Quality of Service for Containers in Cloud Computing

November 2016

By Nathan Kong

For Professor Melody Moh

Cloud Computing

Abstract

As clouds become the central hub for various commercial applications, the question of providing cloud services at a fast rate becomes a major concern. Running cloud applications with Docker containers provides much needed interoperability and isolation while reducing the overheads associated with virtual machines. Docker containers do have its issues. Performance degradation can occur due to the lack of Quality of Service. Currently Docker runs containers with a best effort policy. Quality of Service in containers can be improved through management in both containers and clusters of containers.

Table of Contents

Acronyms

CaaS	Container-as-a-Service
OS	Operating System
PaaS	Platform-as-a-Service
QoS	Quality of Service
VM	Virtual Machine

1 INTRODUCTION

Cloud computing provides on demand and pay-per-use access to a pool of shared compute resources. These resources include memory, storage, networking, and applications. The popularity of cloud computing has created a rising demand for cloud services. This quick rise has created issues and challenges for cloud service providers. They are faced with issue including scheduling, resource allocation, security, and interoperability. The Platform-as-a-Service (PaaS) model needs to create and deliver applications that are lightweight, portable, and provide cross-platform compatibility.

Virtualization provides partitioning, encapsulation, and isolation. It is ideal for addressing the challenges of cloud computing. Virtualization provides elasticity through hardware resource allocation and management. It provides isolation and can be achieved through both VMs and containers.

Containers can be used to eliminate overheads incorporated in virtual machines (VM). They consume less memory and have faster startup times. They do not depend on virtualized Operating Systems (OS). Containers provide ready to deploy packages consisting of the application, along with the libraries, binaries, middleware and other dependencies required for running an application. These advantages make containers ideal for cloud computing.

As majestic as containers sound, they are not a mature technology. They do not adequately address the Quality of Service (QoS) issue. Containers address QoS through a best effort policy. This policy can incur resource contention to the extent of application starvation. QoS in containers can be solved through management of containers at both the node and cluster level of container architecture.

The remaining report is organized as follows: Section 2 will provide a background for virtualization, Section 3 will provide a background on containers in cloud computing, Section 4 will present Quality of Service for containers, Section 5 will provide an enhanced QoS method, and finally we will conclude this paper in Section 6.

2 VIRTUALIZATION

Virtualization began in the 1960s has always been about dividing resources. It creates software based representation of applications or devices to boost efficiency. Virtualization creates portability of applications and services, separation from hardware, efficient use of resources, and recovery.

Portability from software virtualization creates fast deployment and snapshoting. With virtualization, a snapshot of software can be created and deploying to new machines. This allows developers to build and deploy at a faster rate. This also allows for controlling software through version control.

Virtualization separates software from the hardware. The separated software can be used across multiple types of hardware and the software can be migrated through various physical hosts.

With virtualization, entire OSs can be backed up to allow for recovery. Virtualized disk images can be stored on centralized storage to improve performance and availability. Disk images can be maintained in a version control for backup and rollback recovery.

2.1 VIRTUAL MACHINES

A Virtual Machine (VM) is a type of virtualization. It is an emulation of a computer system based on computer architecture. It provides the same functionality of a physical computer. During setup VMs require logical boundaries for memory. This enables a more secure virtual space. VMs require a hypervisor that talks to the OS as if the VM and host machine are separate entities. VMs are more mature than containers. They are highly developed with mature management systems and have proven to run in critical business workloads.

There are two main types of VMs. The first is system virtualization and the second is process virtualization. System virtualization is also known as full virtualization and consists of virtualizing a physical machine. They incorporate a hypervisor to share and manage hardware with the physical machine while keeping isolated. Process virtualization is designed to execute computer applications on a platform environment.

2.2 CONTAINERS

Virtualization has stepped into the next evolution with containers. Containers encapsulate applications with the minimal resources to accomplish their task. The concept of containers is not new, however it has been adopted to progress the future of software.

Containerization is virtualization at the OS level. They are lightweight and designed to run on any hardware that supports a Linux OS. Containers require less memory space and give each application its own isolated environment to run. They share the host server's OS, which creates faster loading time compared to VMs.

A container holds self-contained applications and, if necessary, middleware, as well as business logic to deploy applications [1]. Containers are OS virtualization techniques based on namespaces and cgroups.

