Indonesian Journal of Electrical Engineering and Computer Science
Vol. 12, No. 3, December 2018, pp. 1094~1105
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i3.pp1094-1105

Integration of Linux Containers in OpenStack: An Introspection

Ashish Lingayat[1], Ranjana R. Badre[2], Anil Kumar Gupta[3]
[1,2]Computer Engineering, MIT Academy of Engineering, Dehu Phata, Alandi (D), Pune, 412105, Maharashtra, India
[3]HPC Infrastructure and Ecosystems, C-DAC, C-DAC Innovation Park, Pune, 411008, Maharashtra, India

Article Info

Article history:

Received Mar 28, 2018
Revised Jun 15, 2018
Accepted Sep 25, 2018

Keywords:

Cloud computing
Container orchestration
Linux containers
Open stack

ABSTRACT

In cloud computing, sharing of resources is supported using heavy weighted traditional virtualization techniques. Such techniques involve hypervisors to emulate hardware for creating virtual machines. The inclusion of an additional layer of hypervisor over host operating system depreciates the performance of the virtual machine. Recent evolution is a lightweight alternative to the virtual machine called containers which have gained popularity among developers and administrators. Container Based virtualization has proven very efficient regarding performance, and many industries are now migrating their virtualized environment to run on Linux containers. Containers use host operating systems kernel and isolate each container by encapsulating them with their required services and packages. Linux kernel is very beneficial in implementing containers, which is the reason for the existence of Linux containers. Linux containers utilize less storage space and consume optimal computational power, giving a hike in performance. Having them integrated into the cloud surely benefits consumer and cloud provider. Many projects have extended their support in incorporating containers in the cloud. In this paper, we will discuss various Linux containers and their management tools along with cloud computing software, OpenStack, including projects undertaken by OpenStack for integrating containers in the cloud.

Corresponding Author:

Ashish Lingayat,
Computer Engineering, MIT Academy of Engineering,
Dehu Phata, Alandi (D), Pune,
412105, Maharashtra, India.
Email: ashishvijaylingayat@gmail.com

1. INTRODUCTION

Virtualization technique is used to virtualize native hardware for running guest operating system known as virtual machine. Virtualization has been playing a very significant role in cloud computing which results in its extensive usage. Maximum utilization of resources is carried out by sharing them with virtual machine. The technique of virtualization induces an overhead in the performance of guest operating system by consuming storage, memory and wasting CPU resources [1]. In virtualization, hypervisor is used to emulate hardware for operating system to run on it. Virtualized environment has separate kernels for host operating system and guest operating system. The hypervisor is an abstract layer between guest operating system and native hardware. Having hypervisor between guest operating system and host hardware adds overhead to the performance of virtual machine and ultimately slows the working of virtual machine due to emulated hardware. Figure 1 shows the architecture of hypervisor and Linux containers. It shows us the

layers involved in implementing virtualization. This limitation of virtual machine has led to the development of Linux containers.

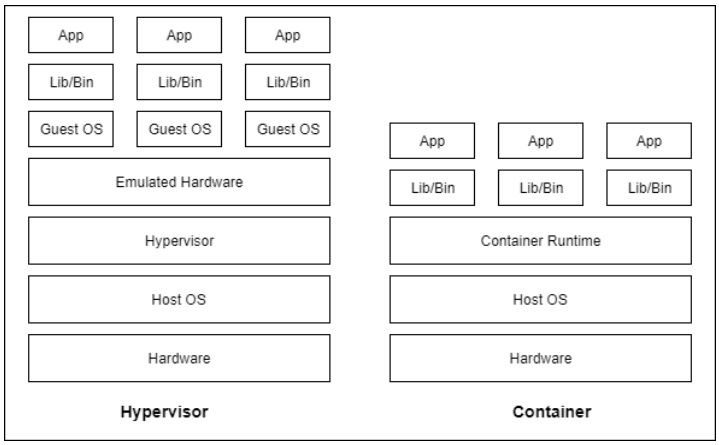

Figure 1. Hypervisor vs. Linux containers

Linux containers are lightly weighted as they consume less amount of space and have shorter deployment time [2]. They are easily manageable and help in maximizing utilization of computing resources. Though Linux containers use the host operating system kernel, there is isolation between them by using various encapsulating techniques. In Linux containers, interoperability and portability have made them prevalent in today's competitive market. Recently, Microsoft has also started building Windows containers and have extended their support in integrating them with Windows system.

In the cloud, management of the resources is very vital in efficiently running services or applications [3]. Underutilization of resources will not benefit consumer nor the cloud provider. To improve utilization of resources, need of managing the resources by using scheduling techniques is beneficial [4], [5]. Features of Linux containers have led to the integration of Linux containers in the cloud that simplifies the management of applications. Projects are being undertaken to achieve maximum integration of containers in the cloud. The scheduling techniques for containers in the cloud will help in proper utilization of resources thus benefiting the cloud provider and consumer [6]. There is a considerable number of projects working in containers and cloud, but we will review a few popular ones amongst them. Red Hat's OpenShift, Amazon Web Services has its Amazon Elastic Container Service (Amazon ECS), Microsoft Azure uses Fabric, Google Cloud Platform (GCP) utilizes Google Container Engine (GCE), and OpenStack has Magnum, Nova Docker Hypervisor, Kolla, Zun, Kata for managing containers in the cloud.

