Virtual machine and Docker container: different approach to Virtualization

Carlo Butelli

VU University Amsterdam,
Computer Science Editorial,
De Boelelaan 1105, 1081 HV Amsterdam,
The Netherlands
c.butelli@student.vu.nl

April 29, 2017

Abstract

This paper is focused on virtualization approaches. The intention is to provide some key answers to the question "What does virtualization need to be beneficial in respect to the use of physical servers? Since physical servers are know to be expensive in costs and even more often effectively under-utilized, how can this wastage be avoided(or at least decreased) to increase the overall profits?". A couple of solutions to build virtualization are presented and the expectations from a developer point of view about those technologies to build a virtualized system are also shown. The first part starts with a brief introduction about cloud computing and a description of "virtualization" meaning, then the article will range over some Virtual Machines(VMs) and Docker descriptions to conclude with evaluations and discussion of both the solutions.

I. Introduction

Nowadays, cloud computing has become the center of demand from many sectors. Instead of building their own IT infrastructure, including hardware and involving the development (and maintenance) of software applications and databases, enterprises have started to move towards the accessibility of computing resources, hosted by third parties, over the Internet. Subsequent to this increase of demands, the cost of cloud infrastructures has progressively increased.[1] **The adoption of cloud technologies has driven to the creation of large scale data centers to provide cloud services which in turn lead also to a massive increase of electricity consumption rising data center property costs and carbon footprints. Most businesses even more often use a combination of several application servers, web servers, document servers, image servers, audio/video servers and the database servers. However, most of the hardware appears to be used by reason of the average number of server requests recorded, meaning that the servers are often broadly under-utilized. A server in general takes only about 1 to 10ms to return a response for each request. Given that, the amount of time a server is maintained up and running is much higher compared with the relative time it takes to service such request. This clearly shows that a significant amount of energy is wasted per server just in the process of keeping the servers up and ready to service requests upon their arrival. Designing energy-efficient data centers has recently received considerable attention. As days passes the importance of new services increases along with the request of more hardware and more effort from IT administrators as well as capacity, storage and networking that are increasing day by day.** ***"Cloud computing focus on what IT always needs: a way to increase capacity on the fly without investing in new infrastructures.*** [2] **So, how exactly do we get rid of this wastage and by that increase the profits? The answer seems to dwell with virtualization.**

II. Cloud computing

Over the years IT complexity has grown while efficiency has plummeted. Virtualization reverses this trend simplifying IT infrestructure so you can do more with less. Virtualization lets run your application on pure physical servers. Most of the time, for non-IT folks, confusion occurs because virtualization and cloud computing work together to provide different types of services. In reality *Virtualization* appears in the picture of cloud computing as the software used to manipulate hardware and to divide physical infrastructures to make various dedicated virtual resources. While cloud computing is the actual service result and describes the delivery of shared computing resources. CPU, memory, hard drive space can be emulated in software, having this pretended environment, where all of this hardware seems to be, allowing to reduce IT costs increasing efficiency, utilization and agility. [3]

In [2] benefits of Cloud computing have been exposed:

- **Flexibility**, IT costs can be adjusted to meet organizationâĂŹs needs.
- **Security**, since it has been studied that data in the Cloud are much more secure than in classic server room.
- **Capacity**, that can always be increased.
- **Cost**, since Cloud and Virtualization can substantially reduce maintenance fees(in general a subscription fee is used to cover all the previous costs).

The Cloud Computing model comprehends two main models, deployment and delivery.[4] The cloud platform deployment models are exposed below:

Private cloud dedicated for specific organization.

Public cloud intended for public users to register and use the available infrastructure. Those represent the the most vulnerable among the deployment models because they are available for public users to host their services who may be malicious users.

Hybrid cloud is a private cloud that can extend to use resources in public clouds.

The cloud service delivery models instead consist of:

Infrastructure-as-a-service (also referred to as IaaS): this service model is based on the virtualization technology providing virtualized computed resources, storage and network over the Internet and abstracting the user from the details of infrastructure like physical computing resources, location, data partitioning, scaling and so on. It offers highly scalable resources that can be adjusted on-demand. Amazon EC2 is a good example to express this model.

