TUTA/IOE/PCU

Journal of the Institute of Engineering, 2016, ***12(1):*** *51-62*
© TUTA/IOE/PCU
Printed in Nepal

CoMMoN: The Real-Time Container and Migration Monitoring as a Service in the Cloud

[1]Babu Ram Dawadi, [2]Subarna Shakya, [3]Rajendra Paudyal
[1,2]Department of Electronics and Computer Engineering, Pulchowk Campus, Tribhuvan University, Nepal
[3]Engineering Department, Nepal Telecom, Kathmandu, Nepal
Corresponding author: [1]baburd, [2]drss@ioe.edu.np,[3]rajendra.paudyal@ntc.net.np

Received: May 16, 2016 Revised: July 9, 2016 Accepted: July 15, 2016

Abstract: With the advancement of computing technologies, the modern cloud computing and virtualization system is highly distributed by the invention of better and light weight distributed applications packaging toolkit over the cloud environment, called the container technology. Cloud containers feature the management of applications with easy plug and play ability, migration, replication, relocation, upgrading et cetera in the real time. Such containers running different applications over the cloud infrastructure may consume different resources that require real time monitoring. Docker is an open source platform independent tool to create, deploy, and run applications by using Containers. Billions of applications for the end users and SMEs running over the cloud require efficient management, monitoring and operations to achieve better SLAs for cloud service providers. With this research, a Docker Container and Migration Monitoring System (CoMMoN) has been developed in which a customer running its application in a Container shall monitor its application in the real-time as well as take decision for the migration of Container to another network by service provider for liad balancing or by the customer under the violation of SLA. Once the customer registers its Container to the monitoring system, the monitoring probe collects different monitoring metrics and sends those parameters to remote monitoring system. The system which stores the monitoring metrics into InFluxDB, a time series database, also monitors the real-time migration of the Containers among the network.

Keywords: Cloud Service Provider, Docker, Container, Monitoring, Migration

1. Introduction

Cloud computing leverages the presence of enterprises with powerful data centers that provides resources on demand: the provision mechanism relies on virtualization. Generally services provided by the cloud are divided into three: IaaS, PaaS and SaaS. But for the efficient, reliable and elastic services, monitoring plays vital role to achieve better Service Level Agreement (SLA). Hence monitoring is becoming one of the cloud services as MaaS. In our previous paper [14], we conceptualized the real-time Container migration and monitoring by implementing the system with JAVA and MySQL with JSP for web visualization of Container migration monitoring. For the robustness of the proposed concept and operation with huge amounts of real time data, we upgraded the system using InFluxDB [26], a suitable storage engine for real-time traffic monitoring, to achieve scalability, reliability, performance/efficiency and many more. Cloud and the data center operation for the best service delivery use extensively the virtual machines with its profound benefits on resource control and isolated workload management [18].

There are two different approaches of resource virtualization in the cloud: Hypervisor technique (Virtual Machine) and a Container virtualization technique. Virtual machines realize the whole operating system stack over a slice of the hardware resource while Container virtualization uses the lower layers of the running operating system (OS) to implement one or more Containers that are isolated environments to run an application. Both types of virtualization are currently available from cloud service providers. The choice between the two approaches is a trade-off between flexibility and efficiency. The hypervisor technique is less efficient, since it implements the whole OS stack, but, for the same reason, it is agnostic with respect to the host OS. Instead the Container is layered over the current OS kernel, and therefore it is more efficient. A Container-based approach evolves towards the realization of complex but agile distributed architectures, composed of small and specialized services: the micro-service approach is a promising design paradigm that is tightly bound to (or merging with) the Container technology. Container-based virtualization presents an interesting alternative to virtual machines in the cloud. With the innovation of Docker based Container technology, there seems to require extra levels of abstraction in the virtualization still to improve the efficiency of services in the virtual world.

Docker based virtualization [16] provides efficient management of Containers, which has several added benefits in compared with VM operations and management. Several studies have been performed to improve the performance in the Virtual Machine technologies including VM synthesis, repository management, Pareto SLA et cetera. [5,1,17]. Cloud services have direct relationship with Serviced Based Applications in the business process and management where end users are to be guaranteed with quality and performance [25]. For this, efficient service monitoring on each layer is required. These days service providers provide Monitoring as a Service (MaaS) [29] in the cloud with different dimensions and motivations. For the reliable services, the virtualized machines need to be continuously monitored. Cloud service providers shall provide their own system of monitoring interface for which the customers mostly dependent on the health information which shall be provider specific. Customers want to have an independent monitoring system so that they can monitor their applications/infrastructure and activities at anytime from anywhere. They can also move their resources to any service providers independent of system and platform specification. Surveillance system monitors the performance of the host machines during Container migration from one machine to another machine within or outside the cloud. With the increase of number of applications in the cloud, resource management is critical. Customers want to switch the infrastructure service to meet the quality they want. On the other hand, cloud service providers, after monitoring the resources, shall transfer the resources under critical condition over available idle infrastructure for better load management in the dynamic environment. The middleware APIs developed for our system collects the monitoring parameters of Containers as well as real-time migration activities from the service provider's network and visualize the activities using Grafana [30].