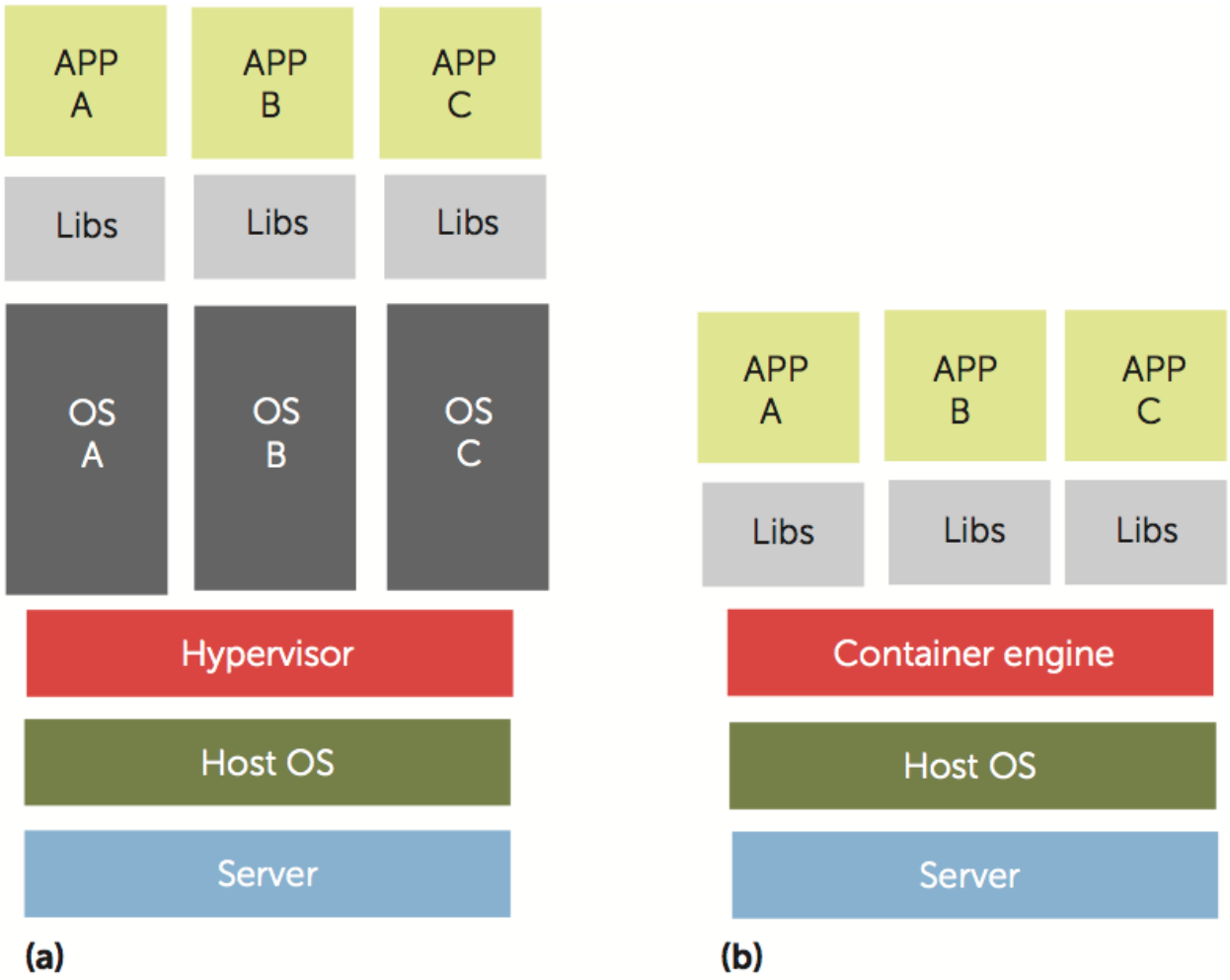

Figure 1: Deployments (a) VM based (b) container based [2]

Virtualization deployment is different between VMs and containers. Figure 1 illustrates a comparison between the two. Figure 1a depicts a VM based deployment. Applications run on virtual OS located on the hypervisor. This deployment is ideal for situations when applications require different OS on the same cloud. It also shows the abstraction is on the VM level.

Figure 1b shows the container based deployment. Applications run in the Container engine that sits in the host OS. This architecture allows less overhead that translates into smaller applications with faster start times. In addition, without the virtual OSs containers will only take the size of the container and not incorporate unused memory.

2.2.1 LINUX CONTAINERS

Linux Containers are often considered as something in the middle between chroot and a VM. It is a Linux OS level virtualization. This concept was developed to run multiple isolated Linux systems on a single host. To make this possible, Linux Containers layer some userspace tooling on top of cgroups and namespaces [2]. Linux Containers are able to run multiple processes in one container. They are stateful to support persistent storage.

2.2.2 DOCKER CONTAINERS

Docker containerization advances virtualization to the next level. It adds value to systems for continuous integration and continuous deployment as well as development and operations [3].

Docker encourages continuous integration and deployment by integrating version control. Docker allows developers to collaboratively build and test code in any environment and at any point of the application development lifecycle. Docker uses integrate with tools like Jenkins and GitHub to streamline development and deployment.

Development to operations has caused loss of time and team momentum due to environmental issues. An industry wide shift has been made in order to streamline this process. Docker removes this barrier. Docker containers will work in any environment that incorporates and allows access to the Linux kernel. In addition, Docker provides the ability to secure and manage its environment.