Platform-as-a-service (also referred to as PaaS): a cloud provider delivers hardware and software tools needed for application development allowing the customer to develop, deploy and manage their own applications, without installing anything on their local machines. This model may also be hosted on top of IaaS model or on top of the cloud infrastructures directly. Google Apps and Microsoft Windows Azure are the most known PaaS.

Software-as-a-service (also referred to as SaaS): a cloud provider hosts applications and makes them available to customers over the Internet removing the need for organizations to install and run applications on their own computers or in their own data centers. This solution also allow to avoid most of the expenses hardware related like acquisition, provisioning and maintenance, as well as software licensing, installation and support. This model may be hosted on top of PaaS, IaaS or directly on cloud infrastructure. SalesForce CRM is an example of the SaaS provider.

Virtual machines are extensively used in cloud computing and nowadays, the concept of infrastructure as a service (IaaS) is widely synonymous of VMs. Many PaaS and SaaS providers are built on IaaS, meaning that they all run their workloads in VMs. However, Linux is the preferred OS to work in the cloud, thanks to its good performance, zero price, good hardware support and reliability. By reason of this fact, containers-based virtualization result as an interesting alternative to VMs in the cloud and although the key concepts underlying Docker containers(like namespaces) are really old, some kernel features required to implement containers in Linux have become mature only the last few years where Docker arise as build system for Linux containers.

III. Virtual machine - VM

Nowadays x86 server's limitations bring IT companies to run only one OS/application at time, on the same machine, with the result that even small data centers have to deploy many servers ending up in operating at a really low percentage of capacity. With VM Virtualization is possible to create a completely new virtual environment encapsulating an entire machine allowing the user to have the same experience of working with a real machine. Application and OS live in a separate software container called virtual machine or VM. VMs are completely isolated and computing resources, CPUs, storage and network are pulled together and delivered to each VM dynamically through a software called Hypervisor (abstraction layer because it first abstracts resources then reallocates them more efficiently). The latter emulates a whole client's PC allocating the shared resources needed to build the VM and to make it working. **Fig. 1** shows a typical structure of a VM. The Hypervisor is so able to emulate a variety of virtual hardware platforms isolated from each other, every one running its own OS instance (Linux, Windows, OS X) on the same underlying physical host while providing high efficiency. The *host machine* is a computer where an hypervisor runs one or more VMs while each VM is called *guest machine*.

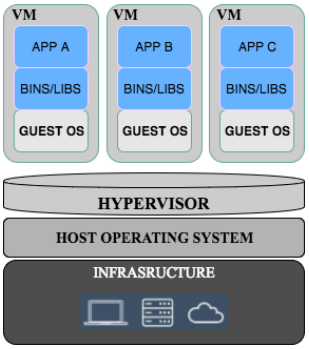

Figure 1: *A typical virtual machine*

In [5] Gerald J. Popek and Robert P. Goldberg have categorized two kind of hypervisor:

1. **Native hypervisors**, run directly on the host allowing to control the hardware and to manage guest's OS(reason why they are also referred to as bare-metal hypervisors). The first native hypervisors was developed by IBM in the 1960s IBM.[6] Some examples of native hypervisors are Microsoft Hyper-V[7], Xen[8], KVM[9], Virtual Box[10] and VMware[11].
2. **Hosted hypervisors** run on traditional OSs same as a program does and a guest OS runs as a process on the host. This kind of hypervisors abstract the guest OS from the host OS itself. Some examples can be VMware Workstation, VirtualBox or Parallels Desktop for Mac.

While the performance of this virtual system is not equal to the performance of the OS running on true hardware, the concept of virtualization works because most guest OSs and applications don't need the full use of the underlying hardware.

