A survey on preferable approach towards implementing Virtualization.

Tejas Jnanesh Ghalsasi

Dept. of Computer Science

California State University, Fullerton

Fullerton, CA 92831, United States of America

tejas.ghalsasi@csu.fullerton.edu

*Abstract*— Virtualization is a critical technology in the cloud computing domain due to its useful features such as server consolidation, power saving, live migration and faster server provisioning. There have been many approaches towards implementing virtualization. A key factor towards implementing virtualization is the Hypervisor. Out of the many hypervisors available in the industry, we are aware of the popular ones such as Xen, OpenVz. We also have KVM and LXC which provide kernel and Container level virtualization. In this survey paper we are going to analyse the best tool used for providing virtualization.

Keywords—hypervisor; virtualization; performance; cloud;

# Introduction

These days Cloud stages and related virtualization advances are in awesome request. Numerous product organizations such as VMware, Citrix (Xen), Microsoft (Hyper-V) rule the virtualization domain. Intel and AMD now offer propelled processors to help virtualization. Altogether these advancements are used for server combination—ordinarily in server farms that offer huge accumulations of servers for outer groups in an adaptable way through for instance flexible scaling. There has been much research identified with virtualization execution. Past research recognized that innovations which use hypervisor-based virtualization, confront elite overheads. What's more they endure from I/O constraints and henceforth have regularly maintained a strategic distance from in HPC conditions. As of late compartment based virtualization and support for micro hosting administrations has increased since it gives a lightweight arrangement that permits packaging applications also, information in a less complex and more execution oriented way that can keep running on various Cloud frameworks. Along these lines of managing with virtualization offers evenly adaptable, deployable frameworks.

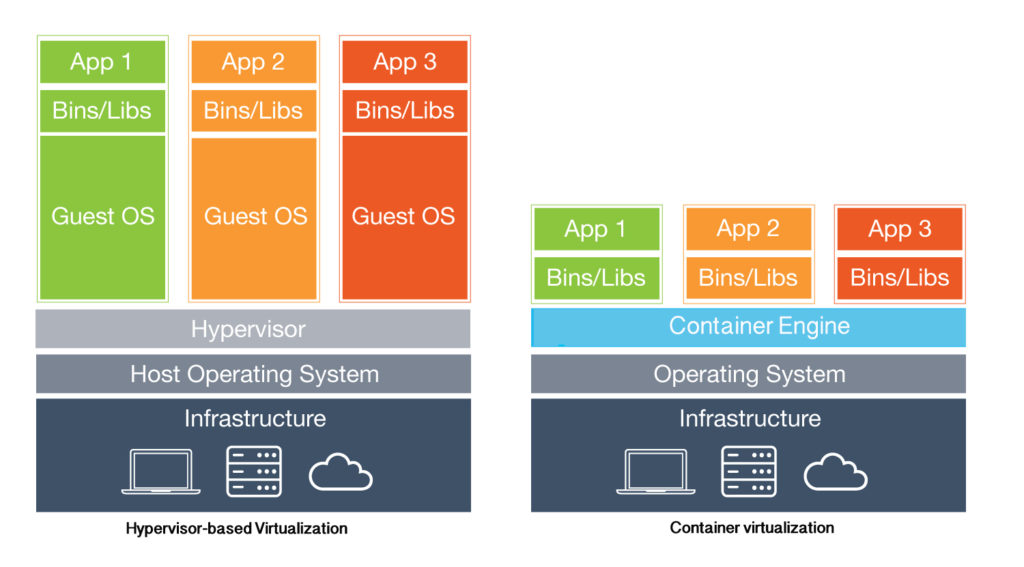
# Types of Virtualizations

The well-known procedure is hypervisor-based virtualization. Surely understood arrangements in light of hypervisor-based virtualization are: KVM and VMware. So as to utilize this sort of innovation there ought to be a virtual machine screen over the basic physical framework. Each virtual machine likewise has bolster for (disengaged) visitor working frameworks. It is very conceivable that one host working framework may bolster numerous visitor working frameworks inside this virtualization approach.

Container based virtualization innovation speaks to another approach. In this model, the equipment assets are partitioned by actualizing many examples with (secure) separation properties. There is distinction between the two innovations. Here, visitor forms acquire deliberations quickly with Container based advancements as they work through the virtualization layer straightforwardly at the working framework (OS) level. In hypervisor-based methodologies however there is normally one virtual machine for every visitor OS. One OS portion is commonly shared among virtual occurrences in Container based arrangements. Along these lines, there is a presumption that the security of this sort of approach is weaker than with hypervisors. From a client’s point of view, containers work as self-governing OSs, which seem ready to run autonomously of equipment and programming.