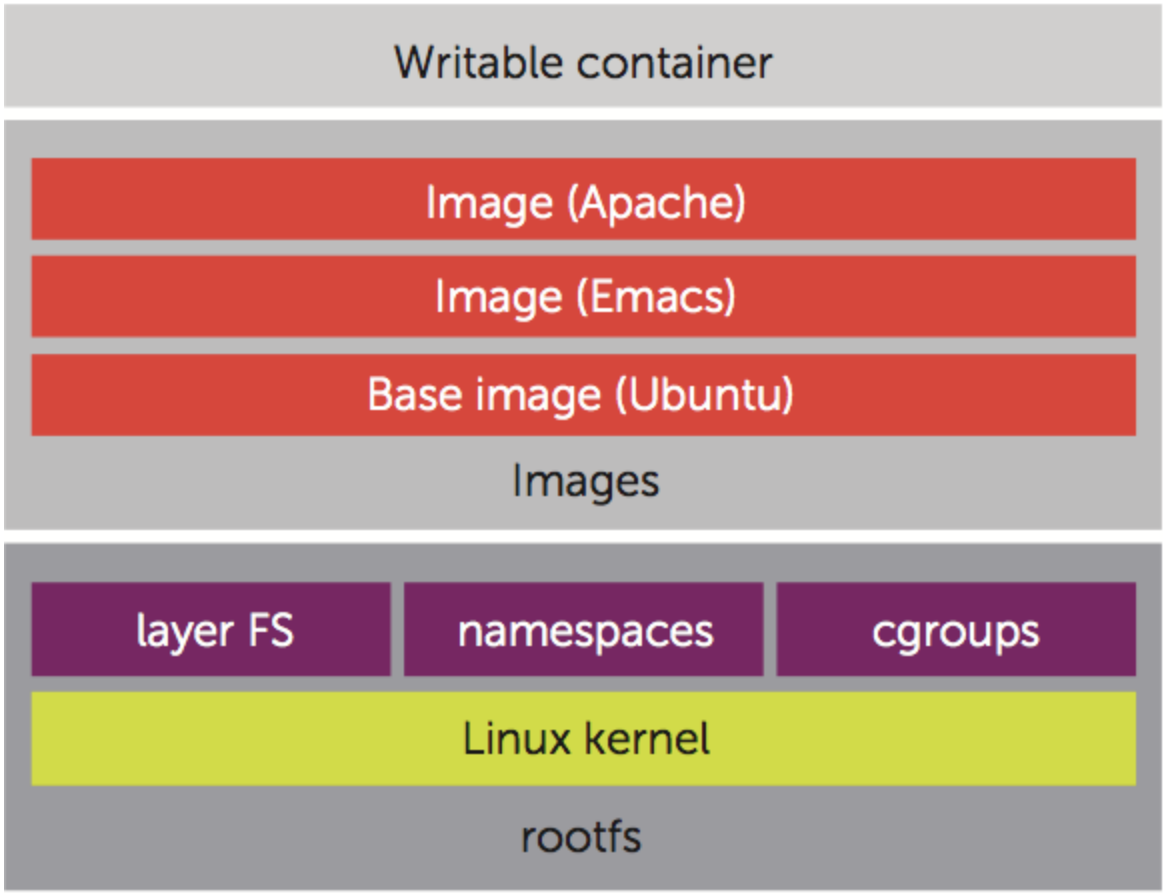

Figure 2: Container image architecture [1]

Docker is a tool to easily create, deploy, and run applications through containers. It is a framework built around Linux Containers. Docker extends Linux Containers with a kernel and application level Application Program Interface that runs processes in isolation. Docker uses namespaces to isolate an application's view of the underlying OS [2]. Docker creates containers through images. Docker images consist of file systems that layer over each other. The images are built in containers, which deploy applications.

When creating containers, Docker uses the kernel to mount the root file system as read-only. It then checks the integrity before using a union mount to add a writeable file system on top of the read-only system [1]. By using this structure, multiple files systems are allowed to be stacked on top of each other. New images can be created on top of base images with only the top layer being writeable. A typical layering is shown in Figure 2. Docker containers are based on layers of individual images built on an extendable base image. Docker uses this lightweight approach to easily change and distribute single images.

Containers for applications can be developed from scratch or base images on Docker's repository. The repository can push and pull various Docker images for version control. The repository also provides access to reusable private and public Docker images. Examples of images include: apache, MongoDB, Node.js, and Ubuntu.

2.2.3 Service Orchestration

Containers can be used to create distributed application by breaking apart applications into independent tasks. For example there can be a separate container for a webserver, application server, message queue, and backend workers [4]. This distributed architecture is known as microservice architecture. A microservice is an application that serves a single function. A microservice may run in a container, yet it could also run as a fully provisioned VM. The concept of microservice architecture is evolved from service-oriented architecture in web services.

The microservice architecture incorporates the development of multiple single applications into a suite of small services. Each service runs in its own container and communicates with other containers using lightweight mechanisms. Each container is independent and can automatically be deployed by Docker.

2.2.4 Container Clustering

Containerization incorporates clusters of grouped hosts. Figure 3 illustrates an example scenario. This example depicts a cluster that assembles host nodes with containers and data volumes. The host nodes containers are connected through links.