If all the servers are not running at full capacity, there is not need to have as nearly as many. By using VM's virtualization, hardware count and overhead drop dramatically while application's performance improves by leaps and bounds reaching greater values at lower costs with less complexity and faster maintenance. Again, no business can afford application downtime and because of this VM's virtualization make sure to provide high availability and fault tolerance in such a way that if a server fails, your application will stay up and running with non downtime, no data lost and no need for human intervention. [12]

Fig. 2 shows the structure of a simple web application by using VM virtualization. As it can be seen the VM contains the full OS, everything is in single system with all the things needed from the application to work (NGINX, PHP, MYSQL). The drawback to put everything in a single system is that administrators have to make sure that every update is supported by every other dependency used by the application itself. For instance, if a dependency like NGINX used in your application needs to be updated and to do that it also need to update the OS(let's say because the new NGINX version does not support the current OS) you have to be sure that all the other technologies support the new OS's version otherwise you are screwed.

i. VM's advantages

In [13] some advantages of using VM virtualization in respect of using physical servers are described as follow:

- Maximum server utilization, minimum server count – Every physical machine is used to its full capacity allowing you to significantly reduce costs by deploying fewer servers overall.

- A faster, easier application and resource provisioning VMs are self-contained software files that can be manipulated with copy-and-paste ease. Bringing simplicity, ease of management, speed and flexibility.
- Reduce the complexity of the environment.

As it has been described so far Virtualization helps to limit costs by reducing the need for physical hardware systems, power and cooling demand. If you need a new server, you can just create a new VM and if you need more memory or CPU in a device, you simply need to change the resources' allocation with a few clicks. System administrators can then benefit the use of virtual environments to simplify backups, disaster recovery, new deployments and also basic system administration tasks. In contrast with this, Virtaulization requires more bandwidth, storage and processing capacity than a normal Server or Desktop if the physical hardware is hosting multiple running VMs.

ii. How VM virtualization works

A VM can be created by using:

- ISO files in a repository (only hardware virtualized).
- Mounted ISO files on an NFS, HTTP or FTP server (only paravirtualized).
- VM templates (by cloning a template).
- Existing VM (by cloning the virtual machine).
- VM assemblies.

VMs may run in three domain types, paravirtualized (PVM), hardware virtualized machine (HVM) and hardware virtualized machine with paravirtualized drivers (PVHVM).

During the creation process of *PVM* type, the location of the mounted ISO file(NFS, HTTP or FTP server accessible to the VM on the VM network) used to create the VM is required. Moreover, PVM does not have a BIOS so the kernel has to be loaded directly and for this reason the VM needs direct access to a kernel that can be loaded at boot. The kernel of the guest's OS is then recompiled to be made available of the virtual environment optimizing memory, disk and network access to achieve maximum performance possible and to allow the PVM guest to run very close to the native speed.

HVM(or fully virtualized) instead requires to provide an ISO file preloaded into a storage repository where the VM is to be deployed. In creating HVM guests it may be required to activate the hardware virtualization in the BIOS of the server. The ISO indeed is configured as a virtual CDROM device inside the VM's BIOS and at boot step, the VM boots from the virtual CDROM in the same manner a common physical system does.

The third virtualization mode, *PVHVM*, follows the same line of HVM but it uses additional paravirtualized drivers to improve performance of the virtual machine since this domain type is used to run Microsoft Windows guest OS with an acceptable performance level. [14]

However, a really common way to deploy VMs is using either templates or assemblies. Templates are essentially copies of an already existing VM that can be reused and distributed to deploy a preinstalled/preconfigured VM quickly. Assemblies are way similar to templates but they can contain multiple VMs and are provided as Open Virtualization Format.

If a VM requires network connectivity to perform the operating system install, at creation point some virtual network interfaces will be generated and bridged to a network defined on the VM Server. All the VM's resources(configuration files, templates, assemblies, ISO files, shared/unshared virtual disks etc.) are stored in the storage repository, a logical disk space available through a file system on top of physical storage hardware, so that they can be available in a server pool(at least one) without having to copy them to each server. In a server pool, VM Servers can access to these resources by using a file-based storage(to multiple server pools) or through a physical disk-based storage(to a single server pool).

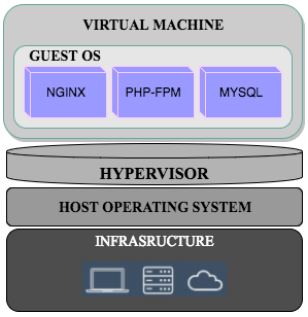

Figure 2: *VM of a simple Web APP*

Supposing that in **Fig. 1** the host machine contains Windows OS (it could also be Linux or MacOS). On top of that host there is the Hyper-

visor that mainly spawns processes (running something in the VM and then clicking on the task manager it can be seen that VM has actually multiple processes running) each one pointing to a VM that runs in a specific platform virtualization software (some of the most famous are Hyper-V from Microsoft, KVM from RedHat, Virtualbox from Oracle, VMware and so on) and each one utilizing the hardware resources like any normal process. Since every OS uses the hardware in a different way, Hypervisor does the magic through different drivers installed in it so that it gets compatibility with every OS.