OpenStack, an open source cloud software, is widely adopted by cloud providers due to its features and has proven an excellent platform for integrating Linux containers. System containers get managed easily in OpenStack since it provides Infrastructure as a Service (IaaS). Open source projects are independently integrated into OpenStack to provide cloud infrastructure. These individual projects get deployed autonomously for fulfilling assigned workload, such as cinder is implemented to provide storage, Neutron for networking. Similarly, OpenStack has integrated open source projects for building and utilizing maximum benefits of Linux Containers. OpenStack Magnum project offers an Application Programming Interface (API) for interacting with Container Orchestration Engines (COEs) [7]. OpenStack Zun is used for managing container images by utilizing resources provided by OpenStack. OpenStack Kolla is a platform for deploying production ready containers. Recently, OpenStack Kata project has been announced to combine benefits of virtual machine and containers.

2. CONTAINER

Containers are a lightweight alternative to virtualization. They are portable and consume less memory which can account for MB's as compared to virtual machines where they consume memory in GB's [8]. They are used to store entire application having all of its dependencies, libraries and configuration files required for running them. Containers are isolated from each other by using various packaging techniques [9]. Features of containers ensure interoperability and portability of applications [9].

” Containerization is nothing but the process of abstracting all the differences in operating system distributions and their underlying infrastructure by encapsulating discrete components of application logic, including the application platform and its dependencies, with the help of lightweight containers [10].”

Types of containerization:

1) Operating system containerization: Operating system containerization provides user-space isolation and utilizes the kernel of the host operating system. They are similar to virtual machine but not precisely virtual machine. For creating operating system containers, we can use techniques such as OpenVZ, LXC, Linux VServer, Solaris and many more.
2) Application containerization: Operating system containers run multiple processes and services whereas application containers are specific to an application. Rocket, Docker are examples that provide application containers.

Advantages of containerization:

1) Containers are lightweight as they occupy lesser storage space than virtual machine.
2) Increased CPU usage over virtual machine.
3) Containers are highly portable.
4) Due to the reduced size of containers, many number of containers can be running on a single host.
5) CRUD operations can be performed smoothly on them.

Drawbacks of containerization:

1) Complete isolation between containers is not possible.
2) They are vulnerable with aspect to security.
3) Containers are not the whole replacement to virtual machine as they use host operating system kernel.
4) Features of containers are:
5) Replication: Identical images of containers can be incorporated containing a complete application.
6) Testing and isolation: Container images are isolated and packaged to carry their dependencies, libraries, binaries which are required to run their service or process. Isolation ensures proper running and abstraction of containers in any environment making it suitable for testing accurately.
7) Scalability: Creation and termination of containers are carried out by building multiple containers. The orchestration tools used for scaling containers are Docker Swarm, Apache Mesos, Kubernetes.
8) Performance: Containers are lightweight and do not have any overhead layer when deployed on host operating system. This contributes to maximum performance as compared to virtual machine.
9) High Availability: High Availability of many containers is possible in clusters because they require less storage and memory and has faster deployment rate.

2.1. Linux Container (LXC)

Linux Containers are developed using C, Python, Shell, and Lua language that aim for offering environment similar to a complete virtual machine. It is free software licensed under GNU LGPLv2.1+ license [11]. It implements system-level virtualization using cgroups [12], namespaces and access control thus isolating containers from each other [13]. Performance of containers is increased by reducing the overhead of using multiple kernels and emulated hardware. Projects developed by linuxcontainers.org are LXC, LXD, and LXCFS.

a) LXD: Founded by Canonical Ltd. has contributors from all over the globe. LXD acts as a manager for system containers. LXD is deployed on top of LXC for improving the user experience [14].
b) LXCFS: The code of LXCFS is written in C language. Developed to overcome the disadvantages of Linux kernel while handling file system for containers [15].
c) LXC: It provides an interface to the user for containing features of Linux kernel. Uses API and tools for managing application or system containers [11].

For containing process, LXC uses following features of Linux kernel [16]:

a) Kernel namespaces
b) SELinux profiles
c) Seccomp policies
d) Chroots
e) Kernel capabilities
f) Control groups (cgroups)

2.2. Docker

Docker is developed in Go programming language [17] and uses features of Linux kernel. Docker is a container based platform designed to ease creation and running of an application. It is open source platform for building, shipping, and running distributed system. Docker system consists of Docker Engine, portable and lightweight; a runtime of packaging tool; and Docker Hub. Applications can be quickly

combined and ease the tasks of development, and quality assurance in a production environment. For isolation, it uses various isolation groups of Linux such as namespace, cgroups, and UFS. Each container runs within that namespace and does not have any access outside it.

Control groups (cgroups) provide control over resources. Cgroups allows Docker for sharing available resources for giving efficient multi-tenant environment on the host. Cgroups can have reservation or limitation to resources for a container.