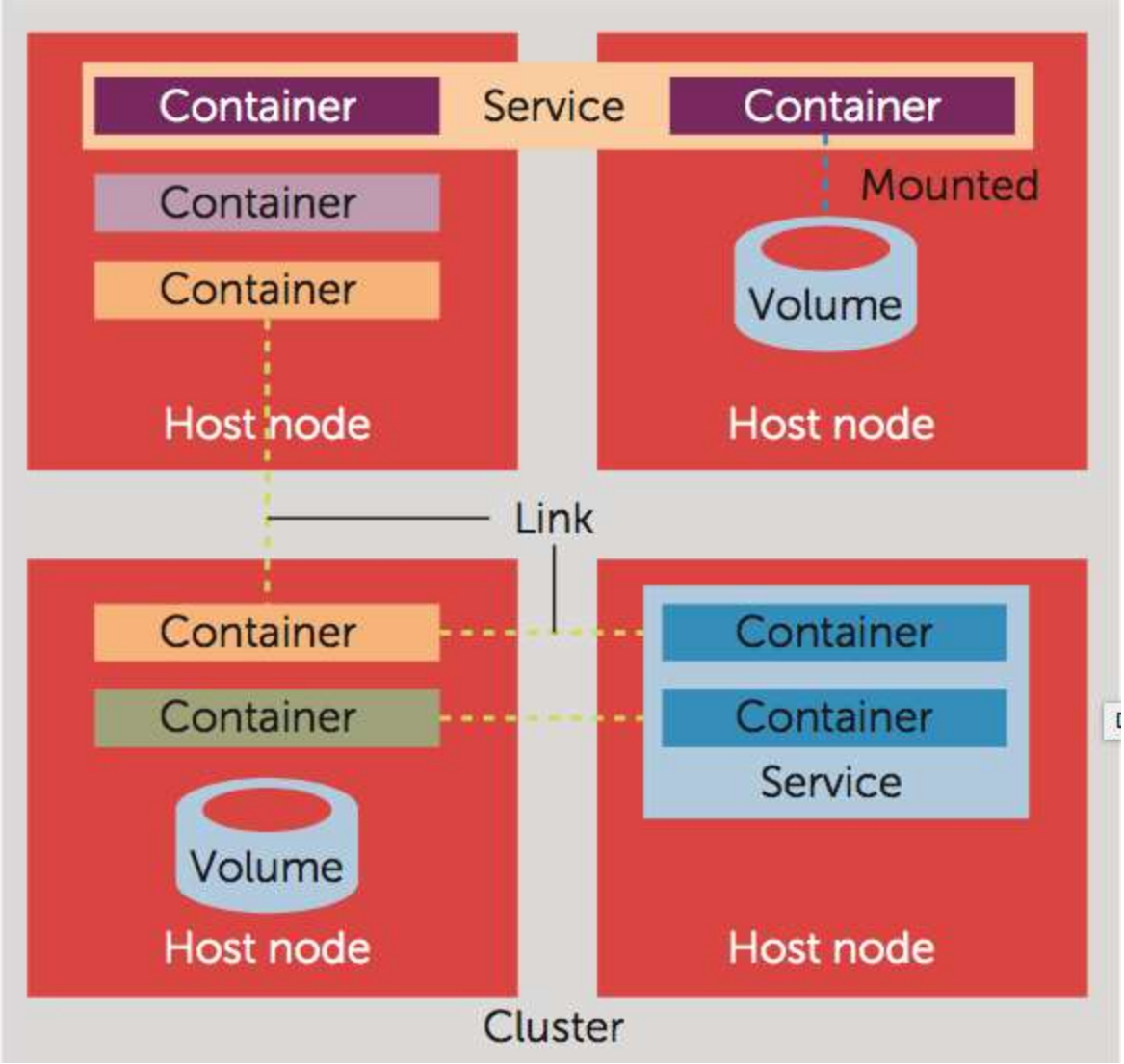

Figure 3: Container-based cluster architecture [1]

Container hosts are linked in a cluster configuration. Hosts nodes are typically virtual servers on hypervisors. Each host can hold multiple containers with common services. These services include scheduling, load balancing, and applications. Application services are logical groups of containers from the same image. Host nodes can contain volumes, which hold data. Containers can mount volumes to access data. Volumes will continue you hold data after containers are removed.

3 CONTAINERS IN CLOUD COMPUTING

In order to keep resources shared, the cloud relies on virtualization. Until recently, VMs have been the primary backbone of this infrastructure. Containers are a new exciting frontier that can replace VMs. They are lightweight and a faster solution for virtualization. Container are tools for delivering software, thus are naturally situated for a PaaS focus.

3.1 CONTAINERS FOR PLATFORM-AS-A-SERVICE

VMs are optimal to provision a PaaS platform at the infrastructure layer. Containers are geared for application packaging and management of a PaaS.

PaaS provides features for deploying applications, designing applications, pushing applications, using services, migrating databases, mapping domains, and building integration tools. To accomplish this PaaS has farms, routing layers and schedulers that dispatch workloads to VMs [1].

Containerization supports a solution to all of these issues through interoperable, lightweight, and virtualized packaging. Interoperability is accomplished by the process of how containers are built, deployed, and managed.

3.2 PLATFORM-AS-A-SERVICE EVOLUTION

Containers in PaaS have recently reached its next step in its evolution. The first generation consisted of containers in a fixed proprietary platform. PaaS containers were used in Microsoft Azure and Heroku. The second generation was built on open source solutions. The solutions include Cloud Foundry and OpenShift. The second generation lets users run their own PaaS built around containers. The third generation is built on Docker from scratch and can be deployed on company servers or public Infrastructure-as-a-Service clouds. These third generations platforms include: Dawn, Deis, Flynn, Octohost, and Tsuru [1].

3.3 CONTAINER-AS-A-SERVICE

Cloud computing has been increasingly adopted by businesses, industries, and governments. As technology in cloud computing has matured, a new type of service has appeared. This service is known as Container-as-a-Service (CaaS). CaaS can lies between Infrastructure-as-a-Service and PaaS. It glues the two layers together with

isolated environments [5]. In industry, CaaS can be found on VMs (Figure 4). This practice is used for security purposes. Providers such as AWS and Google claim that VMs are necessary to provide security for untrusted workloads while containers offer security for semi-trusted workloads.