Hence There are two broad classifications of virtualizations:

1. Hypervisor based Virtualization
2. Container based Virtualization



Several virtualization solutions have emerged, using

different virtualization approaches. KVM and VMware use full

virtualization to provide a complete virtual environment for a

huge range of guests. Xen can use paravirtualization to

provide an API for guest domains, and solutions as OpenVZ

and Linux Containers can provide ways to partition OS in

logically isolated containers. Different virtualizations solutions

have different strengths and weakness.

# detailed analysis of technologies

## VMs can be grouped in Framework VMs, when are utilized to virtualize a whole OS, in resistance of Process VM, for example, those bolstered by the Java Virtual Machine. There are two types of VMMs, type I and II .

## A VMM type I, likewise called hypervisor or local, runs specifically over the exposed metal equipment in the most advantaged Ring, controls and offers assets to all VMs. The VMM type II keeps running as an application inside the OS, and is dealt with as a normal client space process. In the subsequent stage, we depict the virtualization methods to virtualize a whole OS.

## The present virtualization arrangements can make utilization of a significant number of virtualization methods above to achieve better execution. KVM and VMware make utilization of full virtualization, parallel interpretation and equipment help to virtualize any OS. These stages give Para virtualized drivers (VirtIO and VMware Tools, respectively) to permit their full virtualized visitor ass impart in a more straightforward manner with genuine equipment.

## These drivers are not CPU paravirtualization arrangements, they are insignificant, non-meddlesome changes introduced into the visitor OS that don't require OS kernel adjustment.

## Full Virtualization

* Full virtualization can virtualize any x86 working framework utilizing a mix of parallel interpretation and CPU coordinate execution methods.
* This approach makes an interpretation of bit code to supplant non-virtualizable guidelines with new successions of directions that have the expected impact on the virtual equipment.
* In the interim, client level code is specifically executed on the processor for superior virtualization.
* The VMM furnishes each VM with all administrations of the physical framework, including a virtual Profiles, virtual gadgets and virtualized memory administration.
* The visitor OS doesn't know it is being virtualized and requires no alteration.
* Full virtualization is the main choice that requires no equipment help or working framework help to virtualize touchy and advantaged guidelines. The VMM deciphers all working framework directions on the fly and stores the outcomes for some time later, while client level guidelines run unmodified.

## Binary Translation

* Parallel interpretation is a strategy utilized for the imitating of a processor design over another processor engineering.
* Accordingly, it permits executing unmodified visitor OS by copying one guideline set by another through interpretation of code.
* Not at all like unadulterated copying, paired interpretation is utilized to trap and imitate (or interpret) a little arrangement of the processor directions.
* That is, the code that necessities special execution, in Ring 0, for example, part mode. Whatever remains of the guidelines are executed straightforwardly by the host CPU.

## Hardware Assisted Virtualization

* Hardware helped virtualization for the x86 design, was presented in 2006 when Intel VT-x and AMD-V augmentations were discharged.
* Hardware helped virtualization executes a Ring with a higher special mode in the processor engineering. These augmentations bolster permits executing unmodified visitor ass in Ring 0 (non-root mode) also, the hypervisor in Ring - 1 (root mode).
* Hardware helped virtualization improves CPUs to help virtualization without the need of paired interpretation or paravirtualization. The advantage is that this system lessens the overhead caused by the trap-and-copy display, rather than doing it in programming here it is done in Hardware.

## Operating System Level Virtualization

* Here, the host OS is a changed piece that permits the execution of numerous confined holders (CT), otherwise called Virtual Private Server (VPS) or correctional facility.
* Every CT is an occurrence that offers a similar bit of the host OS.
* A few cases that utilize this system are Linux-VServer, OpenVZ and Linux Containers. The physical server and single occasion of the working framework is virtualized into different secluded parcels, where each segment recreates a genuine server. The OS piece will run a solitary working framework and give that working framework usefulness to each of the parcels.