2. Literature Review

Monitoring the resources provided by the cloud is the most prominent and challenging task these days from the perspective of security, capacity and resource planning/management, load balancing the providers network and applications, data centers operations and management, achieve better SLA and many more. Considering the cloud to be resource specific, cloud is modeled into layers as: facility, network, hardware, operating system, middleware, application and user [22,23,9]. This layering simply provides the basic concepts about what to monitor and how to monitor the resources. The facility layers mainly focused over the available physical infrastructure in the centers hosting servers and networking equipments, while the network layer deals with the inter-connections and intra-connection of network devices, links/paths within or outside the cloud which shall be other cloud network or federated cloud. Hardware layer addresses the physical components of server and network equipments. The OS mentions the host and the guest OS/platforms in the cloud. The APIs running in the system are the middleware layers which bridges between OS and the user applications. Applications run by the end-users/customers and the users themselves are the part of application and user layers of the cloud. G. Aceto et al. [2] conceptualized the four dimensions (Fig.1) of cloud monitoring with respect to concepts, properties, issues and necessity. The dimensions mainly focused on the design goals/characteristics of distributed and cloud computing system including of current issues in cloud monitoring and its future directions. We mostly focused on the monitoring of virtual resources consumed by the Containers and its dynamic movement to other networking/computing infrastructure.

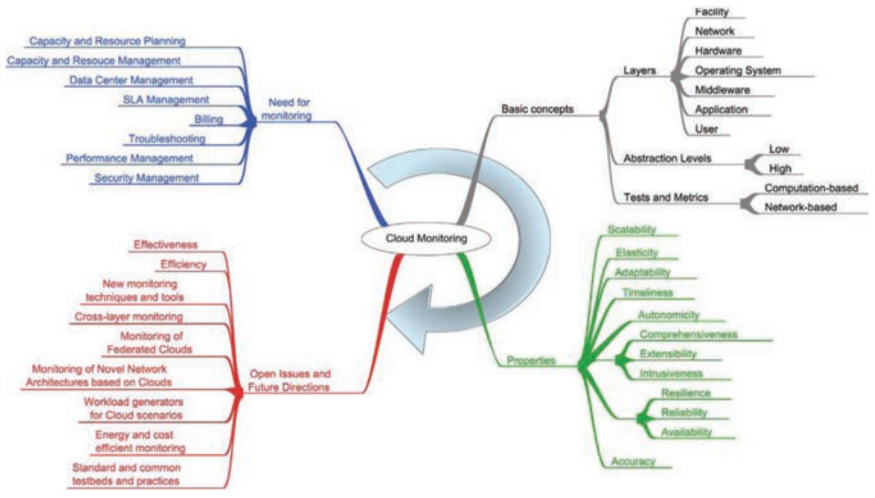


Fig. 1: Different dimensions of cloud monitoring [2]

2.1 Comparison of Container and Virtual Machine

The Container technology in the open source environment has been conceptualized since past several years as a Linux Container (LXC) that support isolated name spaces. Docker provides additional advantages for efficient Container management and operation. The Container based technology is replacing the heavy weight virtual machine technology with efficient and light weight Container technology for cloud applications management, monitoring and operations. Docker Containers are featured as below [20]:

- *Portability of applications over multiple clouds*: Docker provides the platform for building applications and its dependencies into a single object as a Container which shall be migrated to any other Docker enabled machine and executes independently.
- *Application centric*: Docker engine is highly application centric. It provides a best packaging of applications over Container.

- *Tool specific*: Docker tools help developer to automatically assemble a Container from source code with full control over application dependencies, build tools and packaging.
- *Container re-use*: Docker based Containers are the base images which shall be used and reused for packaging any kind of applications and migrate it to any other platform.

Virtual machines run with full OS, where all the functionalities of OS required for its operation, while Containers share the host's OS and are therefore lighter in weight and have better performance. The right way to think about Docker is thus to view each Container as an encapsulation of one program with all its dependencies. The Container is pluggable to any host and it might have everything it needs to operate. In a virtual machine, valuable resources are emulated for the guest OS and hypervisor, which makes it possible to run many instances of one or more operating systems in parallel on a single machine. Every guest OS runs as an individual entity from the host system consumes higher resources. On the other hand, Docker Containers are executed with the Docker engine rather than the hypervisor. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package to the multiple cloud. Containers have therefore less isolation and greater compatibility possible due to sharing of the host's kernel. A virtual machine could take up several minutes to create and launch whereas a Container can be created and launched just in a few seconds. Also, a single server can pack more than one Containers. Docker Containers can run inside Virtual Machines though they are positioned as two separate technologies. Working in heterogeneous environment, VM provides high flexibility whereas Docker Containers' prime focus is on applications and their dependencies [43]. Hence Containers are wrapped-up applications and pieces of software that include all dependencies but use a shared kernel with other Containers (applications). Essentially, Containers are isolated processes in the user-space on the host operating system [6]. While the Hypervisors abstract the entire device, Containers just abstract the OS kernel.