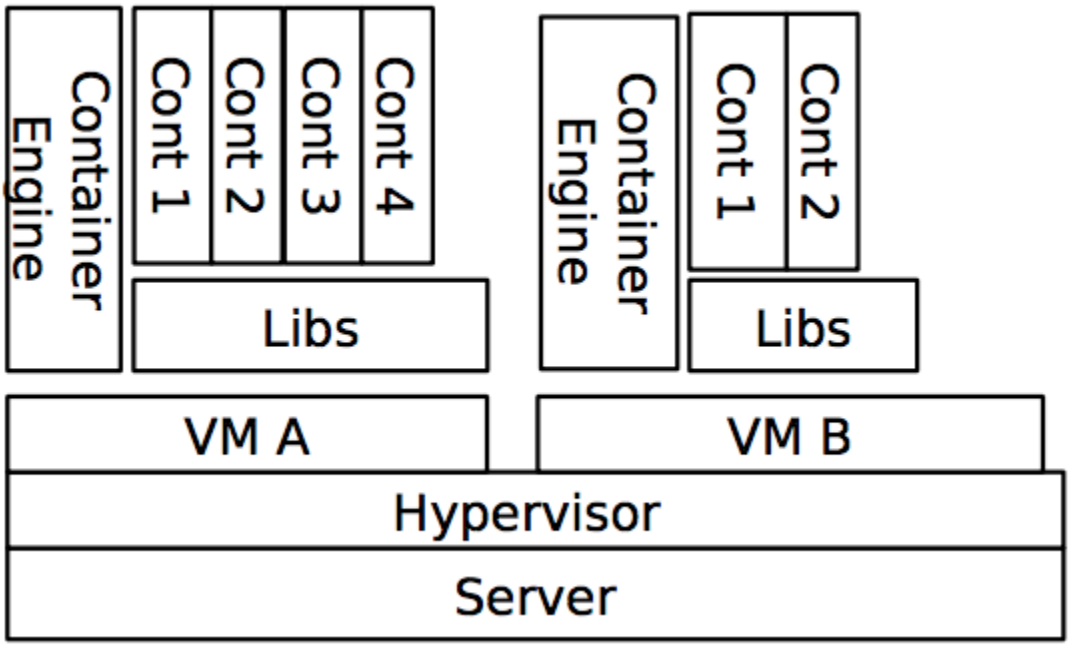

Figure 4: Containers in a virtual machine [5]

The CaaS architecture benefits from multiple service layers for efficient management. These service layers include: Workload management, container lifecycle, VM lifecycle, Resource management, resource allocation, power and energy monitoring, and data center management [5]. Workload management is a service for taking care of container deployment, scheduling, application level performance, and health monitoring. Container lifecycle management is a service for creating, starting, stopping, restarting, migrating, and destroying containers. VM lifecycle management is a service for VM creation, starting, stopping, destroying, and migrating VMs. It also offers resource utilization monitoring. Resource management is a service that ensures that virtualization is satisfied by its resource requirements. It ensures containers are layered on VMs according to its container allocation policy. It ensures VMs are layered on hosts according to its VM allocation policy. It reduces resource fragmentation by reducing container to the least amount of hosts. Resource allocation is a service that contains policies to determine how VM or host resources are scheduled to containers or VMs.

Power and energy monitoring is a service that is responsible for measuring the power consumption of hosts in a data center. Data center management is a service for powering on and off hosts and monitoring resources.

4 QUALITY OF SERVICE IN CONTAINERS

Google cloud provides a plethora of features to its cloud users such as the programmable cloud Domain Name System. This system runs on the same infrastructure as Google and provides low latency high-availability connections, with direct peering. Google uses a combination of Software Defined Networking and Network Function Virtualization to reduce the latency and improve the QoS for the applications that they host and deliver.

At the base of Google's cloud infrastructure, there are stacks of compute resources. Built on top of this, is the virtual network overlay consisting of a cluster of routers, switches, NICs and packet processors. These packet processors are responsible for providing QoS and configuring the network capabilities according to the users' requirements. Packet processors are also responsible for load balancing and Network Function Virtualization capabilities. On top of this lies a virtualization platform called Andromeda. Andromeda consists of a centralized SDN controller that allows all the resources to be viewed and provisioned. It allows customers to configure and provision resources according to their requirements [6].

AT&T also provides several cloud-based services such as NetBond. NetBond is a content delivery network and storage. IBM uses AT&T's NetBond to provide cloud managed services through a hybrid cloud computing infrastructure [7].

AT&T relies on a flexible SDN to ensure QoS while providing cloud services. They created the concept of Cloud QoS. This concept intelligently manages the QoS according to the changing requirements of the customers. NetBond uses an infrastructure that integrates the IBM cloud computing with the AT&T's cloud network. This uses an extension of customers' Multiple Protocol Label Switching into a Virtual Private Network, thus creating a private network connection and ensuring scalability, security and higher-bandwidth [8-9].

Containers have gained popularity in cloud computing. Commercially they have been adopted for business and technology. Through expansion of containers in cloud computing new issues have arose. One such issue is QoS for containers.

4.1 NEED FOR QOS MECHANISMS FOR CONTAINERS IN CLOUD

The current configuration in a CaaS is set to provide a "best effort" to all traffic. This does not guarantee reliability, bandwidth, or speed for a specific application.

The containerized applications run in complete isolation from each other through independent containers. This concept does not limit the applications sizes running within the container. Applications will content for limited and shared resources within the container or host. This leads to severe performance loss and possibly application starvation. While solutions exist to mitigate memory, processing, and network contention, not much has been accomplished to addresses the issue of application prioritization in containers.

Containers are configured within the same subnet in order to maintain connectivity. By default, all the containers get an equal share of bandwidth. QoS is an important factor in containerization. Critical applications in containers require favoritism for higher performance. Containers therefore must provide the ability to configure higher bandwidth, priority, and network resources to specific containers.

4.2 Current Quality of Service Mechanisms for Containers

As CaaS has evolved, it has become apparent that QoS for container needs to evolve as well. Companies such as Google, Docker, and Apache have developed cluster level management technologies to combat the issue containers best effort QoS. These companies each have developed resource management and scheduling software. Their software's are called Apache Mesos, Docker Swarm, and Google Kubernetes. Their software runs on top of Docker and manages a cluster level.