UFS (Union File System), or UnionFS [18] is a file system used for making Docker lightweight and faster by creating layers. Docker uses UnionFS variants such AUFS [16] Device Mapper.

Container format combines all these components and packages it. By default, the format for container wrapping is libcontainer. Docker containers can run on all popular Linux distros due to its open standards. Docker containers can run on virtual machine, native hardware, cloud infrastructure provided by Google, Microsoft, Amazon Web Service and other cloud providers. Each Docker container will have its root file system, processes, memory, networks, namespace, to have strong isolation.

Benefits of using Docker containers:

a) Adding or removing of containers ensures the scaling of services.
b) They are built very easily and quickly and their deployment time is also very less.
c) The efficiency of Docker containers in regards to resource utilization is very high due to limitation and other resource management strategies.
d) The density of Docker container can increase to handle more workloads.

2.3. Rkt

Rkt (spelled as "rocket") developed by CoreOS, Inc. [20] in December 2014 is open source container engine designed for managing application containers. The work of rkt is by default licensed under Apache 2.0 license unless specified [21].

Features of rkt's are:

a) Pod-native
b) Security
c) Composability
d) Open standards and compatibility

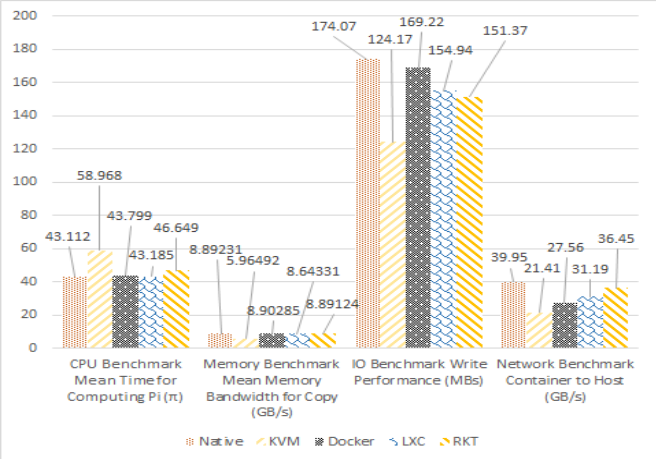

Figure 2. Performance comparison between baremetal, virtual machine & containers

Figure 2 shows the comparison between baremetal, KVM, Docker, LXC, and Rkt where the values were obtained by M. Ali Babar and Ben Ramsey [22] on performing benchmarks for knowing the performance of native machine, Linux containers, and virtual machine (KVM). The results show that lesser the time better is the CPU performance while doing benchmarking for CPU. The memory bench for copy is showing the mean memory bandwidth in MB/s. Here, higher the bandwidth higher the copy rate. Input-Output (IO) benchmark performed for write operation calculated in MB/s shows higher the write rate, higher the performance. Data is sent from container to host and calculated in GB/s. In respect to KVM, the container is deployed on top of KVM, and then data is sent to host. Higher the rate of transfer more efficient is the network performance. According to the results shown in above Figure 2, one can say that

there is no competitor for baremetal but in some cases, containers give equal performance. Discussion on containers and virtual machine show that LXC is better than rkt, Docker, and KVM respectively in CPU performance. While Docker is better than rkt, LXC and KVM regarding memory Bandwidth, similarly for IO write operation LXC is better than rkt while others have same performance obtained with memory bandwidth. Network performance is given better by rkt than LXC, Docker, and KVM respectively. One can see performance variation in containers, but overall they perform better than virtual machine.

Performance for startup time and file copy performance shows that native is much better than Docker, while Docker performs very well than KVM [23].

Table 1 shows the performance comparison between native, KVM, and Docker environments obtained by Wes Felter et al. [24] show that native is always a better option for virtual machine (KVM), and in some cases, Docker performs equal to native. While calculating network latency, one can see that KVM performs better than Docker although not better than native. The results show performance benchmarks given on CPU performance, memory bandwidth, RAM, network latency, block I/O, and network bandwidth.

The reason behind the decline in performance of the virtual machine is the existence of additional layers, whereas Docker has fewer layers and directly interacts with the kernel of the host machine. There are differences seen between different container formats due to their architecture and the purpose of their development.

Table 1. Comparison of native, virtualization and Docker container

	Native	KVM	Docker
CPU (PXZ) compressing 1GB of Wikipedia data	76.2 (±0.93)	59.2 (±1.88) lesser 22%	73.5 (±0.64) Lesser 4%
Memory Bandwidth (Stream) Copy (GB/s)	41.3 (±0.06)	40.1 (±0.21) lesser 3%	41.2 (±0.08) Lesser 0%
RAM (Random Access) GUPS	0.0126 (±0.00029)	0.0125 (±0.00032) lesser 1%	0.0124 (±0.00044) Lesser 2%
Network Latency (Netperf)	less than 40μs	less than 70μs	more than 70μs
Block I/O(fio) (IOPSInput/ output operations	Read IO is near to 80000 IOPS write is above 100000 IOPS	Read IO is near to 40000 IOPS write is below 60000 IOPS	Similar to Native Similar to Native
Network Bandwidth (nuttcp)	Transmit aound 2 cycles/byte Receive above 2 less than 2.5 cycles/byte	Transmit between 2-2.5 cycles/byte Receive above 3 cycles/byte	Above 2.5 cycle/byte Receive between 2.5 to 3 cycles/byte

3. CONTAINER ORCHESTRATION ENGINE

Containers are deployed very densely, where it becomes extensively tricky for managing such enormous number of containers. Orchestration techniques are used to achieve automation for handling containers. Container runtime API can perform all the operations on a container, but it is limited to a single container. Actual problem arises when multiple containers are running on various hosts. To have the efficient management of container and to overcome above obstacles we use orchestration tools.

