Docker Container vs Virtual Machine
CSC/ECE547 Cloud Computing Technology

David Alban (Lead)
dalban@ncsu.edu

Jaspreet Kaur
jkaur3@ncsu.edu

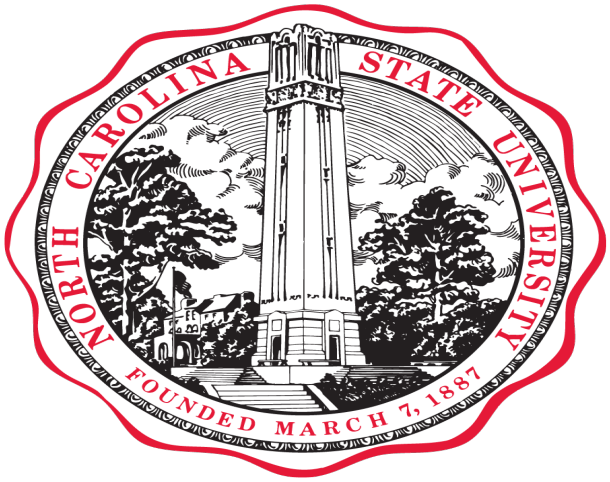
Lahari Kommi
lkommi@ncsu.edu

Abhimanyu Kumar
akumar24@ncsu.edu

Mrunal Mozarkar
mmozark@ncsu.edu

Shyam Ramakrishnan
sramakr9@ncsu.edu

Sunday 3rd December, 2017



Abstract

In the era of new technology and too many options, it has become difficult to make correct choice of technology for a specific system. Our project aims to provide the differences in performance between two of the virtualization technologies: Virtual Machines and Docker containers. We test both these technologies based on various metrics like memory utilization, CPU performance, floating point operations per second, system load and network latency. We make use of various tools like stress, ping, Bonnie++ for our testing. We also take into consideration various factors like scalability, security, ease of deployment, etc. into consideration.

Our study shows that the performance of either VMs or Containers can be considered optimal but, it is largely based on the factors under consideration. Essentially, there is no best option for all.

Contents

List of Figures

List of Tables

1 Introduction

Dreher et al. has defined cloud computing as a service-oriented computational paradigm that, at the requested and appropriate level, seamlessly and securely provisions a wide range of IT services through a combination of connections, computers, storage, software and services accessed over a network[2]. Thus, it offers various levels of virtualized services to users and, in this project, we will be focusing on two of the virtualization services, viz Virtual Machines and Docker Containers, offered by cloud service providers.

A Virtual Machine (VM) is essentially an emulation of a real computer that executes programs like a real computer. VMs run on top of a physical machine using a hypervisor which in turn, runs on either a host machine or on bare-metal[3]. In Cloud Computing, VMs are offered as Infrastructure-as-a-Service (IaaS). More information on IaaS is available in Appendix A

Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the user space[3]. In Cloud Computing, containers are offered as Container-as-a-Service (CaaS). More information on CaaS is available in Appendix A

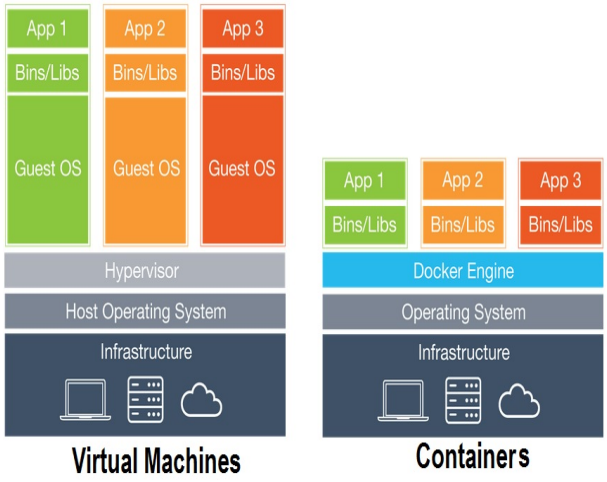


Figure 1: Containers vs VM [1]

1.1 Problem Statement

In this project, we evaluate and compare performances of two different cloud service offerings: Virtual Machines and Docker Containers. We evaluate the two technologies based on quantifiable metrics of performance like CPU performance, memory utilization and disk I/O performance as well as qualifiable metrics of performance like scalability, ease of deployment, start-up time, etc.

1.2 Motivation

To understand why this comparison is required, we look at the business paradigm in general. All the businesses work with the aim of minimizing costs while providing required services and maximizing profits. And this is where the two technologies of Virtual Machines and Containers come into play since both of them offer an ability to run applications isolated from the host OS and can be made available over the network.

Thus, it is important for anyone who wishes to use this technology to understand the implications of using either service. From the perspective of a cloud vendor that provides Application-as-a-Service, it is essential to figure out whether hosting the application on a container is better, or hosting an application on a VM is the better option. Now, we consider a cloud customer's perspective. Even the customer needs to chose the best possible option as per the organization/individual's need. In both cases, the best option would depend on various factors like the required start-up time, security and stability requirements, cost constraints etc. It is difficult to make the optimal choice since it requires careful insight in to the performance of the services under different user application environments. Thus, our project aims to provide an in-depth analysis of the

performance differences between Containers and Virtual Machines enabling vendors as well as customers in making informed decision.

1.3 Environment

To perform this comparative performance study, we use CentOS 7 Minimal as the host OS for both the service models. We chose the same host OS for both architectures to ensure fair comparison. For Virtual Machines setup, we chose "Virtual Box" as the hypervisor and "Docker" as the Container Engine. We make use of applications like Stress, Stress-ng, IOzone3 and Bonnie++ for testing. For fetching various metrics, we deployed the 'collectd' utility along the time series database 'Influxdb' for storage. For the display of data in real time, we make use of the graphical intefrace called *Grafana*. We have also added a secondary graphical interface called *Collectd-web* to visualize more granular details. Detailed information of the system environment is presented in section [3].

1.4 Issues

We faced some issues while installing Operating systems and integrating the monitoring tools. This section provides a comprehensive list and description of the issues we faced as well as the steps taken to solve them.

- **Guest Operating System**
  Description: The guest operating system need not only be light (small) but also comparable in size for fairness between the virtual machine and container. Initially, Alpine OS was chosen since it is one of the lightest OS that is available; the container only needs 8 MB and the Minimal installation requires around 130 MB of storage[4]. Alpine was installed on both the virtual machine and container compute nodes without any major problems. However there were issues with installation of the testing applications in Alpine since the applications were not directly available in the default repositories and mirrors corresponding to Alpine OS and little to no support could be found for installing these applications on Alpine.
  Action: Guest operating system was changed to CentOS 7 Minimal. CentOS container only needs 203 MB and the Minimal iso for VM requires 830.5 MB [5]. CentOS has support for the various testing applications that were chosen for the comparison between virtual machine and container.