4.3 A Two-Tiered Approach to Quality of Service in Containers

One solution for the issue of QoS in containerization is to take a two-tiered approach [10]. In cloud computing, multiple containers are arranged on a node. A group of nodes is considered a cluster. Providing QoS mechanisms for only containers or clusters is not an efficient option. If only a node level solution is adopted, resource imbalance and possible contention at the cluster level will occur. Similarly, if only a cluster level solution is adopted, it will not guarantee the QoS at individual nodes. Multiple containers within the node may try to access the same resources at the same time. Therefore a two-tiered solution is necessary. A QoS mechanism must be achieved at both the node level in addition to the cluster level.

QoS mechanisms can be introduced to cloud computing through Docker. Node level QoS can be implemented through management of the input and output of Docker

containers. At the cluster level, QoS can be accomplished through managing the input and output of the entire cluster.

4.3.1 Node Level QoS

In order to guarantee QoS for individual nodes, constraints on containers must be enforced. This will reduce over-allocation of resources at node level. The Blkio cgroup can be used to accomplish this goal by extending Docker to include the ability to control the input and output of containers [10].

A client-side Application Program Interface can be incorporated to allow controlling of read and writes from containers. It will set general priority levels on containers. It will allow users to manage the number of bytes required for a container.

4.3.2 Cluster Level Quality of Service

At cluster level, QoS must be maintained across the entire cluster. This can be accomplished through a manager that monitors and adjusts all the containers running on cluster. Docker Swarm has been developed for just this reason. It is as a native clustering tool that allows users to interact with the cluster as if it were a single machine. Swarm includes capability of scheduling and monitoring a cluster.

5 QoS for Containers

Cloud computing has become a necessity for consumers and businesses because of its flexibility, disaster recovery, on-demand and pay-for-use access, and shared computing resources. Within the past few years, cloud computing has adopted Docker containers for virtualization through a microservices architecture. Containerization in cloud computing is a new concept and has some growing pains to go through. One issue is QoS between containers. Docker containers allow control of computing and memory, yet only provides a "best effort" for traffic delivery. This delivery scheme does not provide any guarantees.

Quality of Service for containers is an active issue being researched. Quality of Service is commercially being addressed for pods of containers through applications such as Google's Kubernetes [10]. At the individual container level, only academia has completed solutions.

5.1 Background and Implementation

One solution that has arisen in academia is to run containers on a priority basis. Paper [11] presents a scheme in which time-sensitive and critical applications get a higher priority for network bandwidth. This is accomplished through three steps (Figure 5). For step 1, users request the use of an application. For step 2, the requests flow to a classifier. The classifier decides the priority of the flow and assigns the flow to a priority queue. For step 3, a scheduler picks up the queues and assigns the containers when to run. Highest priority containers will run first and lowest priority containers will run last.

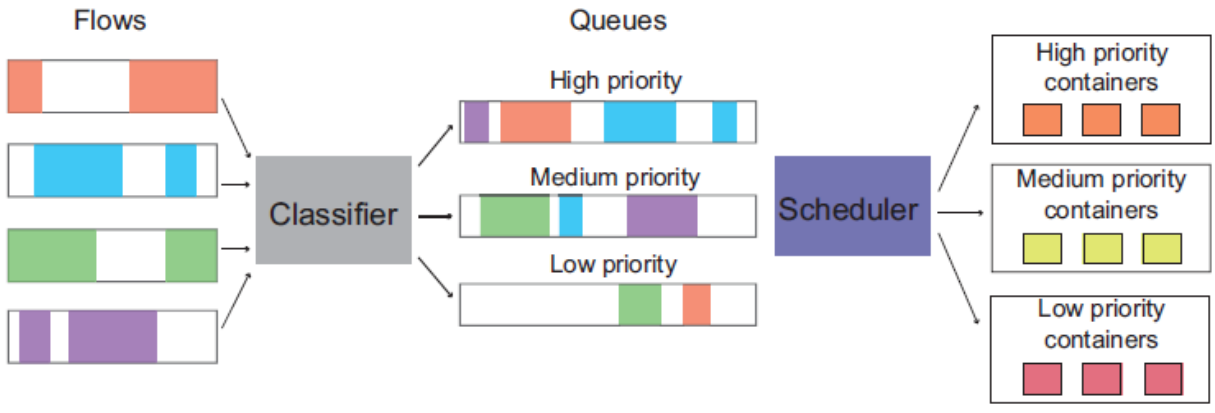

Figure 5: Architecture of priority queue scheduler [11]

The architecture from paper [11] has proved a QoS for containers based on bandwidth. While this provides a level of service, this paper believes that higher priority containers may starve lower priority containers. To eliminate starvation among the containers, an enhanced round robin scheduling has been added to the scheduler. For this project the scheduler will run highest priority containers first 50% of the time. Medium priority containers will run first 33% of the time. Low priority containers will run first 17% of the time.

This concept was tested using a simple java program. The program was built after the evaluation of the cloud simulator CloudSim. The cloud architecture was simulated based on CloudSim's Container-as-a-Service architecture. This architecture consists of hosts, virtual machines, and containers. While virtual machines are not necessary in cloud computing they were kept in the simulation as real clouds use them for additional security [5].

The architecture for this test consisted of three containers running in one virtual machine located on one host. The virtual machine was restricted to a bandwidth of 3000 bits per second. Each container maintained a priority of high, medium, or low. Container one had a priority of high, container two had a priority of medium, and container three had a priority of low. A total of seventy five requests were distributed evenly among the containers. Each request ran for 5 seconds at a different bandwidth. The bandwidth per request was randomly created between 500 to 2500 bits per second. Each container can only run one request at a time.