Complex and multiple container applications deployed in a cluster are carried out by expanding capabilities of Container Orchestration Engine [13]. All management tasks performed on containers from its development, deployment, and scheduling till its termination is automated using container orchestration tools. OpenStack supports only Docker Swarm, Apache Mesos, and Kubernetes as far its current version Pike is concerned. Magnum, one of open source project from OpenStack provides these orchestration tools.

For achieving automation and managing containers different tools are available:

1) Linux Containers (LXC): LXC is the umbrella project behind LXC, LXD, and LXCFS.
2) Apache Aurora: Intelligently reschedules failed containers over other running machines.
3) Docker Engine: An application is shared and run across various Linux systems thus providing a platform for administrators and developers.
4) Kontena: They have been built to be deployed and run on any infrastructure. Applications are developed using Kontena Service.
5) Weaveworks: Contains tools for managing and implementing microservices in clustering.
6) Wercker: Autonomously uses pipelining concept 'automated workflow' for deploying multi-tiered cloud native applications.
7) rkt: Containers managed by using Command Line Interface (CLI).

3.1. Docker Swarm

Docker Swarm is an open source cluster and orchestration management tool [25]. Docker provides a standard API for communicating with containers. Swarm uses" swap, plug and play" for Docker containers. Swap ensures that Docker containers are highly portable even when running. Docker Swarm architecture consists of master nodes and worker nodes. The configuration of Swarm cluster can be declaratively done using YAML files [26].

Keywords used in Docker Swarm:

a) Node: Distributed across on-premises or public cloud. An instance of a Swarm.
b) Swarm: Orchestrating services in a cluster is automated.
c) Manager node: The user sends distributed service definition to worker nodes. They can themselves act as worker node.
d) Worker Node: Receives and runs tasks assigned by the master node.
e) Service: Service is intended for specifying container and replicas.
f) Task: Is an atomic unit of a service assigned to worker node.

Features of Docker Swarm:

a) Inbuilt clustering using Docker Engine.
b) Decentralized design.
c) Declarative Service Model.
d) Scaling.
e) Multi-tenant discovery of service, fault tolerance.
f) Rolling updates.

3.2. Kubernetes (abbreviated as K8s)

" Kubernetes is an open source tool for deployment and management using automation for the containerized application [27]." Kubernetes was developed by Google for internal cluster management was previously known as Borg (a.k.a. Omega) [28]. It was in June 2014 that Google announced its open source cluster management tool called Kubernetes in Google Developer Forum [29]. Later Google donated it to Cloud Native Computing Foundation. It provides a uniform API for managing containers in the cluster of native and virtual machine.

Kubernetes uses its keywords and also adds new concepts. Components of Kubernetes are:

a) Node: Native or VM, running on top where containers are scheduled.
b) Pods: They are set of containers combined logically.
c) Services: Services are provided using IP address and network protocol.
d) Replication Controller: Used for efficient management of the clustered environment.
e) API server: Kubernetes master node uses API as management hub.
f) Controller Manager: The desired state is matched with current state on scaling workloads in a cluster.
g) Scheduler: It assigns workload to an appropriate node.
h) Kubelet: Manages pods that are running on the host, by receiving configuration from API server.
i) Labels: Key-value pairs used for searching and updating containers.

3.3. Apache Mesos

Apache Mesos is also called as the kernel of distributed systems, as it is more open than Docker and Kubernetes and has fine granularity. In Mesos, resource management and task scheduling is implemented individually. Apache Mesos provides an abstraction for computing resources in large clusters. This shared pool is created by combining CPU, memory, and storage. These shared pools are then allocated as per fine-grained resource requirements. Apache Mesos is fault-tolerant and scalable for massively scaling applications or services [30].

Mesos provides distributed systems using:

a) Apache Aurora: It is service scheduler for long-running containers and has the capability of scaling them highly. Additional features of Aurora are rolling updates, a quota for resources and service registration.
b) Chronos: It is fault-tolerant scheduler used for replacing cron jobs to orchestrate containers.
c) Marathon: It is service scheduler built on top of Mesos and Chronos.

3.4. Microsoft Azure Service Fabric

It is a platform for distributed systems used for encapsulating, developing and deploying containers. The deployed services are scalable, highly manageable and reliable for the production-ready environment.

Capabilities of Service Fabric are:

1) Provisioning.
2) Deploying.
3) Monitoring.
4) Upgrading.
5) Deletion.