A classic web application set up is shown **Fig. 2** where there is the host machine, the Hypervisor and three services (NGINX, PHP and MySQL).

IV. Docker Container

A lightweight alternative to the hypervisor is the so called container-based virtualization. [15, 16] Some example of this solution are: Docker[17], Solaris Container[18], Linux containers(LXC)[19] or OpenVZ[20]. In this work our attention is focused on Docker which is a technology based on open standards and allows containers to run on most of the Microsoft Windows, Linux and Mac distributions. Docker containers are a widely used alternative to build the cloud computing environment making such environment dressed with fundamental features, like scalability and elasticity. While in a VM's virtualization one or more independent machines run "virtually" on physical hardware through the *Hypervisor* intermediate layer, Docker containers run user space on top of an OS's kernel wrapping a piece of software in a file system and containing everything needed to run (libraries, code, binaries, system tools and so on). Reason why containerization is seen as a sort of "kernel level virtualization". It ensures that the software will always run the same, independently from its environment, allowing multiple isolated user space instances to be run on a single host. The aspect of sharing the kernel rise up the first security issue, since if you login into a container with root permissions(containers have two different states, they can run as privileged containers(root) or non-privileged containers(user)), you can screw up the kernel itself and the entire system. As known, Linux has policies limiting what an user can do outside its own space. This kernel shared, however, could be also considered an advantage because running on top of your Linux kernel make it really fast since it does not really need to boot a container e.g. supposing that you already have your Linux running, you can just keep spoiling containers without affect to much the performance of your system, unlike VMs.

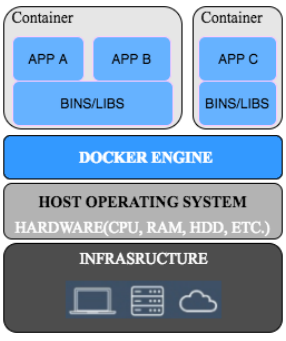

Figure 3: *Docker containers*

Due to the fact that they can only run the same (or a similar guest OS) as the underlying host, containers are considered with a lack of flexibility.[21]

In **Fig. 3** an example of Docker container architecture is shown. Compared with fig.1 it is easy to see that the Hypervisor is entirely cut off(now there is the Docker engine) to directly interact with the host machine's hardware. Each one of the previous services (NGING, PHP, MySQL) runs in its own OS using immediately the hardware it needs without going through the intermediate Hypervisor(this is way better because the Hypervisor slow downs the system). Contrary, the Docker Engine provides the core Docker technology that enables images(stateless, created once and used over and over again) and containers which represent the running state of images and so they change state. Each Linux container has a based Linux OS which share the kernel with the other containers without havaing to boot the entire kernel every time or without having intermediary serving those OSs.

Having an OS for each service used in your application in some cases could be a real big advantage because, based on your system, you can optimize the environment getting the optimal set up for each specific service.

i. Docker's advantages

Docker containerization provide a cost-effective alternative to hypervisor-based VMs. They have a bunch of single features helping in building the cloud computing environment which best fits the user needs[22]:

- **Lightweight**: Since containers share the same OS's kernel and images are built from a layered file system that share common files(binaries, libraries) it helps to use less RAM than VMs and to make disk usage much more efficient.
- **Rapid application deployment**, to reduce container size and to allow a quickly deployment, the minimal runtime requirements are included in each container of the application.
- **Easy building/sharing**: through the Docker Index. An application and all its dependencies can be wrapped into a single container independent from the platform distribution or from the host version of Linux kernel and it can be easily transferred and executed in another machine running Docker without compatibility issues. The container can also be shared with others by using a remote repository.
- **Namespaces isolation**, no process running into a container is able to see or affect other processes running in another container or in the host system.
- **Version control and component reuse**: Docker uses btrfs(a copy on write file system(CoW) for Linux) to keep track of challenging tasks, like tracking container's versions, inspecting differences or rolling-back to previous versions, leaving to the system advanced features and focusing on fault tolerance and easy administration. Containers reuse components from the preceding layers making them definitely lightweight.

ii. How Docker virtualization works

Docker is basically a *client-server* architecture where the Docker *client* talks with the Docker *server*(represented by a *daemon*) responsible of all the main tasks like starting and stopping containers, building and downloading images, exposing a REST API for remote management and distributing the Docker containers.

