IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-6, Issue-3 E-ISSN: 2347-2693

A Comparative Study of Containers for Live Migration in Cloud Computing Environment

Anmol Bhandari[1*], Kiranbir Kaur[2]

[1*]Dept. CET, Guru Nanak Dev University, Amritsar, India
[2] Dept. CET, Guru Nanak Dev University, Amritsar, India

[*]*Corresponding Author: anmolbhandari60@gmail.com, Tel.: +91-9915074177*
Available online at: www.ijcseonline.org

Received: 23/Feb//2018, Revised: 28/Feb2018, Accepted: 21/Mar/2018, Published: 30/Mar/2018

***Abstract*—**Cloud computing, a modern world technology ensures that the resources are efficiently utilized, provisioned and are available all the time. When one virtual machine is running on one host and then urge rises to migrate that Virtual Machine to some other host on the same or different network; then live migration of Virtual Machines comes into the picture. In the simple live migration of processes, applications or virtual machines, process as well as operating system, on which the machine is operating needs to be migrated. But with the introduction of containers in the cloud computing, migration process becomes easy and less complex. Now, only the process or the virtual machine is only migrated to another host, regardless of the underlying operating system. Containers had made migration process easy because they contain inbuilt operating system in them. In this paper, we are going to review different containers that are used in the live migration of virtual machines.

***Keywords*—**Cloud Computing, Containers, High-Performance Computing, Live Migration, Virtualization

I. INTRODUCTION

[1] In distributed computing the essential and most imperative assignment is to arrange the accessible resources. The cloud resource administration procedures are fundamentally utilized for observing of administration demands and smart planning of processes [2]. Distributed computing turns out to be so prominent in light of the reality the resources accessible to a specific host isn't segregated. Virtualization makes it conceivable to separate and offers a wide range of resources in a disseminated situation. The limitation that was joining with respect to the equipment accessibility of resources is expelled in view of the virtualized conditions that gets rid of all the hardware inter-relations of the applications. Hypervisors or Virtual Machine Monitor (VMM) are used to execute the process of virtualization, for example, XEN which stacks the part and different conditions related to the visitor working operating systems [3]. Whenever there are undoubting circumstances that a virtual machine is operating on the top of the host and should be moved because of various reasons like support or absence of equipment resource accessibility. There are distinctive methods for accomplishing this, for example, ceasing the virtual machine and afterward migrating it to various host, however when the critical or important application is being run on the virtual machine, on the off chance that it should be exchanged without closing it down, that is the point at which you utilize Live virtual machine migration [3]. There are diverse situations when the virtual live migration truly has focal points, however in specific situations when the extent of the virtual machine is expansive, at that point this proposal is costly. The Cloud Computing framework gained its recognization essentially in the way that accessible resources can be shared flawlessly between virtual machines[3]. In the event that interoperability and convenience can be accomplished among cloud stages, it would help expel merchant secure as well as evacuate imperfections of a specific cloud stage. To accomplish portability and interoperability among cloud stages, it is necessary to provide a consistent migration of the condition of a specific application frame one cloud stage to another. We have diverse sorts of virtualization strategies out of which OS-level virtualization is utilized to join the containers based virtualization.

Dua, Raja and Kakadia[4] A container is a lightweight working framework running inside the host framework, running guidelines local deeply CPU, disposing of the requirement for direction level copying or without a moment to spare accumulation. Containers give funds in resource utilization without the overhead of virtualization, while likewise giving disengagement.

Since containers have a tendency to be significantly lighter in memory impression, they are favored design when contrasted with a VM for every application. VM is being utilized to have the containers. Different Services in Paas like Authentication, Routing, Cloud Controller, and Persistent Storage is still run straightforwardly on Virtual Machines.

Table[4] gives the mapping between the PaaS Product and the container usage utilized. A portion of the expansive PaaS players doesn't appear to utilize any container depending on VMs or Application runtime containers for different reasons. For instance, Google App Engine utilizes some type of Application Container to segregate occupant applications, however, does not utilize VMs or OS Level containers.