2.2 Overview of Docker and Container Monitoring

One Docker engine may run more than one isolated Containers. The number of Containers in a Docker engine depends on the resources consumed by a Container and the total available resources of the Docker host. The increasing number of Containers consumes more resources shall make the host system more critical. VMs are normally allocated a fixed amount of resources limiting the applications to run. Resources utilized by Containers are in sharable mode. By default, when a Container is launched, there is no memory limits. This can lead to a single Container consumes whole memory leading to the system unstable. However the tools like cgroups [24] help manage the resource utilization by a Container. Hence host's capacity planning and infrastructure load balancing is sensitive in the infrastructure running Containers over the Docker. For this, quality of service metrics like CPU utilization, Memory, Disk, I/O and Network utilization are to be continuously monitored as Docker metrics. Measuring of the Docker metrics help to automatic scheduling of resources like i) stop or move the Containers where resource utilization crossed the limit, ii) start new Containers if the resources are idle. Monitoring of Docker and Container has the main objective to optimize the resources and properly balance the load of existing infrastructure. The Dockers' metrics like CPU, Memory, Disk I/O and Network utilization shall be stored into the time series database (TSDB) as well as processed with the defined rule that provides the threshold for every metric measured and help for automatic adaption of resources in the cloud.

2.3 Related Work in Cloud Monitoring

Based on the virtual machine technologies, there are several open source and commercial cloud monitoring platform and services available. The cloud monitoring platforms and services are like CloudWatch [12], AzureWatch [7], Monitis [31], Nimsoft [34], CloudStatus [11], LogicMonitor [28], Aneka [3], CloudHarmony [10], Zenoss [45] et cetera are commercial cloud monitoring platforms while Nagios [32], OpenNebula [35], Ganglia [19], Zabbix [44], Nimbus [33], PCMONS [36], Sensu[32], ICINGA [21] et cetera are the open source cloud monitoring platforms. There are many monitoring systems already developed for Docker based platform and Container monitoring [6,13,42]. Similarly, a self adaptable platform has been proposed in [46] to make the infrastructure self configurable and adaptable based on the decision performed by the system after getting alarm of resource overloading in the cloud.

For the Container technologies, Docker itself provides the command line tools to its client to extract the status of Container's resource consumption status. "*Docker stats*" command provides the CPU utilization for each Container, the memory used, total memory available, network input/output by the Container. Additionally Docker provides the remote API via netcat [4]. CAdvisor [8] is a free and open source web based tool that simply visualizes the command line output generated by Docker stats and Docker Remote API. It provides the graphs for CPU usage, Memory usage, Network throughput and Disk space utilization. It does not have alerting system when resource consumption exceeds the limit. Scout [39] is another web based hosted monitoring tool for Docker Containers. It can collect metrics from any hosts and Containers then visualize it. It has logical reasoning engine that generates alerts based on those metrics and its defined threshold. DataDog [35] provides the more professional cloud monitoring service including Docker and Containers. It collects metrics about CPU usage, Memory and I/O for all Containers running in the system including counts of running and stopped Containers as well as counts of Docker images. It has flexible alerting system where we can set the metrics threshold as per the SLA. Sensu [41] is another open source monitoring framework that supports the Container monitoring provides self hosted centralized metrics monitoring service. It also supports many plug-in for monitoring to support different business needs. Prometheus [37] is also a self-hosted open-source system monitoring and alerting tool which collectively provide metrics storage, aggregation, visualization and alerting service that collect the metrics on the PULL basis. It features the multi-dimensional time-series data model, flexible query language and autonomous self-hosted and shall be maintained by any organization. Logentries [29] is a log analyzer provides monitoring service based on the log records. It recently added Docker Container monitoring based on the log. It enables logging for Docker of all sizes, aggregate the log for Containers, hosts and applications. Ruxit [38] is the dynamic and scalable Dockerized applications monitoring tool able to monitor micro-services running on Docker Containers. It automatically detects the creation of new Containers and monitors the applications and services contained within them. This helps to applications capacity planning and load balancing of Containers optimizing Container orchestration and clustering. Sematext [40] has added Docker monitoring services into its SPM solution. It is capable to monitor Container lifecycle events (create, exec_create, destroy, export) and runtime events (like die, exec_start, kill, pause, restart, start, stop, unpause) with Containers.

Looking into the State-of-the-Art in the Docker Container monitoring, requirements of application migration to the best performed platform/network help increase the efficiency of services to customers. Monitoring of migration system is required to guarantee that the new decision to move the applications form one cloud to another cloud increase the performance and meet associated service level agreement (SLA). Every cloud service provider maintains the minimum quality of their services as well as they have to meet the minimum requirements to be fulfilled while providing the services agreed upon with customers as SLA.