- **Host Operating System**
  Description: The host operating system needs to be as comparable in size and functionality as possible for fairness between the Virtual Machine and Container. CoreOS was given in the problem statement. Unfortunately, it cannot be used to host virtual machines. Thus a comparable operating system to CoreOS was needed for a fair comparison between the Virtual Machine and Container. CentOS 7 Minimal, which requires 830.5 MB [5] was the closest operating system to CoreOS with its Productions size of 333.4 MB [6]. After further review with management, the size difference between CoreOS and CentOS 7 Minimal was not acceptable.
  Action: CoreOS was dropped as the host operating system since there was no other alternate operating system that was comparable in size and that could host virtual machines. In its place CentOS 7 Minimal was chosen as host operating system for both the virtual machine and container.

- **Fully Qualified Domain Name (FQDN)**
  Description: One of the compute nodes (Dell T310) in our infrastructure had problems with its Fully Qualified Domain Name. Without a functioning FQDN, network to that particular compute node was not possible.
  Action: Reinstalled host and guest OS on the faulty compute node, along with the virtual machine and test applications with no success. Finally decided to turn off the compute node, in order to prevent any network issues with the remainder of the system.

- **Assign Private IP to Virtual Machines**
  Description: To protect the computes nodes from outside interference, the virtual machines needed to be assigned private IPs. Bridging could provide IPs to the virtual machines, but they would be public. Host-only virtual networking could provide private IPs to the virtual machines, but the right adapter

would be needed.
Action: Host-only virtual networking needed manual creation of virtual interface vboxnet0, which could be accomplish with the following command [7]:

```
VBoxManage hostonlyif create
```

2 Requirements

Before designing a system, we need to have a fair idea of what the system is required to do, how it is expected to behave and what features it has to provide. Thus, before we move on to the system environment and design, we need to understand the requirements for our system. While considering these requirements, we can divide the requirements into two types:

- Functional Requirements
- Non-functional Requirements

2.1 Functional Requirements

Functional requirement defines a functionality that is expected of a system or its components[8]. For example, how the system is required to behave and its expected results.

- **FR1** Create a system to compare disk I/O utilization between a virtual machine and a container.
  - **FR1.1** Create a docker container environment to support I/O intensive applications, IOzone and Bonnie++.
  - **FR1.2** Install the above applications onto the Docker container and save the container image on private repository
  - **FR1.3** Create a CentOS virtual machine which can support I/O intensive applications
  - **FR1.4** Install I/O intensive applications IOZone and Bonnie++ on the virtual machines.
- **FR2** Create a system to compare CPU performance as well as Memory utilization between a virtual machine and a container
  - **FR2.1** Create a Docker container with a CPU and memory intensive applications: 'stress' and 'stress-ng'
  - **FR2.2** Install the above applications onto the Docker container and save them on private repository
  - **FR2.3** Create CentOS virtual machine which can support CPU and memory intensive application:
  - **FR2.4** Install CPU and memory intensive application, 'stress' and 'stress-ng' on the virtual machines
- **FR3** Measure system load, memory utilization, CPU utilization and disk I/O
  - **FR3.1** Perform system load, memory utilization, CPU utilization and disk I/O tests with varied load configurations.
  - **FR3.2** Measure the performance across the three hardware systems *(pP390, t310 and t610).*

2.2 Non-functional Requirements

In addition to the obvious features and functions that the system will provide, there are some more additional requirements that describe the behavior and important characteristics of the system. These are called "Non-functional requirements" or "Quality Attributes". These basically specify criteria that can be used to judge the operations of a system, rather than specific behavior.[9].

- **NFR1** Comparison based on 'Ease of Deployment'
  This requirement looks at the ease in effort and time required to bring the resources of a particular service into effective action. Today, many user applications needs to be spawned up instantly. We provide the comparison between our Containers and Virtual Machines, and provide the observations regarding what option requires less installation time and effort.
- **NFR2** Comparison based on 'Scalabililty'
  One of the important factors of the comparison is the ability of any technology to scale. Thus, we consider the comparison between ability and ease of scaling demonstrated by Virtual Machines and Containers as an important non-functional requirement.
- **NFR3** Comparison based on 'Flexibility'
  Another important factor of the comparison is flexibility of the system. This basically factors in the amount of dependency the guest OS has on the host OS.
- **NFR4**. Automation
  In order to make our system user friendly, our system should exhibit the ability to trigger the benchmark applications automatically via some GUI, reducing the manual effort.
- **NFR5** Display test data collected from contained virtual machines in graphical format. Essentially, the system should provide a graphical user dashboard that fetches and displays real time statistics for CPU load, memory etc.

2.3 List of tasks

Now that we have established the requirements, we briefly describe the tasks performed to facilitate our system in providing the above functional and non-functional requirements.

- Install Host OS (CentOS 7 minimal) on all the available servers
- Install Docker engine on Pods 1B-pP390, 1B-t310, 1B-t610.
- Run 3 instances of CentOS Container and install relevant applications. Now, commit the newly created images onto the private repository
- Install Virtualbox on on Pods 1A-pP390, 1A-t310, 1A-t610.
- Install CentOS 7 minimal VM on the virtualbox

2.4 Specifics of the environment

This section discusses basic building blocks to set up a Container and VM environment. The base hardware selected are Dell servers and Cisco switches for network connectivity. OS installed for both the systems is CentOS 7 Minimal. Docker and Virtual Box is installed on the respective base environments. More details are available in section 3

3 System Environment

The following chapter details about the hardware and software setup designed in the laboratory for the implementation of the project. Both the architectures consists of common Dell T610, T310(x2) and PP390 servers as base hardware. All these servers are connected to two network switches and are in the same broadcast domain. Of the total servers, one Dell T310 machine is chosen as the server, to which rest of the client machines will send the test data. This server is then used to display the test results using Grafana and CollectD application. The complete network topology is shown in figure [2].Each of the servers have CentOS7 minimal image installed which serves as host OS to run the benchmark applications.

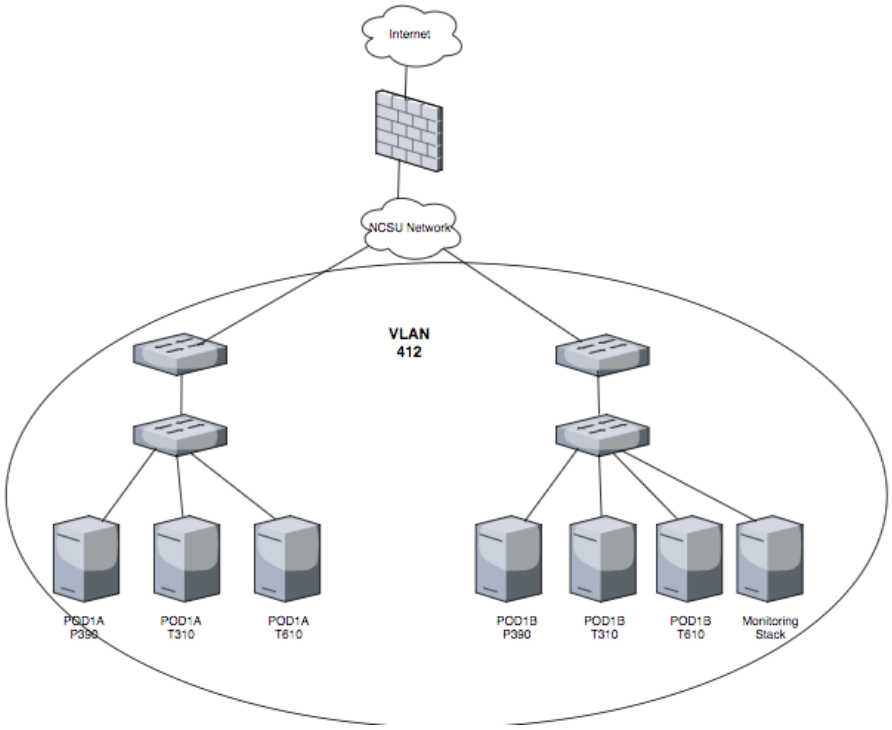


Figure 2: Network Topology for our system

Table below provides the technical specifications for each of the servers:

Server	Dell T610	Dell PP390	Dell T310(x2)
Processor	Intel Xeon E5620 - 2.4GHz	Intel Core 2 Duo 6600 - 2.4 GHz	Intel Xeon E5620 - 2.4GHz
RAM	26GB	2GB	8GB
Storage	1000GB	250GB	500GB
Operating System	Centos 7 Minimal	Centos 7 Minimal	Centos 7 Minimal

3.1 Virtual Machine Setup

VirtualBox is installed on CentOS 7 Minimal host OS on all the machines reserved for VM architecture. Created CentOS VMs using VirtualBox and installed the test applications i.e, IOZone3, Stress, Stress-ng and Bonnie++.

3.2 Container Setup

Installed Docker Engine, on CentOS. Created Docker images for Test applications i.e IOZone3, Stress, Bonnie++. Pushed these, to the Docker Repository. Pulled each applications as per needed for testing.

3.3 Application and Dashboard

Testing applications have been divided into two separate categories:

- Compute Specific
  - Stress (v1.0.4)
  - Stress-ng (v0.07.29)
- I/O and Memory Specific

- IOZone3 (v3.424)
- Bonnie++ (v1.97)

The dashboard used to fetch and display test results in real time is Grafana and CollectD-web.

4 Design

The section explains design approach considered and implemented. It is based upon the lab network topology and system environment given to us, as explained in Section 3. The design has been chosen with careful consideration to provide a fair comparison between Virtual Machines and Containers. The design will also help the user in knowing the benefits and drawbacks of choosing Container-as-a-Service (CaaS) and Infrastructure-as-a-Service (IaaS). We had separated our deployment and monitoring stack, due to which by only making a few configuration changes, our system is able to scale up.

We know that apart from providing CaaS and IaaS, some cloud providers also provide Application-as-a-Service (AaaS), Software-as-a-Service (SaaS), and many others. Fundamentally, these services are applications running on a Virtual Machine or a Container. Hence, our design will help cloud providers in choosing where to host those services, on a Container or on a Virtual Machine.

Achieving this design required careful selection of system resources which needed to be monitored to provide performance comparison between a Virtual Machine and a container. After some research, CPU, Memory, Disk I/O, Load (System Utilization) have been chosen as the system resources on the basis of which we will do our testing. The above metrics will be monitored and will be displayed to the user in a graphical format.

To simplify the process of testing those resources, we have created a Graphical User Interface (GUI). It is a python based application which will execute the benchmark applications at the back-end.

4.1 System Architecture

We have divided our lab setup into two separate architectures, one each for container and virtual machine. The physical hardware remains uniform across both the architectures. Details for hardware and software specifications is mentioned in Section 3. CollectD daemon is used to collect system and performance statistics [5.3.1] Of all the physical machines, one is selected as "Server", the host OS of which is running the server side of the CollectD daemon. While Host OS on rest of the physical machines is running the client side of the CollectD daemon.

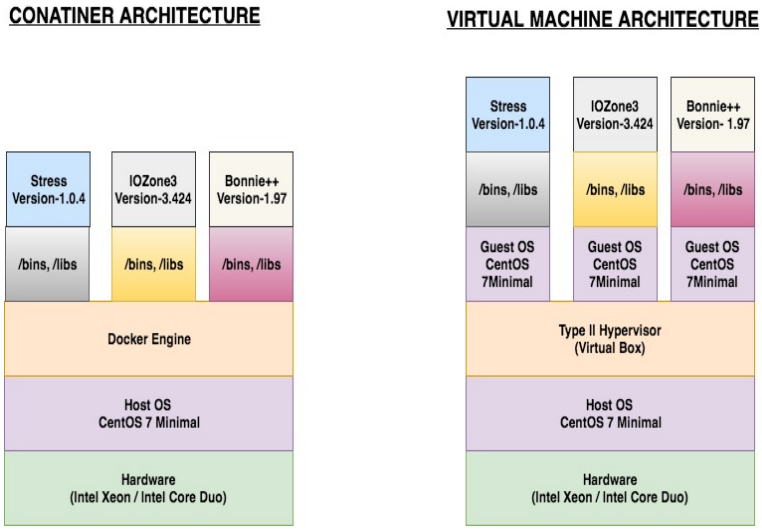


Figure 3: System Architecture

As seen in the Figure [3], we have created two different systems architecture. For the container implementations, we have created 3 separate Docker containers. Each container is running one benchmark application i.e. Stress, Bonnie++ and IOZone3. Similar approach has been adopted for Virtual Machine set up. We have launched 3 separate VirtualBox VMs, each one installed with one testing application.

The host OS on each of these physical machines is running the client side of the CollectD daemon [5.3.1] All the system statistics are fetched by the CollectD daemon and sent to the following:

- **CollectD server:** The CollectD server stores the data in *RRD* format. The CollectD-web application reads the *RRD* data and displays the corresponding graph to the user.
- **InfluxDB server:** The InfluxDB server maintains a *collectd* database and stores the incoming values from CollectD clients in it. The Grafana dashboard reads data from the *collectd* database and displays the corresponding graph to the user.

The monitoring stack is depicted in Figure [4],

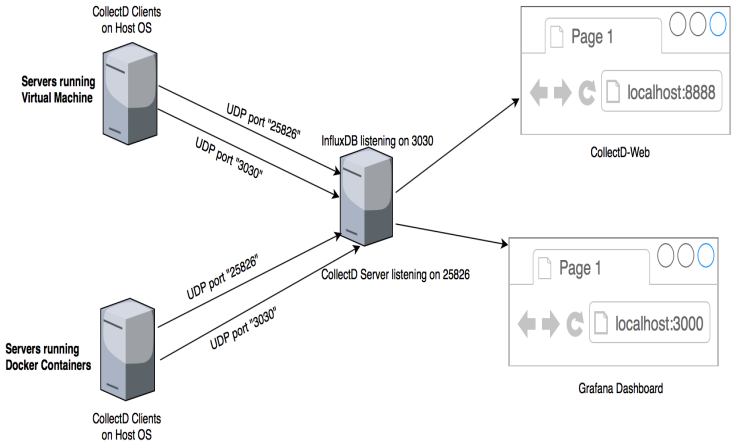


Figure 4: Monitoring Stack Setup

4.2 Testing Applications

Testing applications are used to test our systems. Based upon the functional and non-functional requirements,applications selected are Stress, IOZone3 and Bonnie ++. All these applications are lightweight and easy to work with. They also, offer wide flexibility with large number of configuration options, to test various aspects of a machine like CPU, I/O, memory etc.