The Docker *client* is represented by a command line program used to communicate with the *daemon* through UNIX sockets, by using the REST API or a network interface. A developer will so interact with Docker by using such *client* to send commands to the server.

The Docker host (that could be any machine, a developer's local host, a server in the Cloud or in a physical/VMs within a data center) is the machine used to run the server. Such host machine must be running Linux kernel because Docker uses features that are only available to Linux. Both, client and server can be run on the same system or in different systems by connecting the client to a remote daemon.

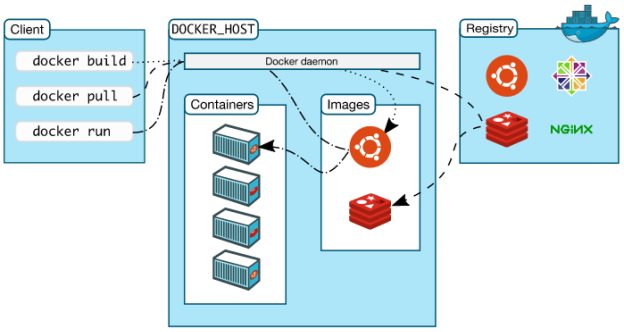

Figure 4: *Docker's architecture*

A graphical example of a simple Docker container architecture is shown in **Fig. 4** from[23]. The daemon runs on the host machine but the developer(user) does not interact directly with the daemon, instead through the Docker *client* which in turn talks to the Docker *daemon*. Internally docker is made by four fundamental objects[21]:

- **Docker Images**: read-only template(the building block of Docker) used to create and launch containers. Generally it contains the OS with all the web development tools and the web application [23].
- **Docker containers** are the **run** components that contain everything needed for the application to run. Containers are created from a Docker image and each one represents an isolated application platform [23].
- **Docker registries**: the **distribution** component used to store the images once they are built(compiled). (the public Docker registry is provided with the Docker Hub)[23].
- **Dockerfile** is a text document made by all the commands that a user can call to assemble an image. The Dockerfile can be considered as the "source code" while images as the "compiled code" for the containers which are represented as the "running code".

Images consist of several layers (files and directories from different file systems overlaid to create a single coherent file system) combined to a single image by a so called union file systems. They are built by using a series of basic steps (like run a command, add file/directory, create environmental variable, specify which process to run when a container is launched from this image etc.) stored in a *Dockerfile* and each step creates a new layer. Every image starts from a basic clean OS image (Ubuntu, Fedora, etc.) taken from *Docker Hub* or with an image of your own. When a developer requests to build an image, Docker goes through all the Dockerfile and executes every step eventually returning an image. An image is a filesystem and parameters to use at runtime. When a command is executed, Docker Engine firstly checks if the image is already present otherwise it downloads the image from the Docker Hub then loads the image into the container, runs it and the latter start getting data(that is the running state).

The lightweightness of Docker comes from these layers. When, for instance, an application is updated to a new version, Docker does not replace or rebuild the whole image (as it may happen with VMs) instead it creates a new layer which is the only thing that needs to be distributed.

The life-time of a container matches the life-time of the command specified in the Dockerfile. The container could be built and run for a single command execution then terminate right away, or it could be kept running by using a long-running command. Making use of Docker Engine allows developers not to care if the computer can run the software in a Docker image because a Docker container can always run it.