Services provided by Microsoft's Service Fabric are [31]:

1) Azure SQL Database.
2) Azure Cosmo DB.
3) Cortana.
4) Microsoft Power BI.
5) Microsoft Intune.
6) Azure Events Hub.
7) Azure IoT Hub.
8) Dynamic 365.
9) Skype for Business.
10) Moreover, many more.

Services can be built using Service Fabric Programming models [32], ASP .NET core [33]. It has the capability of building stateless or stateful services. Microservices can orchestrate and automate in Service Fabric [34].

Windows container types:

1) Windows Server Containers [35]: It utilizes kernel of the host to achieve isolation by abstracting process and namespace. Limitation of Window server container is the requirement for the same version of kernel and configuration due to the utilization of host kernel.
2) Hyper-V Isolation [35]: Microservices run in a virtual machine which is highly optimized. They do not share a kernel of Host and thus are independent of kernel version and configuration.

3.5. Google Container Engine - Kubernetes

Google Container Engine runs Kubernetes 1.8, supporting high availability, multi-master cluster which improves Service Level Objectives to 99.99% [36].

Kubernetes Engine runs on it's open source project called as Kubernetes which was previously Google's internal cluster management platform, then called" Borg" [37].

From Gmail to YouTube to Search, everything at Google runs in containers [37]. Google creates over 2 billion containers each week.

Features of Google Container Engine are:

1) Hardware Acceleration.
2) Node Auto repair.
3) Autoscaling clusters.
4) Metering and scaling.
5) Extensibility.
6) Node Auto upgrade.
7) Security and reliability.
8) Container Native Networking (Exclusive with Google Cloud Platform (GCP)).
9) Monitoring.

Benefits of using Kubernetes Engine:

1) Identity and access management.
2) Opensource portability.
3) Auto upgrade.
4) Security.
5) Hybrid Networking.
6) Autoscaling.
7) Auto repair.
8) Resource limits.
9) Private container registry.
10) Docker image support.
11) Straight full application support.
12) Monitoring and logging.
13) Fully managed.
14) Operating system built for containers.

3.6. Amazon ECS (Amazon Web Services Elastic Container Service)

Amazon Web Services has its propriety container management service called as Amazon Elastic Container Service (Amazon ECS) [38]. To interact with its container ecosystem, it provides native API for containers. It has capabilities of managing cluster environment to make Amazon ECS faster, scalable to operate Docker containers. For accessing Amazon ECS, there is no need for installing any software.

The following methods can be used to access Amazon EC:

1) AWS Management Console: A web-based interface for managing Amazon ECS.
2) AWS Command Line Tool (CLI): a faster and convenient tool than using management console.
3) Amazon ECS CLI: this interface comes under command line tool, used for entirely controlling Amazon ECS.
4) AWS SDK: Used for accessing Amazon ECS through various programming languages.

Features of Amazon ECS are:

a) AWS Fargate Support.
b) Docker Support.
c) Compatible with Windows Container.
d) Local development native support.
e) Cluster control.
    a. Scheduling.
    b. Task placement policy.
f) Network and security.
g) Load balancing.
h) Monitoring and logging.

The types of Container image supported:

a) Docker.
b) Kubernetes.
c) CoreOS.

4. CLOUD AND OPEN STACK

Virtualization techniques are used in the cloud to achieve maximum utilization of resources and scaling of shared resources. There are various versions for the definition of cloud computing given by different institution and organization. In cloud computing, resources are available to use over a network with minimal efforts in managing them.

Cloud provides following services based on its usage:

1) Infrastructure as a Service (IaaS).
2) Platform as a Service (PaaS).
3) Software as a Service (SaaS).

Table 2 gives a list of cloud providers along with their container service name where few of them provide more than one container service.

Table 2. Cloud Providers Along with Their Container Services

Cloud Provider	Container Service
Alibaba Cloud	Alibaba Cloud Container Service
Amazon Web Services	- Amazon Elastic Container Service (ECS) [38] - Amazon Fargate
Cloud Foundry	Warden and Garden [39]
Dell technologies	REX-Ray-A Storage Orchestrator
Google Cloud Platform	Google Kubernetes Engine
IBM Cloud	IBM Cloud Container Service
Microsoft Azure	Microsoft Azure Fabric Service
Red Hat OpenShift	Red Hat OpenShift Container Platform
Oracle Cloud Infrastructure	Oracle Cloud Infrastructure Container Service Classic
Rackspace	Carina [40]
OpenStack	- Magnum - Zun - Nova Docker Hyperviser - Kolla - Kata

4.1. OpenStack

OpenStack is free and open source cloud operating system that provides Infrastructure as a Service (IaaS) by virtualizing computational resources. OpenStack started as a project of Rackspace Hosting and NASA jointly in 2010. OpenStack is now managed by OpenStack foundation and supported by more than 500 companies since September 2012. Many of the leading software industries have extended their support in developing OpenStack.