Table-1 : Mapping between PaaS Product and the Container Used

PAAS PROVIDER	IMPLEMENTATION
OpenShift	Docker with LXC
Heroku	LXC
CloudFoundry	Warden Container
Stackato	Docker with Warden Container
AppFog	Warden Container
Virtuozzo	Based on OpenVZ
DotCloud	Docker with LXC

One of the significant points of interest of doing OS level Virtualization is the straightforward migration of utilization[5]. Also, Osman, Subhraveti, Su and Nieh [6]call attention to that for a clear application migration, equipment, and Operating system Virtualization are the principal accesses. Osman, Subhraveti, Su and Nieh[6] go ahead to clarify that individual applications are executed in a virtualized situation with the approach of Virtualization.

Laadan and Nieh [5] underline the way that the process is confined inside the virtual condition making it conceivable to run an application inside the situation without affecting different cases of a similar application running in other virtualized situations, with OS Virtualization. Hence, application autonomy could be accomplished, which infers that distinctive adaptations of a similar application could keep running at the same physical machine and furthermore this whole idea could be executed on cloud surroundings in which physical machines could be supplanted by virtual machines. In spite of the fact that there are two sorts of OS Virtualization in particular host independent and host dependent OS Virtualization, application migration can be done by only host dependent OS Virtualization. Laadan and Nieh [5] illustrates that there is a private virtual namespace in host independent OS Virtualization that could confine and recognize the resources, referenced by the application. Container-based Virtualization is another name of Operating-System Virtualization.

Xavier et al [7] call container-based virtualization the lightweight elective for hypervisors and furthermore express that the occasions of utilizations running on container cases perform at a close local level because of its closeness to the base working framework. There have been the different usage of Linux container before; both hypervisors and Linux containers are developed innovations. It is as of late that Linux container has increased re-established consideration as a result of the requirement for quantity in the examples of utilization.

For instance, hosting organizations have a homogeneous surroundings where their servers are set up with the same application yet should be separated from one another, containers are the best suite for this sort of homogeneous conditions, in light of the fact that, regardless of what number of occurrences of containers are made in an OS it won't influence the framework execution dissimilar to VMs.

There are different executions of Linux containers, to give some examples OpenVZ, Docker containers and so forth. The principal advantage of a container over a virtual machine that is running on a hypervisor is container can instantiate rapidly in comparison to VMs due to the absence of the requirement of guest OS for the container. As far as the migration of containers is concerned, there are a couple of methodologies that have been actualized. Romero and Hacker[8] elaborate that there are three steps to live migration process in OpenVZ. At first, suspend the container and dump the memory onto a file. Then, replicate the dumped memory record onto the target machine. At last, resume the migrated container and halt the source container and clean the memory. In spite of the fact that downtime is reduced in live migration, but it is quick to kill and replicate a container on a different machine. The diverse methodologies towards migration impact the resources accessible in the cloud bigly. Romero and Hacker [8] have correlated live migration approaches with parallel applications and the outcomes recommend that the performance of an application is diminished with the successive checkpointing operations in the application.

Nadgowda, Suneja, Billa and Isci [9] Containers have been mainstream with the microservice design. In spite of the fact that container migration may appear redundant for stateless containerized applications, yet it is as yet appropriate to a few stateful micro service applications like databases (e.g. MySQL, Cassandra), message brokers (Kafka), and state-coordination service(zookeeper), among others. This is being recognized and upheld in standard systems like Kubernetes' 'StatefulSet'. In existing systems, Portability of stateful containers is likewise investigated, for example, ClusterHQ's Flocker, Virtuozzo[10] and Picocenter[11].

Flocker, particularly for Docker containers, principally is an information administration arrangement, It acts as a supporting system for migration for network-attached storage backends like Amazon EBS, OpenStack Cinder, and VMware vSphere and so on, by re-joining these systems stockpiling for containers. Nearby connected volume migration is upheld just for ZFS file system. On the other hand, Voyager is a non-specific, file system-skeptic and vendor-skeptic migration arrangement.

