service. EC2 has other, and in some cases more comprehensive, tooling options than XenServer. We will discuss each component.

Section 4.1 shows that XenServer has a significantly larger memory footprint than CoreOS. We could not determine the footprint on EC2, but one could assume this is similar to XenServer since EC2 relies on the same technological basis.

5.1 Provisioning

Provisioning is the setup of a new machine to make it ready for use. For example, the virtual machine in the WordPress benchmark was provisioned by installing Apache, PHP, WordPress and monitoring software.

Docker provides the building of container-images with Dockerfiles, which, in essence, exists of a base image and a set of commands to be executed. Each executed command creates a new image, so the resulting image is incrementally built by executing a single command on a new base image. Each new image is simply a diff of changes from the previous image, effectively keeping image files to a minimum.

Table 2. Oparational flexibility comparison

	CoreOS	XenServer	EC2
Technology footprint	+	-	N/A
Image creation	+	0	0
Service discovery	+	0	+
Cluster configuration	0	+	+
High availabilty	+	+	+
Startup time	+	0	0
Machine migration	-	+	+

The Dockerfile enables branching of images, since every resulting image can serve as a new base image for a new Dockerfile. For example, you could have an `ubuntu-essentials` image which contains monitoring software. An `ubuntu-application` image and an `ubuntu-database` image could both use the same `ubuntu-essentials` image as base image. This adds flexibility but also enables rapid image creation.

XenServer and IaaS solutions provide snapshots as well as images. But these can not easily be constructed from a single script and duplicate the entire image data. Therefore one could argue that Docker provides a better format for image creation.

5.2 Service discovery

Service discovery is the mechanism used to find and connect to other servers within your application infrastructure. For example, an application server should be able to find the database server and connect to it, without the need for manual configuration. Another example would be, that a load balancer should be able to find all the available application servers. And if a new application server is started it should be discovered by the load balancer and added to its pool.

IaaS solutions that use Xen like EC2 use metadata services, which every virtual machine can access by sending an HTTP request to a specific URL. XenServer itself does not support this out-of-the-box. This does not have to be a problem, since one could architect infrastructure that leverarges service discovery without an out-of-the-box metadata service. However, a metadata service does provide an easier solution.

CoreOS uses a metadata service called *etcd*. The service runs on the host machine and can be clustered. Correct clustering does require some configuration. Our experience is that debugging a wrongly configured cluster is not always convenient. Since CoreOS has not reached a production ready state we cannot say how convenient this will be in the final release.

5.3 High availability and failover mechanisms

A good application architecture should contain failover mechanisms and distribute virtual machines over a number of physical machines, preferably in separate locations. This way, if one physical machine fails, due to for example a hardware error, the application will continue to run.

XenServer provides comprehensive high availability features. A XenServer cluster could be set up on which running virtual machines can be migrated from one host to the other.

CoreOS uses a service, called *fleet*, to set constraints on the manner containers are distributed within a cluster. For

example, if the cluster contains two database containers, mirroring each other to provide redundancy, it is important that these containers are not placed on the same physical machine. With fleet a system administrator is able to set constraints on which containers can run on the same machine and which can not. The service can also used to migrate containers from one physical machine to another. However, the running containers will not be migrated due to current limitations in LXC. Rather the container will be stopped on the old server and a similar container will start on the new server. This means, the memory used in the old container will be lost. This could be an issue for applications that rely on memory, for storing sessions or other ephemeral data. If the application is architected in a way that it can handle the loss of memory, a container migration with fleet should be fairly similar to a virtual machine migration with XenServer.

The time it takes to start up a new server with Xen could take a couple of minutes, where the same process with LXC and Docker takes less than a second. This is important for failover mechanisms that rely on the starting of new virtual machines or containers.

With all of these features both Xen and CoreOS seem well equipped to provide operational flexibility for complex application infrastructures. CoreOS does address some of the problems Xen has, it reduces the overhead in image size and it can start new services within a second. CoreOS does not support the migration of live machines. This could be a drawback depending on the application infrastructure.