4.2.1 IOZone Filesystem Benchmark

IOZone is an open source filesystem benchmark tool. IOzone3 is useful for determining a broad file-system analysis of a vendors computer platform. The benchmark tests file I/O performance for a wide range of applications like read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write, pread/pwrite.[10] A detailed functioning of the application is referred in Appendix B.1.

4.2.2 Bonnie++ Hard drive Benchmark

Bonnie++ is an open source filesystem benchmark tool. The benchmark tests are data read and write speed, number of seeks that can be performed per second, and number of file metadata operations that can be performed per second. Bonnie++ is a valuable tool for changes on RAID setup, filesystems creation, or performance of network filesystems[11][12]. A detailed functioning of the application is referred in Appendix B.2.

4.2.3 Stress Memory Utilization

Stress is an open source tool design to exercise various operating system kernel interfaces and physical subsystems of a computer. Stress include disk I/O intensive, I/O scheduler or a task scheduler tests[13]. A detailed functioning of the application is referred in Appendix B.3.

4.2.4 Stress-ng CPU Utilization

Stress-ng is an open source tool design to exercise various operating system kernel interfaces and physical subsystems of a computer. Stress-ng includes CPU specific test that exercise floating point, integer, bit manipulation and control flow. Stress-ng is able to measure test throughput rates, which can be used to compare performance between different operating systems[14].A detailed functioning of the application is referred in Appendix B.4.

4.3 Front End Application

We have developed a simple front end application using Python framework named Tkinter. This application allows us to automate the execution of benchmark tests on VM and Container systems. Users can trigger a performance test on their local machine, the program uses a Python module called Paramiko to ssh into the Host machines and then into the guest virtual machine machine or container. The test applications are then executed and once they are completed, the results are displayed back on the user's local terminal running the application.

5 Implementation Details

5.1 Virtual Machine Deployment

Oracle VirtualBox was chosen as the virtual machine since it is being actively deployed, has frequent releases and support in a variety of operating systems. In this design VirtualBox will be installed on top of the host operating system, CentOS 7 Minimal[15].

To install Oracle VirtualBox on CentOS 7 Minimal on the Dell PPP390, T310 and T610 machines, follow the steps provided below[16]:

- Navigate to directory where VirtualBox yum repository will be loaded in your machine:

```
cd /etc/yum.repos.d/
```

- Install wget package for retrieving files using the most widely-used Internet protocols:

```
yum install -y wget
```

- Add VirtualBox yum repository in your machine:

```
wget http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo
```

- Add EPEL yum repository:

```
rpm -Uvh http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages
/e/epel-release-7-11.noarch.rpm
```

- Install all the required packages to run VirtualBox:

```
yum install gcc make patch  dkms qt libgomp -y
yum install kernel-headers kernel-devel fontforge
binutils glibc-headers glibc-devel -y
```

- Update system with latest Kernel:

```
reboot
```

- Check that the path exist:

```
cd /usr/src/kernels/
```

- Return to root:

```
cd ~
```

- Setup environment variable:

```
export KERN_DIR=/usr/src/kernels/`uname -r`
```

- Install VirtualBox package:

```
yum install VirtualBox-5.2 -y
```

- Rebuild kernel modules:

```
yum install -y kernel
```

- Install CentOS Development Tools package:

```
yum groupinstall "Development Tools"    y
```

- Install X11 and xterm packages:

```
yum install xorg-x11-xauth xterm -y
```

- Reboot the system
- Launch VirtualBox and create a new VM by clicking on New icon in Oracle VM VirtualBox Manager
- Provide name of the VM, select Linux for type, and select Other Linux (64-bit) for Version; then press Next
- Enter 1024 MB for Memory size and press Next
- Select Create a virtual hard disk now and press Next
- Select VDI (VirtualBox Disk Image) and press Next.
- Select Fixed size and press Next. Provide name of the virtual hard disk, set it to 8.00 GB and press Next.
- Go to settings by clicking on Settings icon in Oracle VM VirtualBox Manager.
- Select Storage, for Storage Devices select Empty. Under attributes for Optical Drive select Virtual Optical Disk File and press OK.
- Select the image you are loading and press Open. Press OK and start.
- Login as root.
- Execute commands to install operating system.

5.2 Docker Image Deployment

To test docker containers, we needed to install docker on top of the host operating system. Once installed we needed to run CentOS 7 minimal container and install required tools/applications that we use during our testing on it. After this installation, we saved the new containers in our private repository. Configurations steps are as follows[1]:

- Install Docker on the host machine:

```
yum -y install docker
```

- Run the base CentOS container

```
docker run -it centos /bin/bash
```

- Install testing applications on three separate containers:
  - Install *stress* on docker container

```
yum -y install epel-release
yum -y install stress
```

  - Install *iozone3* on docker container

```
http://ftp.tu-chemnitz.de/pub/linux/dag/redhat/el7/en/x86_64/rp
mforge/RPMS/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm

rpm -Uvh rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm

yum install iozone
```

  - Install *Bonnie++* on docker container

```
yum -y install epel-release
yum -y install bonnie++
```

- Commit and push the new containers to private repository

```
docker commit -m "[build_notes]" -a "creator info"
<container name or ID] [name of image]:[version tag]

docker tag centos-stress mmozark/centos-stress

docker push mmozark/centos-stress
```

5.3 Monitoring Stack Setup

We have set up a monitoring stack on CentOS 7 Minimal by using CollectD, InfluxDB, Grafana and CollectD-Web. Configuration steps are explained below[17]

[1]If a normal user is running the commands, they should run 'sudo su' before running all the commands mentioned in this document

5.3.1 CollectD Configuration

CollectD is a daemon which collects system and application performance metrics periodically and provides mechanisms to store the values in a variety of ways. CollectD follows client-server model, i.e. the client nodes (nodes where VM and Containers are installed) will send their system statistics to the server. We configured CollectD on the host OS of the clients to send the data to InfluxDB server and CollectD server (both hosted on the same machine). [18].

- Install CollectD on client machines:

```
yum -y install epel-release
yum -y install collectd
```

- Edit the configuration file **/etc/collectd.conf** on the client machines to have the following lines:

```
LoadPlugin network
LoadPlugin rrdtool
LoadPlugin cpu
LoadPlugin memory
LoadPlugin load
<Plugin network>
    Server ``InfluxDB Server IP" "3030"
    Server ``CollectD Server IP" "25826"
</Plugin>
```

- Edit the configuration file **/etc/collectd.conf** on the server machine to have the following lines:

```
LoadPlugin network
<Plugin network>
    Listen "0.0.0.0" "25826"
</Plugin>
```

- Now start the CollectD service:

```
systemctl start collectd.service
systemctl enable collectd.service
```

5.3.2 InfluxDB Configuration

InfluxDB is an open-source time series database written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics[19].