Virtuozzo[10] is an uncovered metal virtualization arrangement in which container virtualization is incorporated. It encourages Zero-downtime live migration for containers[10]. The first step in the process of this migration is the transfer of container's file system and virtual memory to the destination. When the exchange is done, it solidifies all container forms and incapacitates organizing. Then, dump this memory state to document and these dump records are replicated to the destination machine. The memory and disk blocks that are modified since the last transfer are the migrated to the destination host and at last, the container is assumed again. It has a hidden presumption that measure of memory pages and plate pieces changed (deltas) is little, in this manner blackout time indistinct. For any information concentrated application, these deltas exceptionally tenacious information changes could be substantial.

The main contribution of this paper is to enhance containers in live migration in cloud computing and increase time efficiency by considering space metrice and post migration algorithms. To optimize better results we will review some paper and find the better results to remove the barriers.Rest of the paper is organized as follows, Section II provides the overview of different containers used in the live migration technique, Section III contains the observation table of the different containers, Section IV contains the future scope in the area of containers, Section V contains the methodology followed and Section VI concludes the research work with future directions.

II. RELATED WORK

[4]Containers are methods for giving separation and resource administration in Linux condition. The term is gotten from delivery containers standard technique to store and ship any sort of load. A working framework container gives a non-exclusive method for disengaging a procedure from whatever is left of the framework. The regulation applies to all the youngster forms. Subsequently, one can boot a whole working framework by generating the init procedure.

It guarantees the same level of separation and security as a virtual machine and is all the more firmly coordinated with the host working framework. No reliance on equipment imitating gives execution benefits over full virtualization yet confines the quantity of upheld working frameworks which can be produced as visitor working frameworks; one can't boot Windows in a Linux/Unix Container. This is normal, not a necessity for PaaS suppliers and along these lines, containers are appropriate for giving low-overhead separation.

Presently, we will audit diverse containers that fuse the live migration and backs out the activity of virtualization.

A. Linux Containers

Linux Containers (LXC) are lightweight piece control usage bolstered on few flavors or Linux like Ubuntu and Oracle Linux.

Key Characteristics of the Linux Containers are,

- Process - Each container relegates an exceptional PID. Every container can run a solitary procedure.
- Resource Isolation-Uses cgroups and namespaces to disconnect resources.
- Network Isolation - Containers get a private IP address and veth interface associating with a Linux connect on the host.
- File System Isolation-Each container gets a private document framework by utilizing chroot.

The key favorable position of an LXC is a lightweight execution which performs at local places which giving better system and file system confinement. Presently, LXC experiences following the constraints:

- Containers utilize a Shared bit
- Limitations as far as for secure regulation condition
- Limited to Linux based conditions
- Implementation firmly fixing to a Linux Kernel

B. *Warden Container*

Warden container gives a piece free control execution which can be stopped to different fundamental Host OS. This container execution is utilized by Cloud Foundry task to have applications.

A portion of the Important properties of Warden Container are,

- For separation of Process and Network, namespaces are used.
- For isolation of resources, cgroup idea from Linux is used.
- Every container can run various procedures.
- Supported just on Ubuntu, however, the plan makes it OS unbiased

C. C. *Docker*

Docker is a daemon which gives the capacity to handle Linux containers as independent images. For the container execution, it uses LXC (Linux Containers) and also includes image administration and Union File System ability to it.

Docker container is a Linux container that authorizes the container based live migration. This container is a noticeable decision for application versatility and support. Despite the fact that Live migration is not supported by Docker containers but when combined with CRIU; it is feasible to

efficiently segregate a running application by stopping and continuing a Docker container. It implements live migration utilizing the strategies of checkpointing and live migration.

The key quality of Docker Containers is

- Process - Each Container is doled out a one of a kind procedure id and a private IP. Can't run various procedures in a solitary container.
- Resource Isolation - Uses cgroups and namespaces idea from Linux Containers.
- Network Isolation – It is attained by leverage the usefulness of LXC.
- File System Isolation- It is also attained by leverage the usefulness of LXC.
- Container Lifecycle – To utilize a daemon and command line, container lifecycle is managed.
- Container State – It is stored and recovered by the Docker enabled ability.