The underlying Docker's technology takes advantage from some Linux kernel's features making use of four important components:

- **Control groups**(cgroups) allow Docker Engine to share available hardware resources to containers, setting up limits and constraints like how many CPUs has to be used or how many cycles it should take or how much memory it should have.
- **Namespaces** provide the first and most straightforward form of isolated workspace. When a container is started, behind the scenes Docker creates a set of namespaces (and *control groups*) for that container and each aspect of a container runs in a separate namespace with the access limited on such namespace.
- **Union file systems**(UnionFS) are file systems that operate by creating layers, making them fast and lightweight. Those UnionFS are used by the Docker Engine to provide building blocks for containers.
- **Container format** represents the wrapper where Docker Engine combines cgroups, namespaces and UnionFS. The default is called *libcontainer*.

In an ideal deployment container-based applications are delivered as a series of stateless microservices.

V. Performance comparison

Along the years there has been a lot of different performance evaluation on hypervisors, although, most of them compared to non-virtualized execution or other hypervisors. [24, 25, 26] Comparison of VMs against Docker were also made in the past but mainly using older software like Xen or out-of-tree container patches. [27, 15]

Fig. 1 and 3 show that both VM and Docker container adopt a different structure. VMs represent an entire server including all the software and maintenance concerns, bringing high-level isolation and security since all the communication among guests and host are done through the hypervisor. However, this approach incurs in performance overhead(high performance overheads of hardware virtualization have been deeply studied in [25]) due to the hardware emulation. To reduce this hypervisor overhead many approaches have been proposed in [28] while Docker containers provide namespaces isolation and requires minimum run-time environment for applications to be configured. Furthermore, in the latter both the kernel and parts of the OS's resources are efficiently shared while VMs are isolated and a full OS must be included, that means, dedicated kernel for each application, duplicated binaries, duplicated libraries and also duplicated user spaces. Additionally, due to the need to run their own OS, VMs take a few minutes to come up and boot while containers only need a few milliseconds since they do not require to start any OS. In addition to that, at the creation step of both technologies, only one VM can be started from one set VMX and VMDK files in contrast with Docker

where many containers can be built with the same Docker image. Containers could also limit you since you are forced into a particular OS, but on the other side it can be a good thing as well considering that you don't have to worry about dependencies once your application is properly up and running in the container.

The architecture presented in Docker is built in such a way that dissimilarities between 2 different images can be tracked, for instance, if you start a container with a base Ubuntu image, then you add NGINX and you make a fresh image out of it, the new image will be a layer that depends on the OS(Ubuntu) layer so it will not contain the whole Ubuntu OS but just NGINX. Thanks to this, a large number of snapshots can be taken for backup reasons with Docker. With VMs instead this doesn't happen, mainly due to the fact that VMs are way heavier than containers. In [29] with 500GB HDD, it shows that 45 VMs snapshot of 10GB each can be generated while Docker uses half of the same HDD and 177MB of resources to generate more than 100 images.

Another dissimilarity which underline a drawback for Docker is about migration, VMs can be migrated also during execution while containers cannot and must be stopped before moving from an host machine to another one with the consequent waste of time and resources.

An interesting comparison has been done in [30], where, to explore the performance of traditional VM deployments(using KVM[9]) in contrast with Linux containers, high workloads were used to stress memory, storage, CPU and network's resources. Their results show that both Docker and KVM introduced trivial overhead on CPU and memory performance while KVM seems to add some overhead to every I/O operations making it less suitable for high I/O rates or latency-sensitive workloads. In contrast Docker adds overhead with networking in workloads with high packet rates. A similar test has been developed in [31], where both virtualization techniques in idle state and in CPU/Memory with high workloads test seem to behave in a similar way since they both make use of an optimized Linux power saving system, while, the network performance shows dissimilarities between them.

Another question that worths to be made is about the practice of deploying a technology inside another. Considering to deploy containers inside VMs, it results worthless if not useless since the operation would add the performance overhead of VMs without giving any benefit compared to the deployment of containers in a non-virtualized environment. The vice versa, that is, deploying a VM inside a container could be useful since it generates an extra security layer e.g. even if an attacker succeeds in hacking the hypervisor, it still will be inside the container. Overall results from [30] reveal that containers outcome in equal or better performance than VM in almost all cases. [31]

From the resulting studies presented in [32] its seems that containers "may suffer from performance interference in multi-tenant scenarios". Containers look also representing a viable alternative to VMs in concerns where rigorous isolation is not the most important thing(like big data processing[33] or high performance computing)[34].