For the enhanced solution, the scheduler ran the requests in one of three variations: Priority 1, priority 2, priority 3; Priority 2, priority 3, priority 1; Priority 3, priority 1, priority 2. These variations were run in a sequence. The sequence was created giving the higher priority one more opportunity than the priority below it. Once the sequence was completed it was repeated until all requests were completed. The sequence was as follows: variation one, variation one, variation two, variation one, variation two, and variation three.

5.2 Performance Evaluation

The simulation was completed for both paper [11] and the enhanced concept. Figure 6 shows the results for paper [11] while Figure 7 shows the results for an enhanced round robin solution.

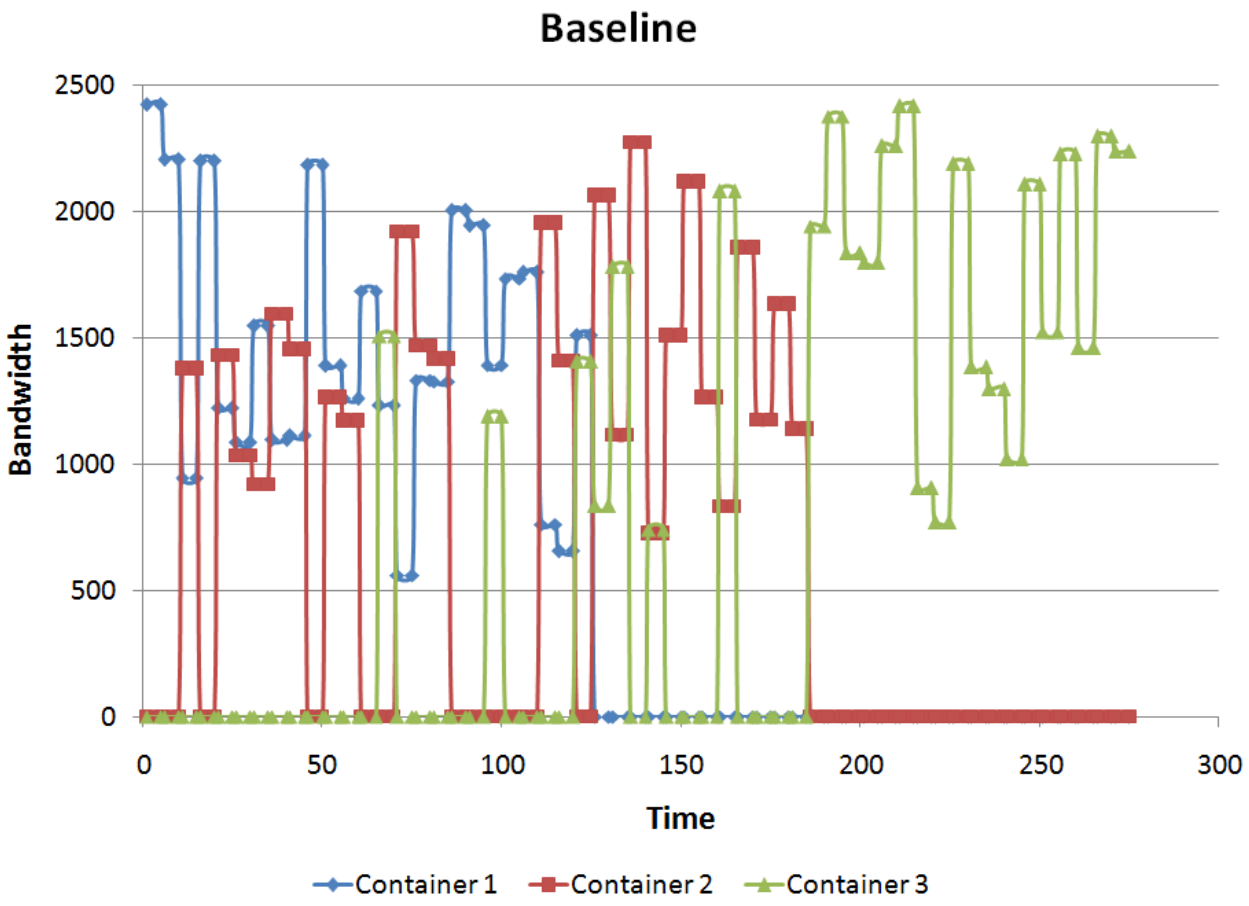

Figure 6: Baseline Results

Figure 6 shows the high priority container running requests first. The high priority container ran from the beginning to 125 seconds. When additional bandwidth was available the medium priority container ran requests as well. Medium priority containers ran from 11 to 185 seconds. The low priority container ran after most of its requests after all the high priority requests were completed. Low priority containers ran from 66 to 275 seconds.