Table 1. Properties of container monitoring platforms

Platform	Docker Support	Container Status Moni-toring	Container Migration Monitoring	Container Log Monitoring	Open Source	Alert System
CAdvisor [39]	Yes	Yes	No	No	Yes	No
Scout [40]	Yes	Yes	No	-	Yes	Yes
DataDog [35]	Yes	Yes	No	Yes	Yes	Yes
Sensu [32]	Yes	Yes	No	-	Yes	-
Prometheus [41]	Yes	Yes	No	-	Yes	Yes
Logentries [42]	Yes	Yes	No	Yes	Yes	No
Ruxit [43]	Yes	No	No	No	-	No
Sematext [44]	Yes	Yes	No	Yes	-	No
*CoMMon**	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes**

"*CoMMon**" is robust and fully open source Docker based monitoring system and it is expandable to have alerting system and easily integrated with other monitoring platforms.

3. Concept Development

Basically system throughput, network throughput, memory and CPU utilization are the major indicators that may lead to take the decision required to migrate Containers form one host to another. The migration decision shall be from service provider side or from customer side depending upon the type of services and agreement. Our concept is not only to monitor system and application performance of Containers but also monitor the events related to Container migration. The system is useful for the purpose of Container migration whether the Container is successfully migrated to destination or not. Its associated metrics shall be measured from the destination network after migration. We have developed our own java based client server communication system, in which the agent simply detect the Container events at the source and integrate the destination events after successful migration to destination. It can also perform the host system status monitoring with associated individual Container status monitoring. After the successful transfer, the destination sends the destination events (import, start, re-start) to the same monitoring station where the information shall be mapped with the source events with the help of unique Container ID.

The CoMMon client monitoring probe, installed on the Docker host, collects the Container status (CPU, Memory Utilization and Network Bandwidth) using *Docker stats* command and converts the parameter values into JSON format and periodically sends this information to the Central Monitoring Station (CMS). While in the migration, the probe sends another category of message (DateTime, ContainerID, ContainerSize, Container Description, Source IP, Destination IP and Status,) to the CMS. The destination Docker host also has the same monitoring probe which sends the same message with receive status to the station once it successfully receive and run the Container. The collected parameters passed into the monitoring station are directly inserted into the time series database for real time visualization.

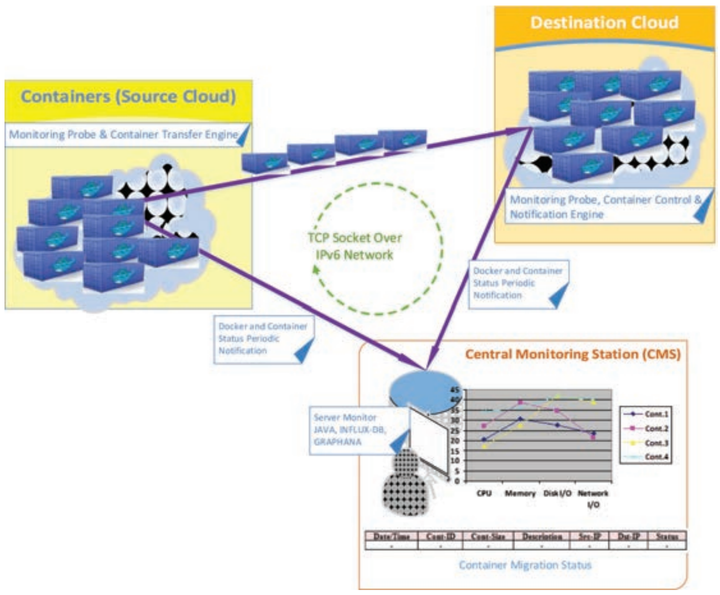

Destination Cloud
Containers (Source Cloud)
Monitoring Probe & Container Transfer Engine
Monitoring Probe, Container Control & Notification Engine
TCP Socket Over IPv6 Network
Docker and Container Status Periodic Notification
Docker and Container Status Periodic Notification
Central Monitoring Station (CMS)
Server Monitor JAVA, INFLUX-DB, GRAPHANA
CPU
Memory
Disk I/O
Network I/O
Cont.1
Cont.2
Cont.3
Cont.4
Date/Time
Cont-ID
Cont-Size
Description
Src-IP
Dst-IP
Status
Container Migration Status

Fig. 2: Real-Time Container migration and monitoring system

Source, destination and monitoring stations are three separate entities shall be located independently far away with each other. However TCP connections are to be established over IPv6 based network while sending and receiving the Containers. The test bed is setup over the IPv6 network environment, because comparatively IPv6 addressing is better than the IPv4 addressing for security, quality of service, high speed and expandability features with sufficient addresses to uniquely identify the host and Containers in the cloud. Monitoring server continuously receive the data from the migration source and destination, store the collected monitoring data to the time series database (InFluxDB) and visualize those data as graphs in the web which shall be accessible from anywhere.