6. CONCLUSION

This paper presented the results of a comparison between the Xen and LXC virtualization technologies, using them to run several components of an application. A macro-benchmark has been performed, which was set up using tailored operating systems and with networking between two virtual machines. Since performance is not everything the operational flexibility of the technologies has also been discussed.

Both hypervisor-based and container-based virtualization can provide portability, isolation and optimize the utilization of hardware resources. The technologies are both the right choice for different use cases. If it is important that resources are distributed equally and performance should be less dependent on other tasks, executed on the same system, Xen is a good option. However, if you want to get the most out of your hardware or if you wish to execute a lot of small isolated processes, LXC is a better option, since it introduces less overhead and is more performant.

The CoreOS, Docker, LXC stack has a lot of potential. If the technology could be improved to provide better resource isolation, which seems to be its only major flaw, it could drastically reduce overhead on major server deployments, even in public clouds. This could lead to more efficient use of datacenter resources and thus reduce the electricity needed for running cloud applications, as described by Mills [14].

For now Xen could be a good option on public IaaS clouds. Because the customer can not influence the exact server on which his virtual machine will be running. The customer also has no knowledge on the workload of other virtual machines. Virtual machines, running on Xen, are not fully independent from the resource consumption of other machines on the system. However this is far better than when using LXC.

LXC is currently used at DotCloud in their PaaS environment. Since DotCloud can monitor the workload of several containers on several machines, and since it has control over specific parts of the infrastructure, it can migrate containers which are experiencing performance problems to other machines. So, LXC is a good option for a PaaS environment like DotCloud as well as in private clouds.

When a stable versions of Docker and CoreOS become available and LXC has implemented the new namespaces and improved isolation, it would be interesting to do the same benchmarks again and see how the results differ.

7. REFERENCES

[1] Coreos. URL `http://coreos.com/`. Accessed: 2013-05-30.

[2] Docker. URL `https://www.docker.io/`. Accessed: 2013-05-30.

[3] Lxc, linux conainers. URL `http://linuxcontainers.org/`. Accessed: 2013-05-30.

[4] Xen project, . URL `http://www.xenproject.org/`. Accessed: 2013-05-30.

[5] Xen project membebers, . URL `http://www.xenproject.org/project-members.html`. Accessed: 2013-05-30.

[6] Mircea Bardac, Razvan Deaconescu, and Adina Magda Florea. Scaling peer-to-peer testing using linux containers. In *Roedunet International Conference (RoEduNet), 2010 9th*, pages 287–292, 2010.

[7] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13, 2008.

[8] Jianhua Che, Yong Yu, Congcong Shi, and Weimin Lin. A synthetical performance evaluation of openvz, xen and kvm. In *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, pages 587–594, 2010.

[9] John L Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[10] Michael Hüttermann. *DevOps for Developers*, volume 1. Springer, 2012.

[11] Michael Kerrisk. Lce: The failure of operating systems and how we can fix it, 2012.

[12] Piotr Luszczek and J Dongarra. Hpc challenge benchmark, 2005.

[13] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), 2014. ISSN 1075-3583.

[14] Mark Mills. The cloud begins with coal. Technical report, 2003.

[15] Benjamin Quétier, Vincent Neri, and Franck Cappello. Scalability comparison of four host virtualization tools. *Journal of Grid Computing*, 5(1):83–98, 2007.

[16] Sampathkumar. Virtualizing intelligent river(r): A comparative study of alternative virtualization technologies. Clemson University, 2013.

[17] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, 2010.

[18] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange, and Cesar AF De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240, 2013.

[19] Andrew J Younge, Robert Henschel, James T Brown, Gregor von Laszewski, Judy Qiu, and Geoffrey C Fox. Analysis of virtualization technologies for high performance computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 9–16, 2011.

[20] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.