## Kernal Virtual Machine(KVM)

* KVM (Kernel-based Virtual Machine) is a portion occupant virtualization framework for Linux on x86 equipment.
* KVM was the principal hypervisor to wind up some portion of the local Linux bit (2.6.20). KVM has bolster for symmetrical multiprocessing (SMP) hosts (and visitors) and backings endeavor level highlights, for example, live movement (to permit visitor working frameworks to relocate between physical servers).
* Since the standard Linux bit is the hypervisor, it profits by the progressions to the primary line variant of Linux (memory support, scheduler, et cetera). Improvements to these Linux parts advantage both the hypervisor and the Linux visitor OSs.
* KVM is actualized as a piece module, permitting Linux to end up plainly a hypervisor basically by stacking a module.
* KVM gives full virtualization on equipment stages that give virtualization guidelines bolster (Inter VT or AMDV). KVM has two noteworthy segments; the first is the KVM loadable module that, when stacked in the Linux piece, gives administration of the virtualization equipment, uncovering its abilities through the/dev document framework.
* The second part gives PC stage imitating, which is given by an altered rendition of the QEMU emulator. QEMU executes as a client space process, planning with the part for visitor working framework demands.
* At the point when a visitor OS is booted on KVM, it turns into a procedure of the host working framework and in this manner booked like any different process. Yet, dissimilar to different process in Linux, the visitor OS is distinguished by the hypervisor as being in the visitor mode (free of the part and client modes).
* The KVM module exports a device called/dev/kvm which empowers the visitor method of the bit. With/dev/kvm, a VM has its own particular memory address space isolate from that of the piece or whatever other VM that is running. Gadgets in the gadget tree (/dev) are normal to all client space process, however/dev/kvm is distinctive in that every procedure that opens it sees an alternate outline, it underpins detachment of the VMs .
* At long last, 110 demands are virtualized through a gently adjusted QEMU process that executes on the hypervisor. A duplicate of which executes with every visitor OS process.
* Other virtualization stages have been contending to get into Linux piece mainline for quite a while, (for example, UML and Xen), but since KVM required so few changes and was capable to change a standard part into a hypervisor, it's beautiful clear why it was picked.

## OpenVZ

* OpenVZ is a operating system-level virtualization technology for the Linux kernel. It consists of a modified kernel that adds virtualization and isolation of various subsystems, resource management and checkpointing. Open VZ allows a physical server to run multiple isolated operating system CTs.
* Each CT is an isolated program execution environment that acts like a separate physical server. A CT has its own set of process starting from init, file system, users, networking interfaces with particular IP addresses, routing tables, etc.
* Multiple CTs can co-exist on a single physical server, each one can operate different Linux distributions, but all CTs run under the same kernel , which results in excellent density, performance and manageability. Virtualization and isolation enable many CTs within a single kernel.
* The resource management subsystem limits (and in some cases guarantees) resources, such as CPU, RAM, and disk space on a per-CTs basis. All those resources need to be controlled in a way that lets many CTs co-exist on a single system and not impact each other.
* The checkpointing feature allows stop a CT, saving its complete state to a disk file, with the ability to restore that state later, even in another physical host.
* The Open VZ resource management subsystem consists of three components :
  + Two-level disk quota - Disk quotas can be set per-CT (first level) and inside CT, via standard Unix quota tools configured by the CT administrator.
  + Fair CPU scheduler - The Open VZ CPU scheduler is also two levels. On the first level it decides which CT to give the time slice to, taking into account the CT's CPU priority and limit settings. On the second level, the standard linux scheduler decides which process in the given CT to give the time slice to.
  + User Beancounters - Is a set of per-CT counters, limits and guarantees. There is a set of about 20 parameters that cover all aspects of CT operation, so no single CT can abuse any resource that is limited for the whole computer and thus do harm to other CTs. The resources accounted and controlled are mainly memory and various in-kernel objects such as IPC shared memory segments, network buffers, etc.

## Linux Containers

* Linux Containers (LXC) are light weight Kernel regulation usage upheld on few flavors or Linux like Ubuntu and Oracle Linux.

Key Characteristics of the Linux Containers are

• Process Each compartment is doled out a novel PID. Each

compartment can run a solitary procedure.

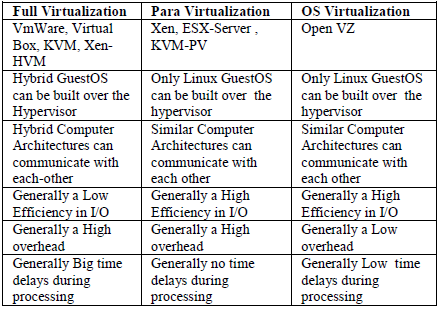
• Resource Isolation Uses cgroups and namespaces to

detach assets.

• Network Isolation Containers get a private IP address also, interface associating with a linux connect on the host.

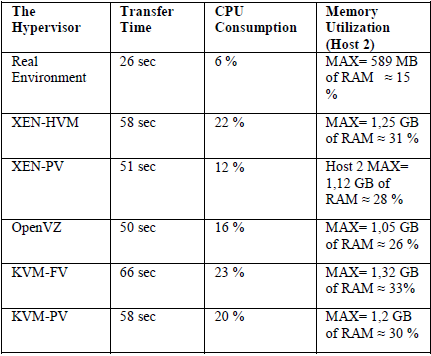
• File System Isolation Each compartment gets a private record framework by utilizing chroot.