VI. Evaluations and discussion

This work is not meant to find which way of virtualization is the best between the two analyzed technologies, no experiments have been running here, no plain metrics available, instead it is intended to give some guidance about how cloud infrastructure can/should be built in order to be profitable in respect to the use of physical servers and reduce the wastage caused by their under-utilization. Ordinary insight tells that *IaaS* should be implemented using VMs and *PaaS* by using containers. Of course, there are no fixed rules or technology constraints especially in situation where container-based IaaS can provide a more straightforward deployment or give better performance.

Virtualization brings up the advantage to reduce costs of maintenance and energy wastage which is not very surprising. Since the amount of energy wasted turns out to be function of the number of running physical servers, having them virtualized brings to have a much lower number of physical ones, allowing to preserve a lot more energy(go green!). Moreover, as you have less physical servers, only the ones you have need to be maintained with the result that maintenance becomes easier and cheaper and also helps to reduce the data center footprint(fewer servers, less networking gear, a smaller number of racks needed...).

Faster server provisioning is another valuable advantage of virtualization that enable flexible capacity to provide deployment at a moment's notice. A gold image, template or VM can quickly be cloned to get a server up and running in a small amount of time. Most virtualization's server platforms offer advanced features like live/storage migration, fault tolerance, high availability, and distributed resource scheduling helping to increase uptime and continuity. Combining servers to fewer physical machines in production, a business can easily make a cheap replication site, which in turn with the hardware abstraction capability by removing dependencies on a specific hardware vendor(or model), increases disaster recovery avoiding the requirement to keep identical hardware on hand in order to match the production environment.

As day passes, containers are getting extremely popular. Name any tech enterprise and you will see that is investing in containers. This of course does not mean that VMs are outmoded. They are not. The advent of containers is not meant to blow away VMs, rather those two technologies are complimentary. As it has been mentioned in the previous section, VMs can run inside containers(or vice versa) as they would run in any public cloud environment, they can work well together and they are supported together. Both technologies have similar resource isolation and allocation benefits, both allow to save money on hardware, consolidate management and use resources in a much more efficient way obtaining greater capabilities in power consumption, physical space, and energy reuse, although, their peculiar architectural/structural approach makes containers more portable and more efficient(in some way).

Whereas every single VM includes binaries, libraries, application and in principle an entire OS, all stuff that can yield in some GBs in size, containers only include the application with all of its dependencies sharing the kernel with other containers and running as isolated processes in the user space of the host's OS. Docker containers also augment the existing stack of software and infrastructure bringing additional capabilities like deployment, speed(and so on...) to the stack itself.

On important aspect that every business should take into account is the scope of their work and the classification of the personal workloads, not only in respect of the technology to use but also to understand where these technologies do make sense to be used, where they do not and if they are suited to handle such workloads and use cases. If your scope is to run a large amount of applications on a really small number of servers, you may want to create highly specific Docker containers each one intended to run one application (e.g. MySQL, Nginx, Php, or any other). Container technology allows to modularize applications and run components in their own separate space giving the the user the possibility to scale or update individual components independently. After all, Docker's quick startup times and low overhead make running multiple containers banal. On the other hand, if you are interested in having high flexibility coming up in running multiple applications you may want to use VMs which is scoped to any recent OS release(unlike containers where you are forced to one OS). Additionally, if one big concern is the budget, it is most probably best to use containers since they allow you to use more of them with way less physical servers.

In most cases Linux is the preferred OS to use in the cloud thanks to their free of costs, good performance, reliability and good support. Because of this and from the previous section Docker may appear as the right technology(implemented in short time and use resource economically though) to choose from the point of view of a business who wants to build its infrastructure on the cloud. However, a VM can be expressed like a new computer to the user which in turn can feel the easiness in managing and applying the policy of system, network, user, security.

To gain more efficiency in using VMs some kind of optimization could be used apart from the obvious hardware(BIOS and firmware) updates. A good practice on the network setup is it to create a separated isolated network used for moving workloads among hosts to provide fast and secure layer. Reducing the number of vCPUs(virtual CPUs) can also decrease the wait on host resources resulting in a VM's performance increase. Last but not the least is making use of templates(copy of a VM used to build clones) to speed up efficiency since templates allow administrator to deploy VMs rapidly.