OpenStack combines other open source projects (or tools) for providing cloud infrastructure. For providing core cloud computing resources, it uses six open source tools for computing, storage, networking, image storing, identification, orchestration. The services providing the tools are called Nova, Swift, Cinder, Neutron, Glance, Keystone, and Heat respectively. OpenStack uses Application Programming Interface (API) for interacting with resources. In 2014, the OpenStack community decided to support containers in OpenStack. OpenStack is capable of handling system containers like LXC (Linux Containers) and Virtuozzo. From Liberty release of OpenStack orchestration tools like Docker Swarm, Kubernetes and Apache Mesos are also included. OpenStack Containers Team formed in May 2014 is working towards the creation of new tools for handling container technology. Their aim and objective is to give user-friendly experience for managing and creating containers in OpenStack.

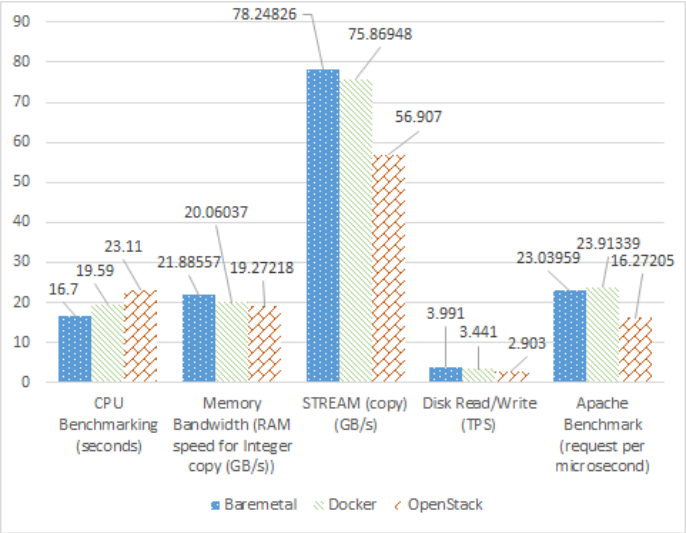

Figure 3. Comparison between baremetal, OpenStack, and Docker container

Figure 3 Jyoti Shetty et al. [41] provides the performance results obtained using Benchmarks on CPU, memory, data read/write and Apache benchmark. The results show that baremetal is a better option than Docker and OpenStack, but Docker outperforms baremetal in Apache benchmark. The reason behind the decline in performance of OpenStack is the presence of additional layers. Docker shows better results for Apache benchmark against baremetal. Docker outperforming baremetal is due to the lower complexity of Docker for providing the services. The results show that Docker is a better option than OpenStack, but it lacks capabilities of cloud computing. OpenStack carries characteristics of cloud that are not provided by Docker. By results, one can understand that there is enormous scope for improving the performance of OpenStack by integrating containers with them.

Projects for integrating containers in OpenStack:

4.1.1. OpenStack Kolla

It provides containers that are production ready along with the deployment tools required for managing team.

Sub-projects available for Kolla installation are:

1) Kolla-Ansible.
2) Kolla Kubernetes.

Kolla provides Docker containers and Ansible playbooks to deploy in OpenStack on a baremetal or virtual machine, and kubernetes templates to deploy kubernetes in OpenStack and meet Kolla's mission.

4.1.2. OpenStack Magnum

Magnum uses OpenStack API for providing container orchestration engines like Swarm, kubernetes and Apache Mesos [42]. It uses Heat service of OpenStack for orchestrating operating system Image of Docker and kubernetes to run in virtual machine or bare metal in cluster. Magnum provides the same level of abstraction similar to Nova running a virtual machine. Python magnum client uses two binaries, REST API server, and python conductor to run the process.

Python magnum client is horizontally scalable. Magnum uses Heat for orchestrating Docker and COE for automating cluster configuration.

The features of OpenStack Magnum are:

1) Cluster isolation-multitenancy in cluster.
2) Availability of Apache Mesos, Docker Swarm, k8s.
3) Keystone integration-choice of using VM or native.
4) Neutron integration.
5) Cinder integration.
6) The containers in cluster give less icon management using standard API.

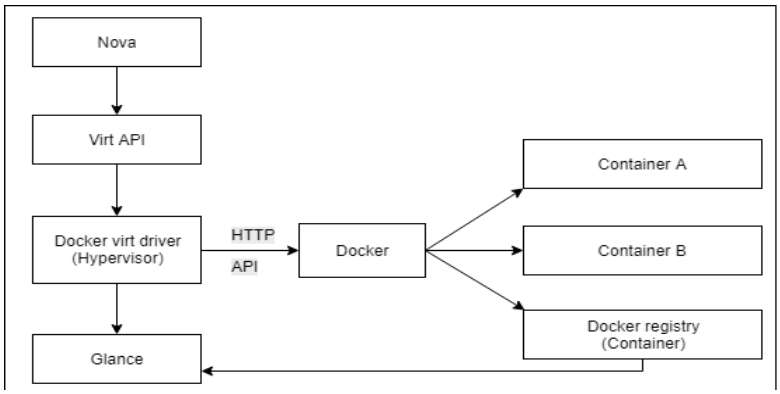

Figure 4. The architecture of Nova Docker hypervisor [43]

4.1.3. OpenStack Nova Docker hypervisor

After the launch of Havana, OpenStack has introduced a Docker driver which is hyper- visor driver in OpenStack Nova compute called OpenStack Nova Docker hypervisor [41]. Docker internal API interacts with the Nova's HTTP client using Unix socket. Figure 4 shows Docker is managed and controlled using HTTP API. All the Docker images get stored in image service of OpenStack called Glance. The containers run using the Docker virt driver in Nova by using virt API.