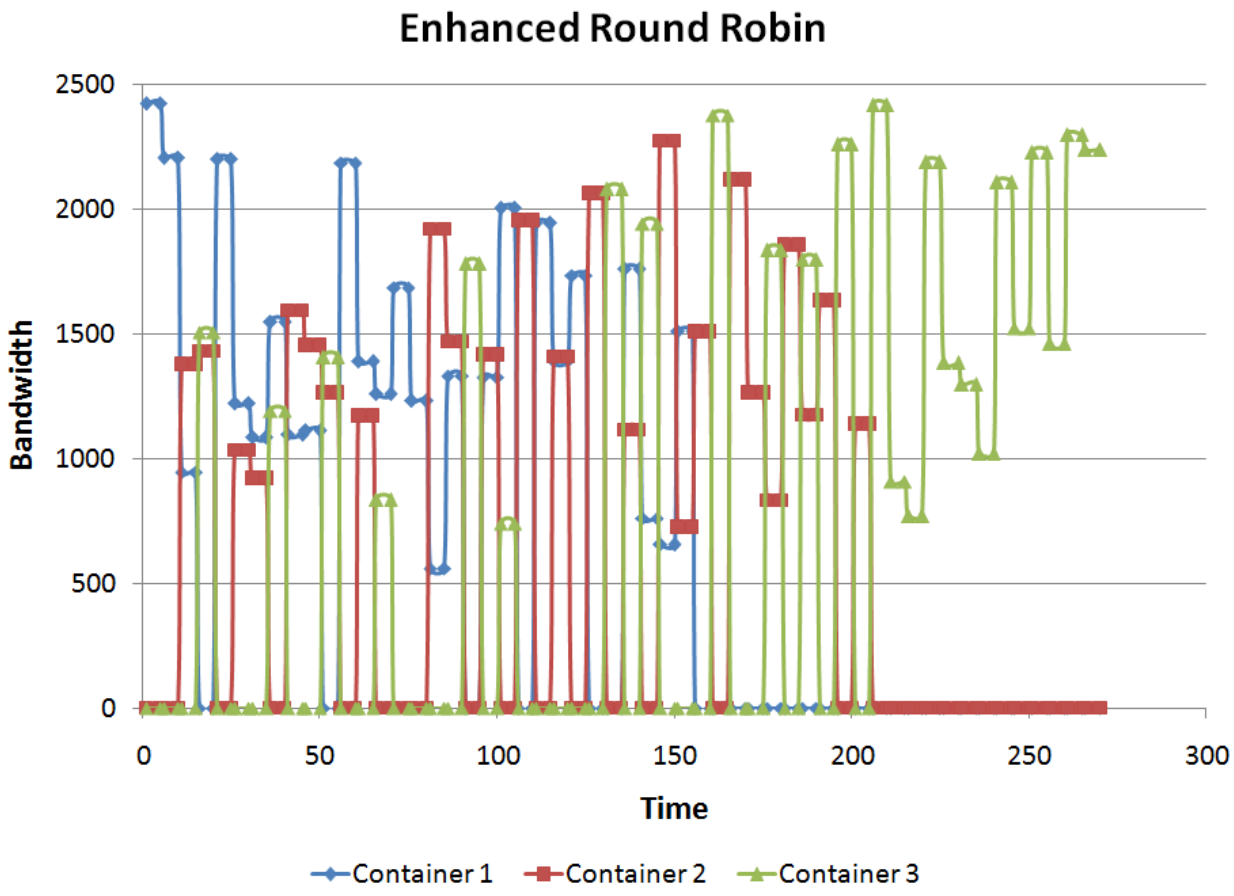

Figure 7: Enhanced Round Robin Results

Figure 7 shows a better dispersion of requests being fulfilled. High priority requests started at the beginning and ended at 155 seconds. Medium priority containers ran from 11 to 205 seconds. Low priority containers ran from 16 to 270 seconds. High priority requests were fulfilled at a slower rate when compared with the baseline, however these requests were still given priority to complete.

6 CONCLUSION

The increased use of containers in cloud computing for commercial industries has expanded. Containers have become an efficient way of encapsulating and implementing the cloud services and applications. Containerization provides portable, self-contained, light-weight applications, while reducing the overhead associated with memory and time consumption in VMs. This rise of containers has shown QoS issues between containers. In order to address this issue, a two-tiered approach can be employed at both the node level and the cluster level.

Our concept of an enhanced round robin scheduling proved QoS to containers based on priority. It prevented starvation of bandwidth due to priority. Providing QoS to containers enables time-sensitive and critical applications to get priority over other containers. This priority assists to deliver the proper network bandwidth for each container.

REFERENCES

[1] Claus Pahl, “Containerization and the PaaS cloud,” *IEEE Cloud Computing*, vol 2, no. 3, pp. 24-31, May 2015.

[2] David Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes”, *IEEE Cloud Computing*, vol 1, no 3, Sept. 2014

[3] Docker (2016). Docker Use Cases [Online]. Available: https://www.docker.com/use-cases

[4] AWS (2016). Container Use Cases [Online]. Available: https://aws.amazon.com/containers/use-cases/

[5] Sareh Fotuhi Piraghaj et al, “ContainerCloudSim: A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers,” in *An Environment for Modeling and Simulation of Containers in Cloud Data Centers*, 1st ed. USA, John Wiley & Sons Ltd, June 2016, DOI: 10.1002/spe.

[6] Rivka Gewirtz Little (2014). With Google SDN, Scalable Cloud Infrastructure as a Service [Online]. Available: http://searchsdn.techtarget.com/news/2240215974/With-Google-SDN-scalable-cloud-Infrastructure-as-a-Service

[7] AT&T (2016). Cloud QoS [Online]. Available: http://about.att.com/innovation/labs/cloudqos

[8] AT&T (2016). IBM Cloud Managed Service with AT&T NetBond [Online]. Available: https://www.synaptic.att.com/clouduser/html/productdetail/Network_Enabled_Cloud_Computing.htm

[9] AT&T (2016). AT&T NetBond [Online]. Available: https://www.synaptic.att.com/clouduser/html/productdetail/ATT_NetBond.htm

[10] Sean McDaniel et al, “A Two-Tiered Approach to I/O Quality of Service in Docker Containers,” in *IEEE Int. Conf. on Cluster Computing*, Chicago, IL, 2015, pp. 490-491.

[11] Ayush Dusia et al, “Network Quality of Service in Docker Containers,” in *IEEE Int. Conf. on Cluster Computing*, Chicago, IL, 2015, pp. 527-528.