Configuration Steps:

- Add repository for InfluxDB:

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

- Install InfluxDB:

```
yum install -y influxdb
```

- Edit the InfluxDB configuration file /etc/opt/influxdb/influxdb.conf and change the lines near the [collectd] heading as follows:

```
[[collectd]]
enabled = true
bind-address = "127.0.0.1:3030"
database = "collectd"
typesdb = "/usr/share/collectd/types.db"
```

- Then start the daemon:

```
systemctl start influxdb.service
```

5.3.3 Grafana Configuration

Grafana is an open source application for visualizing large-scale measurement data[20].

- Add repository for Grafana:

```
cat <<EOF | sudo tee /etc/yum.repos.d/grafana.repo
[grafana]
name=grafana
baseurl=https://packagecloud.io/grafana/stable/el/6/basearch
repo_gpgcheck=1
enabled=0
gpgcheck=1
gpgkey=https://packagecloud.io/gpg.key https://grafanarel.s3.amazonaws.com/
RPM-GPG-KEY-grafana
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
EOF
```

- Start Grafana:

```
systemctl start grafana-server.service
systemctl enable grafana-server.service
```

- Start a web browser and navigate to http://localhost:3000. Login with user-name "admin" and password "admin"
- Add InfluxDB as a data source to Grafana as shown in figures5 and 6
- Add a new graph to the dashboard as shown in the figure [7]
- Select the metrics and the hosts as shown in figure [8] to view the metrics in a graphical format.