Considering the migration of Containers from multiple sources to multiple destinations, we need to calculate the status of source and destination hosts based on the performance metrics to take decision with respect to resource optimization and load balancing. Considering the resource utilizations based on the Containers only and also considering the equal virtual channel capacity for throughput to each Container, the corresponding metric values at the source and destination Docker host shall be calculated as the sum of resources consumed by the individual Container.

For example, the source host total CPU utilization () is the sum of CPU utilization of individual Container (), where *i* varies from 1 to *m* for the hosts running *m* number of Containers. The calculation is same for the destination host CPU utilization (). Similarly memory, network and disk utilization for source and destination hosts shall be calculated as the sum of resources consumed by individual Containers for the corresponding metrics as shown in the equation below.

$$C_u^s = \sum_{i=1}^{m} c_u^s[i], \quad C_u^d = \sum_{i=1}^{m} c_u^d[i] \tag{1}$$

$$M_u^s = \sum_{i=1}^{m} m_u^s[i], \quad M_u^d = \sum_{i=1}^{m} m_u^d[i] \tag{2}$$

$$D_u^s = \sum_{i=1}^{m} d_u^s[i], \qquad D_u^d = \sum_{i=1}^{m} d_u^d[i] \tag{3}$$

$$T_u^s = \sum_{i=1}^{m} t_u^s[i], \qquad T_u^d = \sum_{i=1}^{m} t_u^d[i] \tag{4}$$

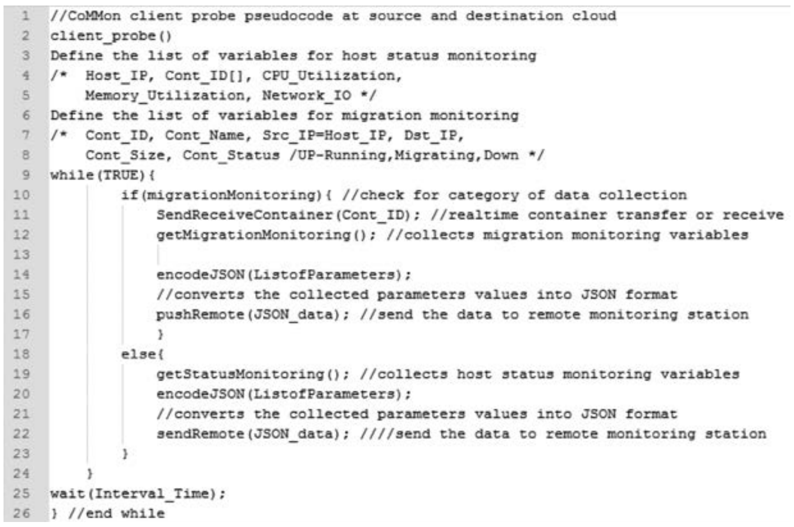
A knowledge based system defines the threshold value for each metric and when it is compared with real-time metric value, this leads to take an automatic decision whether to move the Container to the destination. If and ,   and , and , and are the source host's threshold and real value for CPU, Memory, Disk and Network utilization respectively, following condition holds to take the decision for resource migration.

If $((C_u^s \geq C_\sigma^s) \parallel (M_u^s \geq M_\sigma^s) \parallel (D_u^s \geq D_\sigma^s) \parallel (T_u^s \geq T_\sigma^s))$ then evaluate the destination hosts resource utilization for real-time migration..(*Rule-1*)

To meet the required service level agreement with the customers, if the running Containers make the Docker host overloaded like defined monitoring parameters consumed high resources (Eg: check by *Rule-1*), the service provider shall take a decision to move those specific Containers to other idle hosts within the cloud. Customers, who require only the infrastructure services, shall request any service provider to host their Container from the provider's network. If the service provided is not satisfactory, then customer shall move their Container to another cloud service provider to achieve better performance. In such cases, customer shall independently use this monitoring service.