To enhance security, there are High-level Goals of Docker venture (which are constraints of Linux Containers)

- Root client of the container and non-root client of Docker are mapped
- Docker daemon is made to keep running as a non-root client

D. *OpenVZ*

OpenVZ utilizes an adjusted Linux Kernel with an arrangement of expansions. OpenVZ deals with numerous physical and Virtual servers, by utilizing dynamic ongoing apportioning. Like containers, OpenVZ has minimal overhead and offers higher execution and can be overseen superior to Hypervisor advancements. Much the same as different containers, OpenVZ utilizes cgroups and Namespaces. OpenVZ furthermore gives formats that assistance in precreated Virtual situations.

- Process - Each container has its own PID namespace, IPC namespace with its own mutual memory, semaphores and messages.
- Resource Management- Resource are being managed by sharing utilizing Bean Counters, Fair Share CPU Scheduler, disk portions in light of clients and Containers, and can oversee per container disk I/O need
- Network Isolation - Uses net namespace, has its own virtual system gadget, its own particular IP Address, channels and steering tables
- File System Isolation - Provides separation to Application Files, System Libraries and so forth.
- Container Lifecycle - Supports make, begin, change and stop works as part the lifecycle. Can be moved down, moved. Backings remote administration with the Libvirt API
- Container State - Provides Checkpointing highlight for putting away and recuperating the last known state. The Complete condition of the container like running procedures, open files, arrange associations, cradles, memory fragments can be put away on a document. This empowers Live Migration of OpenVZ containers

E. Voyager

Voyager[9] is a Just-in-Time (jit) Zero-duplicate migration arrangement, wherein the container is moved instantly before the entire information is replicated to the destination host. Also, the task of second information exchange performed by Virtuozzo is not required by Voyager, in this manner application downtime for Voyager is as yet littler than Virtuozzo. Promote Voyager gave highlights like information alliance get, double band information replication, OCI consistency which is not given by any current container migration arrangement.

Voyager[9] gives userspace-level filesystem-skeptic migration of locally relentless container state, while guaranteeing consistency over every one of these states, and also least application downtime. Basically, we are migrating container state crosswise over three distinct information stores, to be specific in-memory, neighborhood file system, and organize file system. We talk about each of these migration abilities underneath.

III. OBSERVATION TABLE OF DIFFERENT CONTAINERS

Table-2: Comparison Table of Different Containers in Cloud Computing

PARAMETER ->	PROCESS ISOLATION	RESOURCE ISOLATION	NETWORK ISOLATION	FILESYSTEM ISOLATION	CONTAINER LIFECYCLE
LXC	pid namespace is used	Cgroups is used	net namespace is used	Chroot is used	To create, start and stop a container,

					Tools lxc-create, lxc-stop, lxc-create is used
WARDEN	pid namespace is used	Cgroups is used	net namespace is used	Overlay File System using Overlays	To manage the containers, execute the commands on warden client which further talks to warden server
DOCKER	pid namespace is used	cgroups is used	net namespace is used	Chroot is used	Docker daemon and a client is used to manage the containers
OpenVZ	pid namespace is used	cgroups is used	net namespace is used	Chroot is used	Uses vzctl to manage container lifecycle
VOYAGER	pid namespace is used	cgroups is used	net namespace is used	userspace level filesystem-skeptic migration of locally persistent container state is provided	-nil-

IV. METHODOLOGY

Following are the steps that are considered in the methodology:-

1. Available resources are checked, whether available or not
2. The memory utilized and the required memory of the destination node is checked.
3. The space required for creating a new container is checked.
4. The fitness of destination node whether it will be able to handle the migration process is checked
5. After all the evaluation, memory and space are reserved on the destination node
6. The same base image of the to be migrated application is pulled down from the global repository
7. Network proxy is initiated and diverted to the destination node
8. All the traffic is diverted to new container and initial container is removed or paused.
9. Evaluate and analyse the post migration process for better results

V. FUTURE SCOPE

Intercloud migration involves data and applications to the destination host. In the existing literature, the focus is to migrate the Linux container to the host machine depending upon the availability of the resource. Job requirements and fitness of host machine are not considered during the migration process. The fitness of destination host is given in terms of available resources such as RAM, and processors. The speed with which host can complete the operation is also not considered. The metric which is considered in existing literature is a space constraint. Migration consumes less time through the proposed technique but the execution of the job i.e. post-migration process is not considered that can be optimized in future work.