4.1.4. OpenStack Zun

Managing containers in OpenStack is done using Zun (ex. Higgins). It provides OpenStack API for deploying and managing containers using various container technologies. It provides the API for managing and controlling different containers in OpenStack [44].

Components of Zun:

1) Zun API: Deploying containers.
2) Zun Compute: Deploying and managing containers.
3) Keystone: Identity management.
4) Neutron: Networking.
5) Glance: Storing containers images.
6) Docker network plugin.

4.1.5. OpenStack Kata

It is an open source project which aims for combining security advantages of virtual machines with the speed and manageability of containers. This project will ensure that lightweight virtual machines will be built using features of VMs and containers. Features such as workload isolation and security from virtual machine and performance like containers are proposed to integrate into Kata. OpenStack Kata design makes it compatible with Open Container Initiative (OCI), and Container Runtime Interface(CRI) for kubernetes.

For initiating the project, Intel is giving their Intel Clear Container Technology and Hyper is giving runV technology.

5. CONCLUSION AND FUTURE WORK

In this paper, we discussed a lightweight alternative to a virtual machine called containers, popular among developers and administrators. Container-based virtualization is very efficient regarding performance and resource utilization (storage space and computational power).

We discussed types of containerization, and its advantages and also presented performance among them. When carrying out performance comparison between some of the container environments, the results show that native is always better than KVM, while Docker performs equal to native in some cases. Concerning network latency, KVM performs better than Docker, but not better than native as seen in Table 1.

For managing Linux containers, we use management tools called Container Orchestration Engine. Container Orchestration Engine helps in the smooth management of containers when dealing with a vast number of containers. COEs are chosen based on the application, features, and benefits provided by them. For example, our requirement is to deploy containers in a production environment, then we decide COE which is stable, offers enterprise-level support and minimal cost of usage. In a production environment, it is preferred to use proprietary COE due to the extent of support provided by them. Whereas, for the development environment, we prefer open source COE with continuous development.

The future work identified includes working on the scheduling of containers in the OpenStack cloud to achieve maximum possible computational performance. The performance of containers needs to be improved as seen from Figure 3 so that it outperforms baremetal machine as seen in Apache benchmark. This can be achieved by using dynamic resource provisioning to applications using containers [45].

ACKNOWLEDGEMENTS

This work was supported by the Centre for Development of Advanced Computing, Pune, India.

REFERENCES

[1] Kozhirbayev Zhanibek and O. Sinnott Richard. *A performance comparison of container-based technologies for the Cloud*. *Future Generation Computer Systems*, 68:175–182, March 2017.
[2] Fu Silvery, Liu Jiangchuan, Chu Xiaowen, and Hu Yueming. *Toward a Standard Interface for Cloud Providers: The Container as the Narrow Waist*. *IEEE Internet Computing*, 20(2):66–71, March 2016.
[3] B. Asgari, M. G. Arani, and S. Jabbehdari, "*An effiecient approach for resource auto-scaling in cloud environments*," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 6, no. 5, p. 2415, October 2016.
[4] S. Wu, L. Zhou, H. Sun, H. Jin, and X. Shi, "*Poris: A scheduler for parallel soft real-time applications in virtualized environments*," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 841–854, March 2016.
[5] W. Iqbal, M. N. Dailey, and D. Carrera, "*Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier web applications*," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1435–1446, December 2016.
[6] O. Adam, Y. C. Lee, and A. Y. Zomaya, "Stochastic resource provisioning for containerized multi-tier web services in clouds," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 7, pp. 2060–2073, July 2017.
[7] Calinciuc Alin, Constantin Spoiala Cristian, Octavian Turcu Corneliu, and Filote Constantin. *OpenStack and Docker: Building a high-performance IaaS platform for interactive social media applications*. In 2016 International Conference on Development and Application Systems (DAS). *IEEE*, May 2016.
[8] Kathy Cacciatore, Paul Czarkowski, Steven Dake, John Garbutt, Boyd Hemphill, John Jainschigg, Andre Moruga, Adrian Otto, Craig Peters, and Brian E Whitaker. *Exploring Opportunities: Containers and OpenStack*. OpenStack White Paper, 19, 2015.
[9] *Enabling Microservices: Containers & Orchestration Explained,* https://www.mongodb.com/ collateral/microservices-containers-and-orchestration-explained (accessed 24th March 2018).
[10] RATAN VIVEK. *Docker: A Favourite in the DevOps World,* http://opensourceforu.com/2017/02/ docker-favourite-devops-world/ (accessed 18th March 2018).
[11] *What's lxc?,* https://linuxcontainers.org/lxc/introduction/ (accessed 31st December 2017).
[12] Menage Paul. *Cgroups,* https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt (accessed 18th March 2018).
[13] Pahl Claus. *Containerization and the PaaS Cloud. IEEE Cloud Computing*, 2(3):24– 31, May 2015.
[14] *What's lxd?,* https://linuxcontainers.org/lxd/introduction/ (accessed 31st December 2017).
[15] *What's lxcfs?,* https://linuxcontainers.org/lxcfs/introduction/ (accessed 31st December 2017).