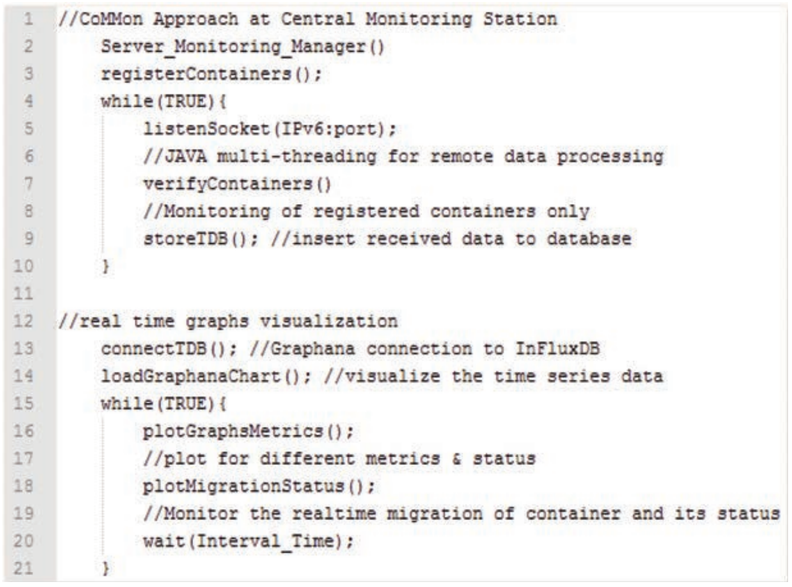
4. Implementation and Evaluation

The system is developed in JAVA by installing Docker on Ubuntu 14.04 virtual machines and Container is configured as web server over IPv6 network environment. The software module consists of client probe to capture real-time metrics and Container migration module for testing purpose at the source host. The destination host consists of the probe for regular Container monitoring together with the file receiver and notifies the status. The probe performed the tasks of collecting monitoring metrics data. It uses the "*Docker stats*" [15], the command line tool to display current status of Containers installed on the Docker hosts. The status information extracted every fifteen seconds period is converted into JSON format and push into remote monitoring station. The probe also collects data while moving the Containers and sends that information to CMS. Server module consists of monitoring manager which receives the data into JSON format, inserts into InFluxDB and stores at different points. The analytical processing logic within the manager incorporates Grafana for visualization.



```
//CoMMon client probe pseudocode at source and destination cloud
client_probe()
Define the list of variables for host status monitoring
/*  Host_IP, Cont_ID[], CPU_Utilization,
    Memory_Utilization, Network_IO */
Define the list of variables for migration monitoring
/*  Cont_ID, Cont_Name, Src_IP=Host_IP, Dst_IP,
    Cont_Size, Cont_Status /UP-Running,Migrating,Down */
while(TRUE){
        if(migrationMonitoring){ //check for category of data collection
            SendReceiveContainer(Cont_ID); //realtime container transfer or receive
            getMigrationMonitoring(); //collects migration monitoring variables

            encodeJSON(ListofParameters);
            //converts the collected parameters values into JSON format
            pushRemote(JSON_data); //send the data to remote monitoring station
            }
        else{
            getStatusMonitoring(); //collects host status monitoring variables
            encodeJSON(ListofParameters);
            //converts the collected parameters values into JSON format
            sendRemote(JSON_data); ////send the data to remote monitoring station
        }
    }
wait(Interval_Time);
} //end while
```

Fig. 3: Pseudo code for client Probe



```
//CoMMon Approach at Central Monitoring Station
    Server_Monitoring_Manager()
    registerContainers();
    while(TRUE){
        listenSocket(IPv6:port);
        //JAVA multi-threading for remote data processing
        verifyContainers()
        //Monitoring of registered containers only
        storeTDB(); //insert received data to database
    }

//real time graphs visualization
    connectTDB(); //Graphana connection to InFluxDB
    loadGraphanaChart(); //visualize the time series data
    while(TRUE){
        plotGraphsMetrics();
        //plot for different metrics & status
        plotMigrationStatus();
        //Monitor the realtime migration of container and its status
        wait(Interval_Time);
    }
```