V.I OBJECTIVE

The proposed work consider the post-migration process to decrease overall execution time associated with the migration. Fittest machine on the basis of available resource can be selected in order to save from starvation problem. The objective in terms of parameters is listed as under

1. Minimise overall execution time associated with migration.
2. Enhancing reliability by ensuring checkpointing in case of task failure.
3. Cost in terms of overhead required to be minimized.

VI. CONCLUSION

Containers are the alternate choice to virtual machines in cloud computing but with better and advance features. Containers assure smooth-running, effortless employment and configuration, secured and safe techniques of administering particular system requirements. Moreover,

Containers enables to execute multiple applications to run on a single operating system by making instance of the operating system. Containers are better than a virtual machine for live migration considering every aspect of both. Containers are not required to instantiate the guest Operating system on the host machine, which results in the decrease in the migration time, other performance attributes are also improved. To improve the overall performance, speed and fitness of host machine should be considered and post migration algorithms can also be taken in account.

REFERENCES

[1] C. Computing and I. Dublin, "Inter-Cloud application migration and portability using Linux containers for better resource provisioning and interoperability Ivin polo sony," no. September 2015.

[2] R. Buyya, "Market-oriented cloud computing: Vision, hype, and the reality of delivering computing as the 5th utility," *2009 9th IEEE/ACM Int. Symp. Clust. Comput. Grid, CCGRID 2009*, vol. 25, no. 6, p. 1, 2009.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *Proc. Ninet. ACM Symp. Oper. Syst. Princ. - SOSP '03*, p. 164, 2003.

[4] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," *Proc. - 2014 IEEE Int. Conf. Cloud Eng. IC2E 2014*, pp. 610–614, 2014.

[5] O. Laadan and J. Nieh, "Operating system virtualization," *Proc. 3rd Annu. Haifa Exp. Syst. Conf. - SYSTOR '10*, p. 1, 2010.

[6] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design and implementation of Zap: a system for migrating computing environments," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 361–376, 2002.

[7] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High-Performance Computing Environments," *2013 21st Euromicro Int. Conf. Parallel, Distrib. Network-Based Process.*, pp. 233–240, 2013.

[8] F. Romero and T. J. Hacker, "Live migration of parallel applications with OpenVZ," *Proc. - 25th IEEE Int. Conf. Adv. Inf. Netw. Appl. Work. WAINA 2011*, pp. 526–531, 2011.

[9] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete Container State Migration," *Proc. - Int. Conf. Distrib. Comput. Syst.*, no. Section III, pp. 2137–2142, 2017.

[10] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," *2008 Linux Symp.*, pp. 1–8, 2008.

[11] L. Zhang, J. Litton, F. Cangialosi, T. Benson, D. Levin, and A. Mislove, "Picocenter," *Proc. Elev. Eur. Conf. Comput. Syst. - EuroSys '16*, no. Llmi, pp. 1–16, 2016.

Authors Profile

Anmol Bhandari pursed Bachelor of Technology in Computer Science from Guru Nanak Dev University, Amritsar in year 2016. She is currently pursuing Masters of Technology in Computer Science from Guru Nanak Dev University,Amritsar and currently working as Research Scholar in Department of Computer Science. Her main research work focuses on Virtualization, Live migration in Cloud Computing.

Kiranbir Kaur pursed Bachelor of Technology and Master of Technology in Computer Science from Guru Nanak Dev University,Amritsar, India in year 2008. She is currently pursuing Ph. D. and currently working as Assistant Professor in Department of Computer Science,Guru Nanak Dev University, Amritsar. She has published more than 10 research papers in Ugc approved Conferences. Her main research work focuses on Cloud Computing Interoperability and Portability, Cloud Security and Privacy. She has 6 years of teaching experience.