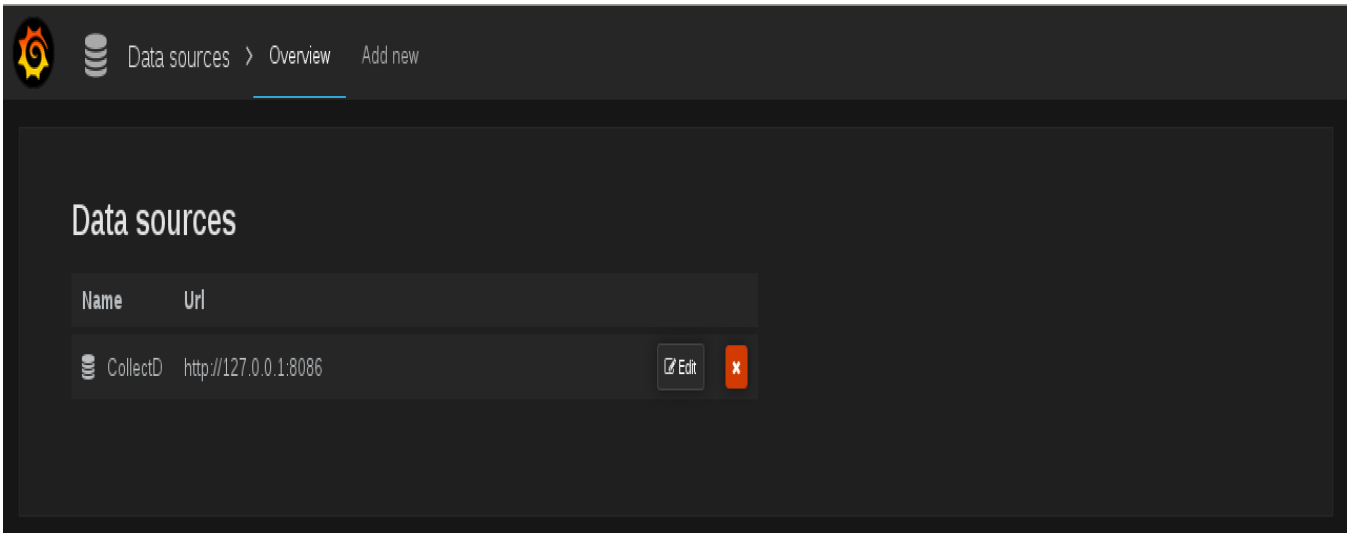


Figure 5: Add data source to the dashboard - Part 1

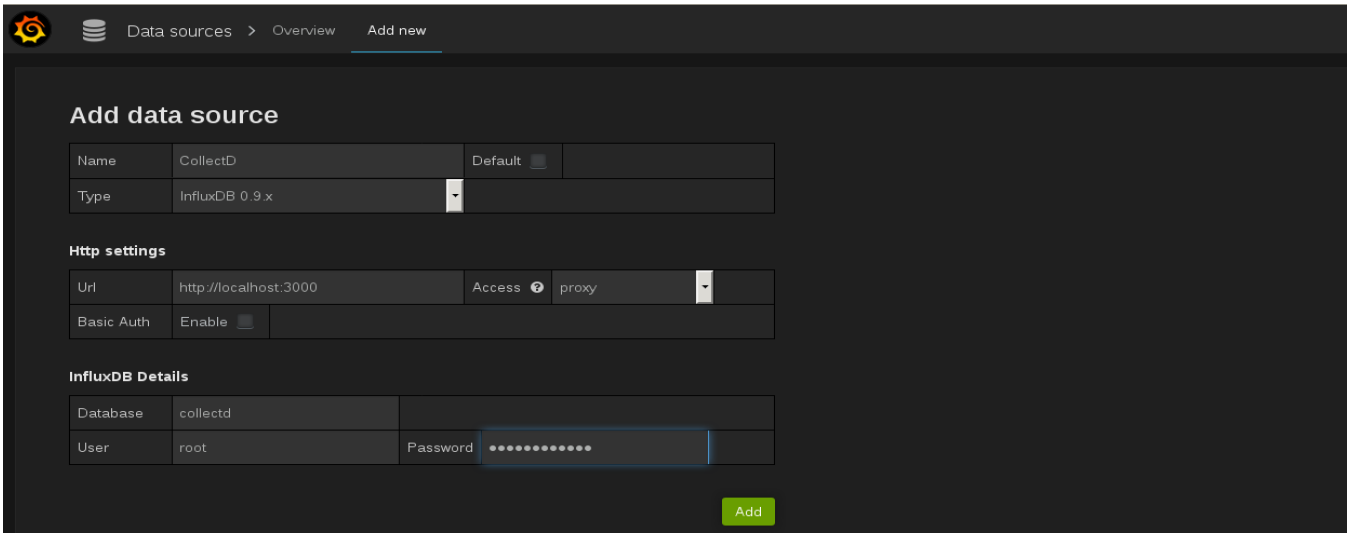


Figure 6: Add data source to the dashboard - Part 2

Figure 7: Add a new graph to the dashboard

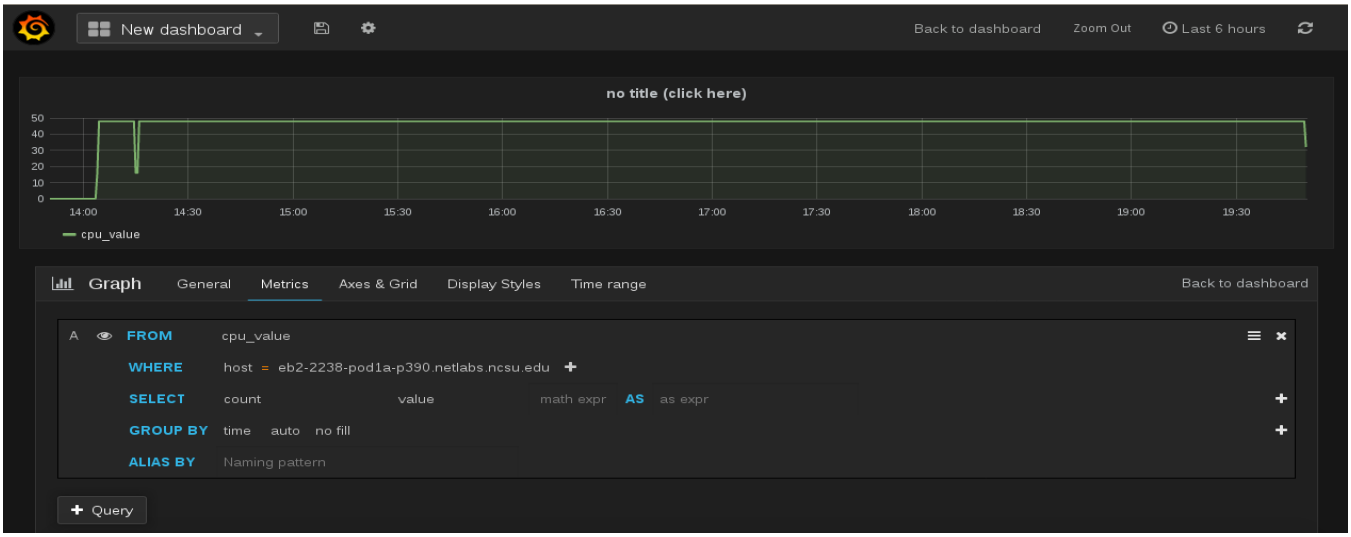


Figure 8: Select metrics to be displayed

5.3.4 CollectD-Web setup

CollectD-Web is CollectD's web-based front-end which is reads RRD data and plots them in a graphical format.

- Installing Dependencies:

```
yum install -y git
yum install -y install  rrdtool rrdtool-devel rrdtool-perl
perl-HTML-Parser perl-JSON perl-CGI
```

- Configurations:

```
cd /usr/local/
git clone https://github.com/httpdss/collectd-web.git
cd collectd-web/
```

```
ls
chmod +x cgi-bin/graphdefs.cgi
nano runserver.py
```

Change the IP address to the IP address of the client or "0.0.0.0" for multiple clients, as shown in figure [9].

```
#!/usr/bin/env python

import CGIHTTPServer
import BaseHTTPServer
from optparse import OptionParser

class Handler(CGIHTTPServer.CGIHTTPRequestHandler):
    cgi_directories = ["/cgi-bin"]

PORT = 8888

def main():
    parser = OptionParser()
    opts, args = parser.parse_args()
    if args:
        httpd = BaseHTTPServer.HTTPServer((args[0], int(args[1])), Handler)
        print "Collectd-web server running at http://%s:%s/" % (args[0], args[1])
    else:
        httpd = BaseHTTPServer.HTTPServer(("0.0.0.0", PORT), Handler)
        print "Collectd-web server running at http://%s:%s/" % ("0.0.0.0", PORT)
    httpd.serve_forever()

if __name__ == "__main__":
    main()
```

Figure 9: CollectD-web configuration in *runserver.py*

- Create a directory *collectd* in */etc/*. Create the file *collection.conf*, add the following line to it:

  ```
  DataDir: "/var/lib/collectd/rrd/"
  ```

6 Verification and Validation

We have distributed our test cases into three main categories. The first category tests the correctness of our testing system. The second performs the quantitative comparison and the third is based on the qualitative performance.

Note: We had given a Virtual Machine one core as its CPU. By default, when we run a Docker Container, its CPU core affinity is x (varies with respect to hardware), where $x > 1$. This gives the containers an unfair advantage. To avoid this, we have provided each Docker Container with CPU core affinity of 1.

Test No.	Description	Expected Result	Actual Result	Requirement
T1.1	Verify Docker Engine Installation	docker –version should display the correct version if installation was successful	Docker version 1.12.6 was the output	FR1.1, FR2.1

Table 1: Test Cases

From the figure 10, we see that the docker engine has been successfully installed.

```
[root@eb2-2238-pod1b-t610 ~]# date
Sun Dec  3 17:31:49 EST 2017
[root@eb2-2238-pod1b-t610 ~]# docker --version
Docker version 1.12.6, build 85d7426/1.12.6
[root@eb2-2238-pod1b-t610 ~]#
```

Figure 10: Docker Engine Version

Test No.	Description	Expected Result	Actual Result	Requirement
T1.2	Create docker container IOzone and verify IOzone Docker image installation	Docker images command should display IOzone	IOzone image displayed with image ID dba8355c6f9b	FR1.1
T1.3	Create docker container Bonnie++ and verify Bonnie++ Docker image installation	Docker images command should display Bonnie++	Bonnie++ image displayed with image ID 037a3f5de3fc	FR1.1
T1.4	Create docker container Stress and verify Stress Docker image installation	Docker images command should display Stress	Stress image displayed with image ID f671f3ac3e0e	FR2.1
T1.5	Create docker container Stress-ng and verify Stress-ng Docker image installation	Docker images command should display Stress-ng	Stress-ng image displayed with image ID f671f3ac3e0e (stress and stress-ng are installed on same container)	FR2.1

Table 2: Test Cases (Cont.)

The figure 11 shows that the container images have been updated with above mentioned applications and saved as new image. Please note that both stress and stress-ng have been installed in centos-stress container.

```
[root@eb2-2238-pod1b-t610 ~]# date
Sun Dec  3 17:30:26 EST 2017
[root@eb2-2238-pod1b-t610 ~]# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
docker.io/centos                    latest              3fa822599e10        3 days ago          203.5 MB
docker.io/mmozark/centos-iozone     latest              dba8355c6f9b        2 weeks ago         278.1 MB
docker.io/mmozark/centos_bonnie     latest              037a3f5de3fc        2 weeks ago         348.1 MB
docker.io/mmozark/centos-stress     latest              f671f3ac3e0e        2 weeks ago         316.8 MB
[root@eb2-2238-pod1b-t610 ~]#
```

Figure 11: Docker Engine Application Version

Test No.	Description	Expected Result	Actual Result	Requirement
T1.6	Verify VirtualBox installation	VirtualBox GUI should popup when VirtualBox command is run from CLI	VirtualBox GUI can be accessed	FR1.3, FR2.3

Table 3: Test Cases (Cont.)