* Linux Containers utilize Apparmor Security profile for solidifying of the host. It additionally utilizes cgroups to restrict device limit for the Containers.
* LXC utilizes a different record framework for each compartment which can be upheld by a LVM.



##### Performance evaluations

Various evaluation methods have shown varying results. Igli TAFA and Team has followed the Evaluation of Performance for 6 types of Environments based on HTTP Client/Server approach. They have summarized it in the following table:



1. Regarding to Transfer Time, the real environment has a performance approximately 2 times better than other hypervisors. It means that the Hypervisor includes a consider time delay during FTP transmission.
2. The real environment has a better performance in CPU consumption and Memory Utilization than the Hypervisors.
3. The Time Transfer in OpenVZ\_FTP Server is smaller than that of the other hypervisors. This happens because
4. OpenVZ hypervisor is less complex than the others. In OpenVZ each GuestOS has the same kernel as HostOS and each Guest looks like a simple process.
5. XEN-PV\_FTP Server has a better performance in CPU consumption (13), because XEN-PV has a better isolation between GuestOS and HostOS.
6. The Memory Utilization in XEN-PV\_FTP Server is better than the memory utilization in other Hypervisors. The second one is OpenVZ Hypervisor. The worse Memory Utilization is KVM\_FTP/Web Server. This is because KVM offers more overhead than other Hypervisors [3], [4].
7. 6.The Transfer Time in OpenVZ and XEN-PV Web Server is better than that of other hypervisors.
8. CPU consumption in KVM–PV is better than in KVM-FV because of the implementation of “virtio driver” in KVM-PV.

##### Acknowledgment

I would like to acknowledge the efforts of my mentor, Dr. Yun Tian from Computer Science Dept of California State University Fullerton. My progress would have not been the same without Dr. Tian.

##### References

1. G. R. Gangadharan, "Open Source Solutions for Cloud Computing," in Computer, vol. 50, no. 1, pp. 66-70, Jan. 2017.
2. E. Roush and Z. Thanga, "Zone Clusters: A virtual cluster based upon solaris containers," 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, 2009, pp. 1-8.
3. R. Dua, A. R. Raja and D. Kakadia, "Virtualization vs Containerization to Support PaaS," 2014 IEEE International Conference on Cloud Engineering, Boston, MA, 2014, pp. 610-614.
4. R. Rizki, A. Rakhmatsyah and M. A. Nugroho, "Performance analysis of container-based hadoop cluster: OpenVZ and LXC," 2016 4th International Conference on Information and Communication Technology (ICoICT), Bandung, 2016, pp. 1-4.
5. A. B. S., H. M. J., J. P. Martin, S. Cherian and Y. Sastri, "System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer," 2014 Fourth International Conference on Advances in Computing and Communications, Cochin, 2014, pp. 247-250
6. I. Tafa, E. Beqiri, H. Paci, E. Kajo and A. Xhuvani, "The Evaluation of Transfer Time, CPU Consumption and Memory Utilization in XEN-PV, XEN-HVM, OpenVZ, KVM-FV and KVM-PV Hypervisors Using FTP and HTTP Approaches," 2011 Third International Conference on Intelligent Networking and Collaborative Systems, Fukuoka, 2011, pp. 502-507.
7. M. Ahmed, S. Zahda and M. Abbas, "Server consolidation using OpenVZ: Performance evaluation," 2008 11th International Conference on Computer and Information Technology, Khulna, 2008, pp. 341-346.
8. P. R. M. Vasconcelos, G. A. d. A. Freitas and T. G. Marques, "Virtualization technologies in web conferencing systems: A performance overview," 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, 2016, pp. 376-383.
9. J. Ahrenholz, "Comparison of CORE network emulation platforms," 2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE, San Jose, CA, 2010, pp. 166-171.
10. N. Regola and J. C. Ducom, "Recommendations for Virtualization Technologies in High Performance Computing," 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, 2010, pp. 409-416.
11. Zhanibek Kozhirbayev, Richard O. Sinnott, "A performance comparison of container-based technologies for the Cloud", Future Generation Computer Systems, pp. , 2016, ISSN 0167739X.
12. Igli TAFA, Ermal BEQIRI, Hakik Paci, Elinda KAJO, Aleksandër XHUVANI "The evaluation of Transfer Time, CPU Consumption and Memory Utilization in XEN-PV, XEN-HVM, OpenVZ, KVM-FV and KVM-PV Hypervisors using FTP and HTTP approaches"