Fig. 4: Pseudo code for the manager at CMS

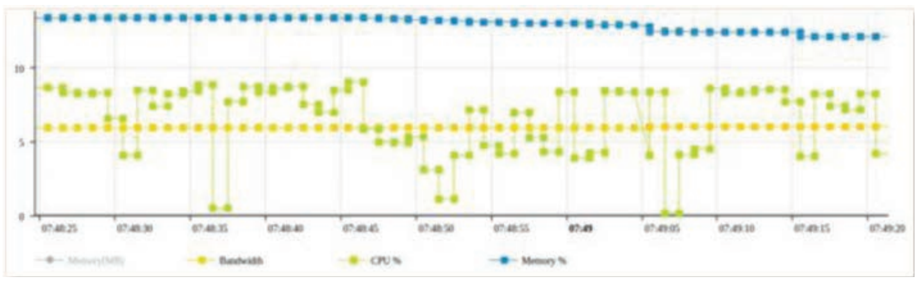


Fig. 5: CPU, Memory and Bandwidth statistics in the source host before migration

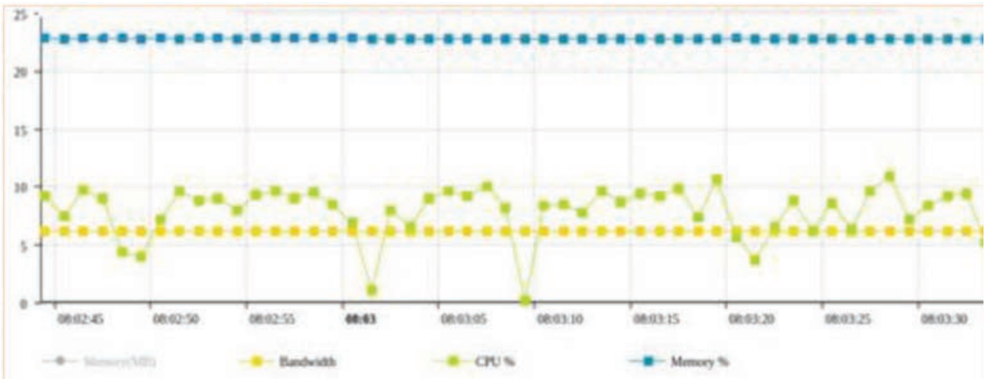
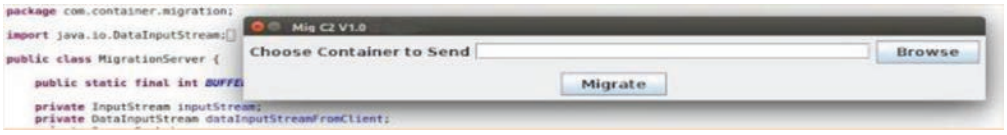


Fig. 6: CPU, Memory and Bandwidth statistics after migration in the destination host

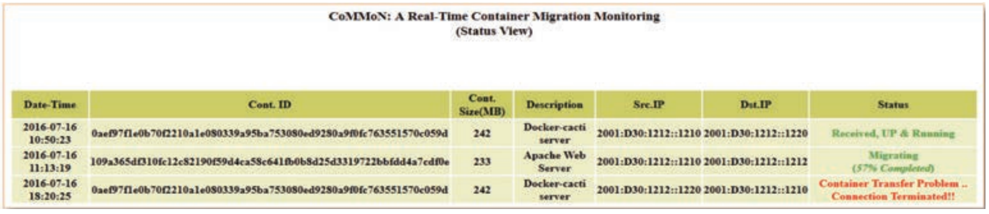
Fig. 5 and 6 comparatively provide status of container before and after migration. The CPU utilization The total CPU, Memory and disk utilization of a host can't be measured from the Container parameters only, however if a host only runs the Containers, then the cornel libraries, system processes shall have fix amount of utilization. However this amount shall be of different values with different host. The graph shows the different pattern of utilization in the source and destination. This research is focused on total utilization based on Containers that ultimately take decision for the migration.



```
package com.container.migration;

import java.io.DataInputStream;

public class MigrationServer {

    public static final int BUFFE

    private InputStream inputStream;
    private DataInputStream dataInputStreamFromClient;
```

Fig. 7: Container migration test snapshot in JAVA

Based on the objective of this research to monitor the migration events of Container, a simple JAVA code (Fig. 7) using FTP has been written to transfer container form one host to another and extract the sending and receiving status. The web based system simply visualizes the migration status and its update on every fifteen seconds interval as shown in the Fig. 8.



CoMMoN: A Real-Time Container Migration Monitoring
(Status View)

Date-Time	Cont. ID	Cont. Size(MB)	Description	Src.IP	Dst.IP	Status
2016-07-16 10:50:23	0aef97f1e0b70f2210a1e080339a95ba753080ed9280a9f0fc763551570c059d	242	Docker-cacti server	2001:D30:1212::1210	2001:D30:1212::1220	Received, UP & Running
2016-07-16 11:13:19	109a365df310fc12c82190f59d4ca58c641fb0b8d25d3319722bbfdd4a7cdf0e	233	Apache Web Server	2001:D30:1212::1210	2001:D30:1212::1212	Migrating *(57% Completed)*
2016-07-16 18:20:25	0aef97f1e0b70f2210a1e080339a95ba753080ed9280a9f0fc763551570c059d	242	Docker-cacti server	2001:D30:1212::1220	2001:D30:1212::1210	Container Transfer Problem .. Connection Terminated!!

Fig. 8: Status view of container migration

5. Conclusion

Monitoring tools for the Containers in the cloud are under commercialization process. We have conceptualized and implemented a robust open source cloud Container migration and monitoring tools in the cloud. This system is easily expandable and integrated into the other open source tools. It can also utilize built-in tools for Docker host monitoring together with Container monitoring and its migration. The monitoring system "*CoMMon*" provides the complete solution for service providers and customers to monitor the resources. While to take decision for Container migration, either client shall take a decision to move their Containers or service providers shall move the Containers transparently within its network. For both of the cases, our monitoring approach monitors the Container migration events with its resource utilization status. The research challenges like: proper message queuing system, security in the migration, knowledge base & automatic decision making system for migration and load balancing are the area of further areas required to be continued with this research in the cloud and sky computing environment.

References

[1] Anderson J and McDougall R (2010), Virtualization performance: Perspectives and challenges ahead. SIGOPS Oper. Syst. Rev. 44(4):40-56.