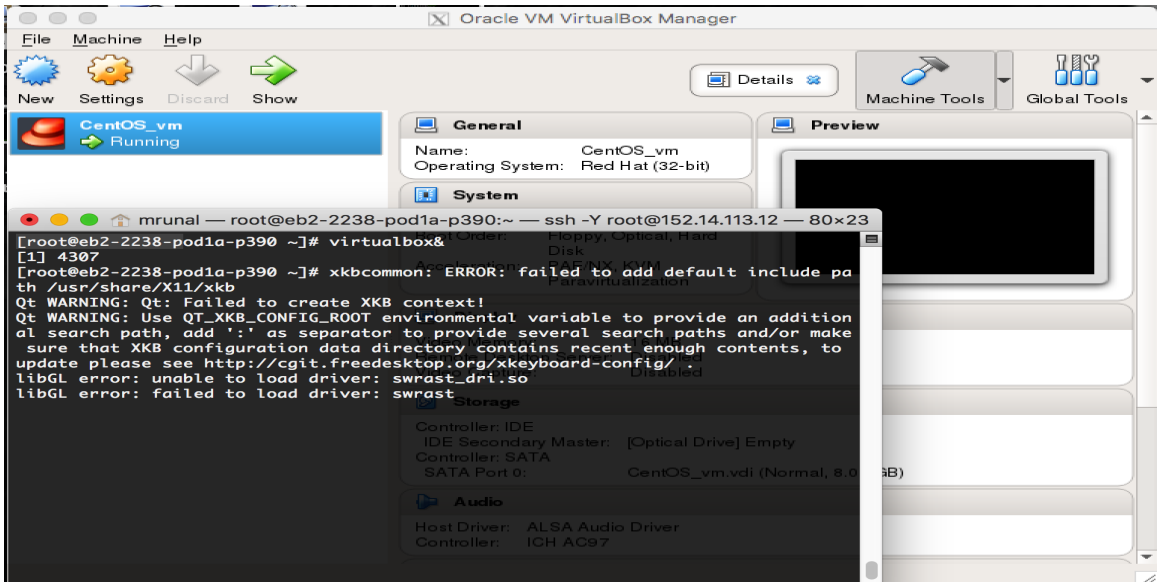


Figure 12: SSH to VM from Host Os

Test No.	Description	Expected Result	Actual Result	Requirement
T1.7	Verify VM creation and SSH connectivity	user should be able to login to the virtual machine via ssh from the host OS	User is able to login to the VM using ssh	FR1.3, FR2.3

Table 4: Test Cases (Cont.)

The following figure 13 shows that the user is able to login to the VM via ssh from the host OS.

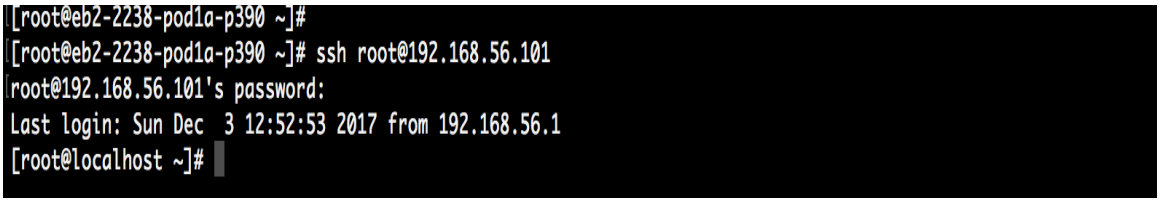


Figure 13: SSH to VM from Host Os

Test No.	Description	Expected Result	Actual Result	Requirement
T1.8	Verify CollectD client is running	Wireshark captures on client should show traffic destined to destination ports 25826 and 3030	Captured traffic destined to ports 26826 and 3030 on the client	NFR5

Table 5: Test Cases (Cont.)

The following figures 14, 15 show captures of traffic destined to ports 25826 and 3030 respectively on the client machines