The best approach, although, remains certainly not to get rid of VMs, rather to encourage the proper use of containers in addition to VMs when it worths.

References

[1] Dines Dwivedi Nagaraj Bhat K C Gouda, Anurag Patro. Virtualization approaches in cloud computing. *International Journal of Computer Trends and Technology (IJCTT)*, June 2014.

[2] M. Masud Rana Mohammad Mamun Or Rashid and Jugal Krishna Das. Implementation of the open source virtualization technologies in cloud computing. *arXiv:1605.03283v1 [cs.DC]*, page 19, 2016.

[3] Sara Angeles. Virtualization vs. cloud computing: What's the difference? `http://www.businessnewsdaily.com/5791-virtualization-vs-cloud-computing.html`, 2014.

[4] John Grundy Mohamed Al Morsy and Ingo MÃijller. An analysis of the cloud computing security problem. *APSEC 2010 Cloud Workshop*, September 2016.

[5] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[6] Shannon Meier. *IBM Systems Virtualization: Servers, Storage, and Software*. Redpaper, December 2015.

[7] Microsoft hyper-v. `http://www.microsoft.com/Hyper-V`.

[8] Xen. `http://www.xenproject.org/`.

[9] D. Laor U. Lublin A. Kivity, Y. Kamay and A. Liguori. Kvm: the linux virtual machine monitor. June 2007.

[10] ORACLE VirtualBox. Virtualbox. `https://www.virtualbox.org/`, 2016.

[11] Vmware. `http://www.vmware.com/`.

[12] vmware. Virtualization. `http://www.vmware.com/solutions/virtualization.html`, 2016.

[13] Matthew Portnoy. *Virtualization Essentials, 2nd Edition*. SYBEX, The Doker Book, 2016.

[14] Oracle. Understanding virtual machines. `https://docs.oracle.com/cd/E50245_01/E50249/html/index.html`, July 2016.

[15] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3):275–287, March 2007.

[16] Performance evaluation of container-based virtualization for high performance computing environments. *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, April 2013.

[17] Docker Inc. What is docker? `https://www.docker.com/what-docker`, 2016.

[18] Solaris containers. `http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html`.

[19] Linux containers. `https://linuxcontainers.org/`, 2008.

[20] Openvz. `https://openvz.org/`.

[21] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, The Doker Book, 2016 - v1.10.3.

[22] Inc. Red Hat. Linux containers with docker format. `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/7.0_Release_Notes/chap-Red_Hat_Enterprise_Linux-7.0_Release_Notes-Linux_Containers_with_Docker_Format.html`, 2017.

[23] Docker Inc. Understand the architecture. `https://docs.docker.com/v1.8/introduction/understanding-docker/`, 2016.

[24] Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev. Evaluating and modeling virtualization performance overhead for cloud environments. pages 7–9, 2011.

[25] Jinho Hwang, Sai Zeng, Frederick Wu, and Timothy Wood. A component-based performance

comparison of four hypervisors. pages 269–276, 2013.

[26] Richard McDougall and Jennifer Anderson. Virtualization performance: Perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev.*, 44(4):40–56, December 2010.

[27] Performance evaluation of virtualization technologies for server consolidation. *Enterprise Systems and Software Laboratory*, April 2007.

[28] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. Nohype: Virtualized cloud infrastructure without the virtualization. *SIGARCH Comput. Archit. News*, 38(3):350–361, June 2010.

[29] Il-Young Moon Oh-Young Kwon Byeong-Jun Kim Kyoung-Taek Seo, Hyun-Seo Hwang. Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters*, 66:105–111, 2014.

[30] An updated performance comparison of virtual machines and linux containers. *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, March 2015.

[31] *Power Consumption of Virtualization Technologies: an Empirical Investigation*. IEEE, December 2015.

[32] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Y. C. Tay. Containers and virtual machines at scale: A comparative study. pages 1:1–1:13, 2016.

[33] Miguel Gomes Xavier, Marcelo Veiga Neves, and Cesar Augusto Fonticielha De Rose. A performance comparison of container-based virtualization systems for mapreduce clusters. pages 299–306, 2014.

[34] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. pages 233–240, 2013.