[2] Aceto G et al. (2013), Cloud monitoring: A survey. Computer Networks 57, 2093-2115.

[3] Aneka, http://www.manjrasoft.com/

[4] Armstrong T (2001), Netcat - The TCP/IP Swiss Army Knife, SANS Institute.

[5] Avi K, Yaniv K, Dor L, Uri L and Anthony L (2007), KVM: the Linux virtual machine monitor. In the Proceedings of the Linux Symposium , volume 1, pages 225-230, Ottawa, Ontario, Canada.

[6] Axibase view on Container monitoring, http://axibase.com/docker-monitoring/

[7] AzureWatch, http://www.paraleap.com/azurewatch

[8] CAdvisor, https://github.com/google/cadvisor

[9] Cloud Security Alliance (2009), Security Guidance for Critical Areas of Focus in Cloud Computing v2.1, www.cloudsecurityalliance.org/csaguide.pdf

[10] CloudHarmony, https://cloudharmony.com/status

[11] CloudStatus, https://cloudstatus.eu/

[12] CloudWatch, http://awsdocs.s3.amazonaws.com/AmazonCloudWatch/latest/acw-dg.pdf

[13] Data dog: dynamic infrastructure monitoring tool, https://www.datadoghq.com/

[14] Dawadi BR and Paudyal R (2016), Tri-Angular Monitoring Approach for Real Time Container Migration. International Journal of Science and Research (IJSR), 5(6) : 774 - 781, DOI: 10.21275/v5i6.NOV164294

[15] Docker Stats API, https://blog.logentries.com/2015/02/what-is-the-docker-stats-api/

[16] Docker Virtualization, https://opensource.com/resources/what-docker/26-02-2016

[17] ENTICE, Horizon2020 Project, http://www.entice-project.eu/

[18] Felter W et al. (2014), An Updated Performance Comparison of Virtual Machines and Linux Containers, IBM Research Report, RC25482 (AUS1407-001) July 21, USA

[19] Ganglia Monitoring, http://ganglia.info/

[20] https://www.openstack.org/summit/tokyo-2015/videos/presentation/monitoring-docker-container-and-dockerized-applications.

[21] ICINGA, https://www.icinga.org/

[22] Spring J (2011), Monitoring cloud computing by layer, Part 1, IEEE Security & Privacy 9 (2) 66-68.

[23] Spring J (2011), Monitoring cloud computing by layer, Part 2, IEEE Security & Privacy 9 (3) 52-55.

[24] Juvva K (2016), Resource Monitoring for cgroup hierarchies, The Linux Foundation Collaboration Summit-2016, http://events.linuxfoundation.org/sites/events/files/slides/CollaborationSummit2016-Slides-Kanaka_0.pdf

[25] Lau K, Lamersdorf W and Pimentel E (ESOCC 2013), LNCS 8135, 188-195, Springer-Verlag Berlin Heidelberg.

[26] Leighton B et al (2015), A Best of Both Worlds Approach to Complex, Efficient, Time Series Data Delivery. In Environmental Software Systems. Infrastructures, Services and Applications (pp. 371-379). Springer International Publishing

[27] Logentries: Log Management & Monitoring System, https://logentries.com/

[28] Logic Monitor, http://ls.logicmonitor.com/monitoring/storage/netapp-filers/

[29] Meng S, Monitoring-as-a-Service in the cloud (MaaS). Master's Thesis, Georgia Institute of Technology.

[30] Metric and Analytic Dashboards, http://grafana.org/

[31] Monitis, http://portal.monitis.com/

[32] Nagios, https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/

[33] Nimbus, http://www.nimbusproject.org/doc/nimbus/faq/

[34] Nimsoft, http://www.ca.com/us/content/page/ca-nimsoft-solutions.aspx

[35] OpenNebula, http://archives.opennebula.org/documentation:archives:rel2.0:img

[36] PCMONS, S.A. Chaves, R.B. Uriarte, C.B. Westphall, Toward an architecture for monitoring private clouds, IEEE Communications Magazine 49 (2011) 130-137

[37] Prometheus, https://prometheus.io/docs/introduction/overview/

[38] Ruxit: All-in-one monitoring for cloud natives, https://ruxit.com

[39] Scout Monitoring, https://scoutapp.com/

[40] Sematext, https://sematext.com/spm/integrations/docker-monitoring/

[41] Sensu, https://sensuapp.org/

[42] Sysdig Cloud: Infrastructure and Application Monitoring for Container, https://sysdig.com/

[43] Virtual Machine Vs Docker, http://devops.com/2014/11/24/docker-vs-vms/

[44] Zabbix, http://www.zabbix.com/

[45] Zenoss, https://www.zenoss.com/

[46] Zhao Z et al. (2015), Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach. In the proceedings of HOLACONF - Cloud Forward: From Distributed to Complete Computing, Pisa, Italy. Elsevier, Procedia Computer Science, 68 : 17-28.