[16] Tosatto Andrea, Ruiu Pietro, and Attanasio Antonio. *Container-Based Orchestration in Cloud: State of the Art and Challenges*. In 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems. *IEEE*, July 2015.
[17] Singh Sachchidanand and Singh Nirmala. *Containers & Docker: Emerging roles & future of Cloud technology*. In 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). *IEEE*, 2016.
[18] Peinl René´, Holzschuher Florian, and Pfitzer Florian. *Docker Cluster Management for the Cloud - Survey Results and Own Solution. Journal of Grid Computing*, 14(2):265–282, April 2016.
[19] Dirk Merkel. Docker: *Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal*, 2014, March 2014.
[20] Polvi Alex. *CoreOS is building a container runtime, rkt,* https://coreos.com/blog/rocket.html (accessed 18th March 2018).
[21] *Rkt, a security-minded, standards-based container engine,* https://coreos.com/rkt/ (accessed 31st December 2017).
[22] Ali Babar M and Ben Ramsey. *Evaluating Docker for Secure and Scalable Private Cloud with Container Technologies*, 2017.
[23] Yamato Yoji. *OpenStack hypervisor, container and Baremetal servers performance comparison*. *IEICE Communications Express*, 4(7):228–232, 2015.
[24] Felter Wes, Ferreira Alexandre, Rajamony Ram, and Rubio Juan. *An updated performance comparison of virtual machines and Linux containers*. In 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). *IEEE*, March 2015.
[25] Naik Nitin. *Building a virtual system of systems using docker swarm in multiple clouds*. In 2016 IEEE International Symposium on Systems Engineering (ISSE). *IEEE*, October 2016.
[26] *Container Management: Kubernetes vs Docker Swarm, Mesos + Marathon, Amazon ECS.* Platform9, 2017.
[27] *What is kubernetes?,* https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (accessed 18th March 2018).
[28] Burns Brendan, Grant Brian, Oppenheimer David, Brewer Eric, and Wilkes John. *Borg, omega, and kubernetes. ACM Queue*, 14:70–93, 2016.
[29] Bernstein David. *Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing.* 1(3):81–84, September 2014.
[30] DelValle Renan, Rattihalli Gourav, Beltre Angel, Govindaraju Madhusudhan, and J. Lewis Michael. *Exploring the Design Space for Optimizations with Apache Aurora and Mesos*. In 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). *IEEE*, June 2016.
[31] *Overview of azure service fabric,* https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview (accessed 21st January2018).
[32] *Service fabric programming model overview,* https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-choose-framework (accessed 21st January 2018).
[33] *Asp.net core in service fabric reliable services,* https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-communication-aspnetcore (accessed 21st January 2018).
[34] Russinovich Mark. *Announcing azure service fabric: Reducing complexity in a hyper-scale world,* https://azure.microsoft.com/en-gb/blog/announcing-azure-service-fabric-reducing-complexity-in-a-hyper-scale-world/ (accessed 21st January 2018).
[35] *Windows containers,* https://docs.microsoft.com/en-us/virtualization/windowscontainers/ about/ (accessed 21st January 2018).
[36] Paik Dan. *Google container engine - kubernetes 1.8 takes advantage of the cloud built for containers,* https://cloudplatform.googleblog.com/2017/09/google-container-engine-kubernetes-18.html (accessed 30th January 2018).
[37] *Containers at google, a better way to develop and deploy applications,* https://cloud.google.com/containers/ (accessed 21st January 2018).
[38] *Amazon container services,* https://aws.amazon.com/containers/ (accessed 21st January 2018).
[39] Zhylinski Maksim. *Cloud foundry containers: The difference between warden, docker, and garden,* 2016.
[40] Weaver Christina. *Carina™ by rackspace simplifies containers with easy-to-use, instant-on native container environment,* 2015.
[41] Shetty Jyoti, Sahana Upadhaya, Rajarajeshwari HS, Shobha G, and Chandra Jayant. *An Empirical Performance Evaluation of Docker Container, Openstack Virtual Machine and Bare Metal Server. Indonesian Journal of Electrical Engineering and Computer Science*, vol. 7, no. 1, pp. 205-213, 2017.
[42] Clark Tim. *Containers and openstack creating a platform for distributed applications.* A FactPoint Group white paper sponsored by Red Hat Inc., 2016.
[43] *Docker,* https://wiki.openstack.org/wiki/Dockerhttps://wiki.openstack.org/wiki/Docker (accessed 21st January 2018).
[44] *Zun,* https://wiki.openstack.org/wiki/Zun (accessed 31st January 2018).
[45] S. Suresh and L. M. Rao, "*Cccore: Cloud container for collaborative research,*" *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 3, pp. 1659–1670