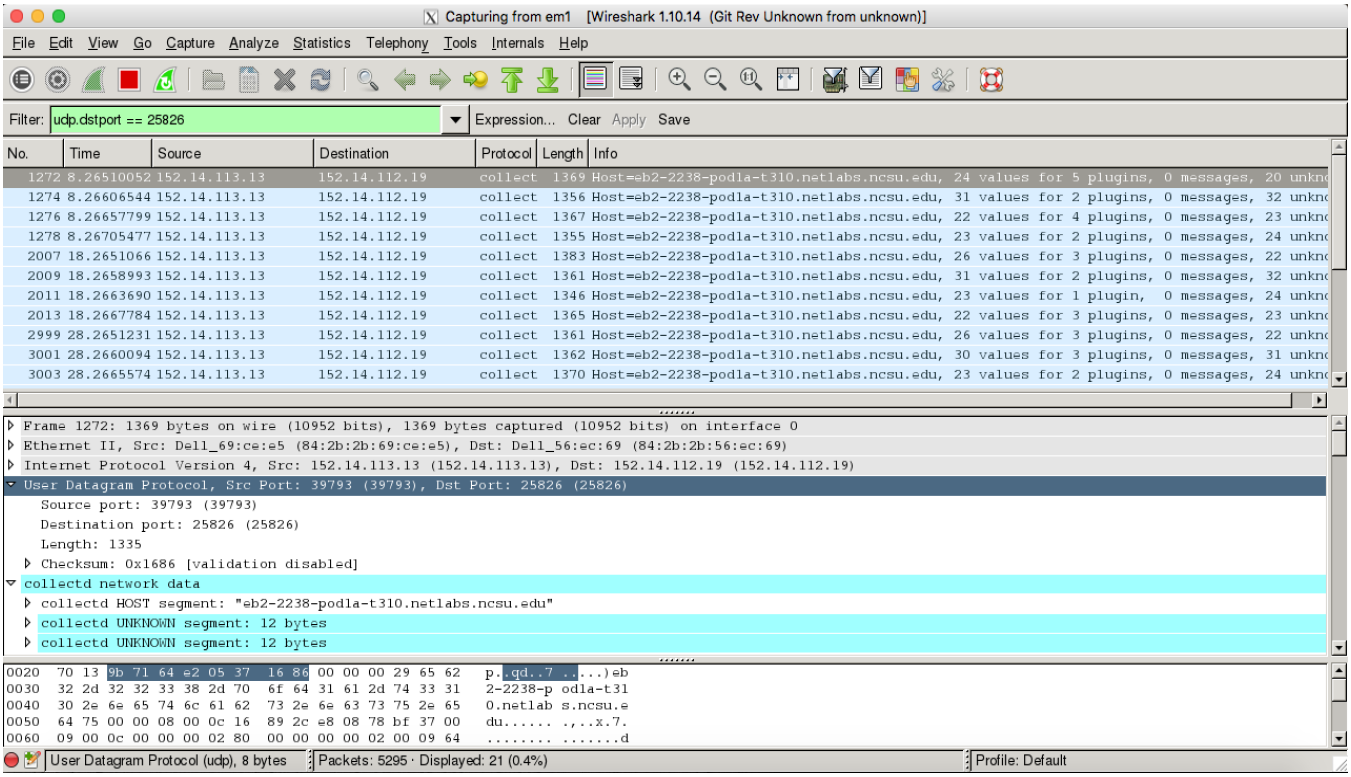


Figure 14: Capture of traffic to destination port 25826 on CollectD client

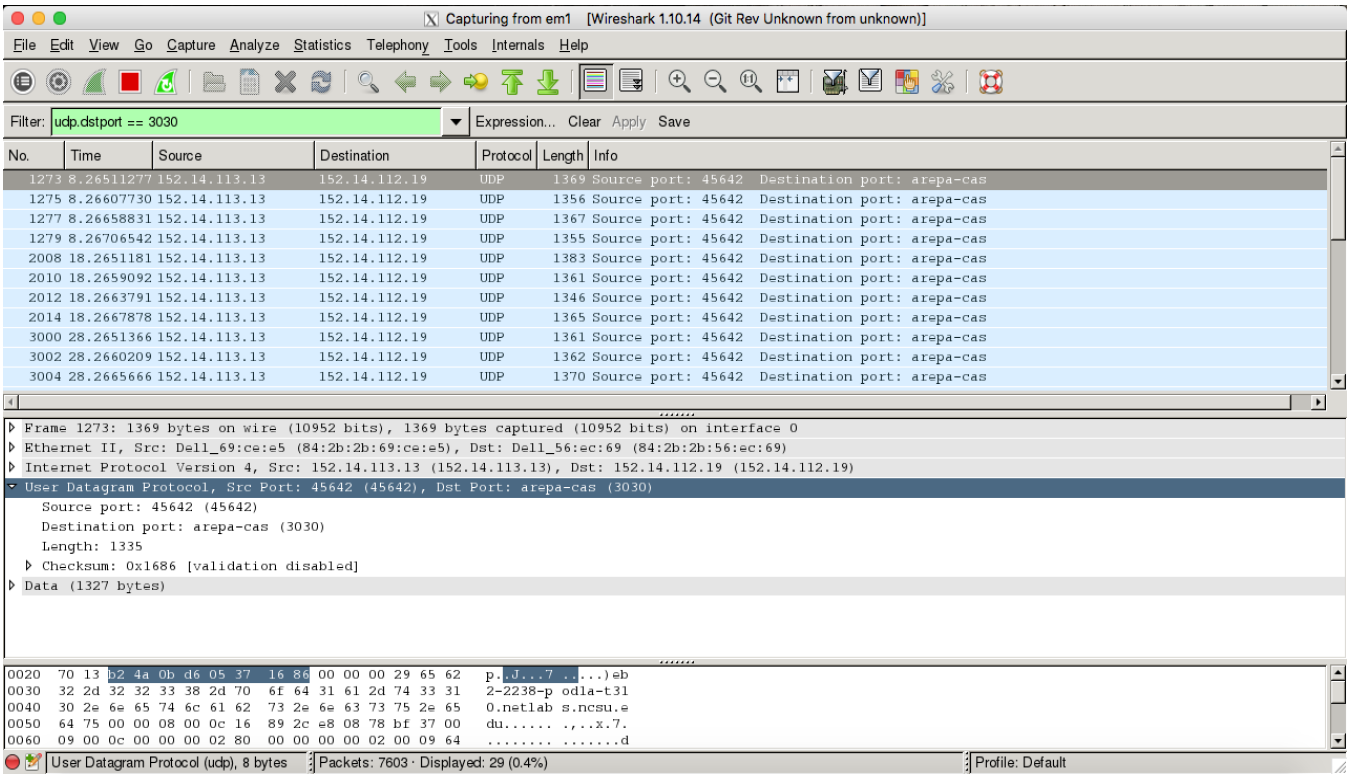


Figure 15: Capture of traffic to destination port 3030 on CollectD client

Test No.	Description	Expected Result	Actual Result	Requirement
T1.9	Verify CollectD server is running	Wireshark captures on server should show traffic destined to destination port 25826	Captured traffic destined to port 26826 on the server	NFR5

Table 6: Test Cases (Cont.)

The following figure shows a capture of traffic destined to port 25826 on CollectD server

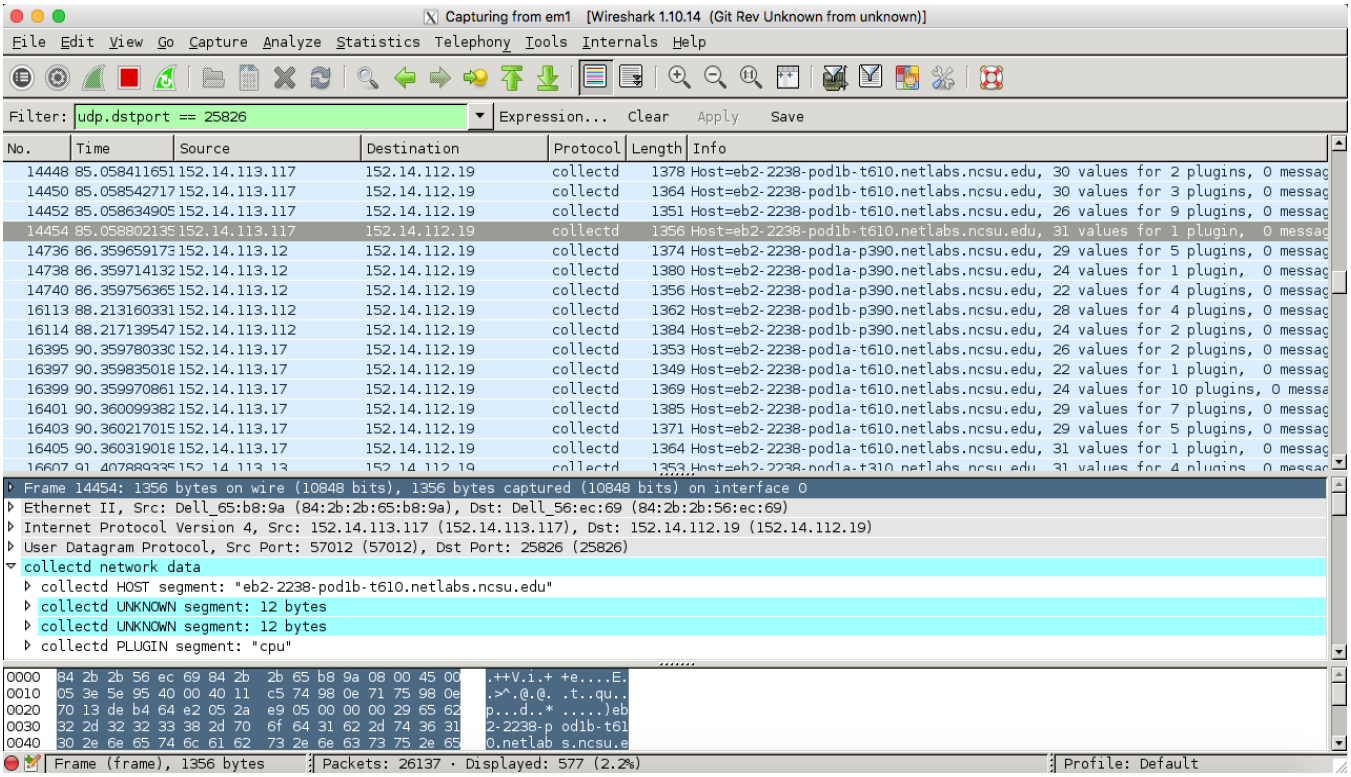


Figure 16: Capture of traffic destined to port 25826 on CollectD server

Test No.	Description	Expected Result	Actual Result	Requirement
T1.10	Verify InfluxDB server is running	Wireshark captures on InfluxDB server should show traffic destined to destination port 3030	Captured traffic destined to port 3030	NFR5

Table 7: Test Cases (Cont.)

The following figure shows a capture of traffic destined to port 3030 on the InfluxDB server

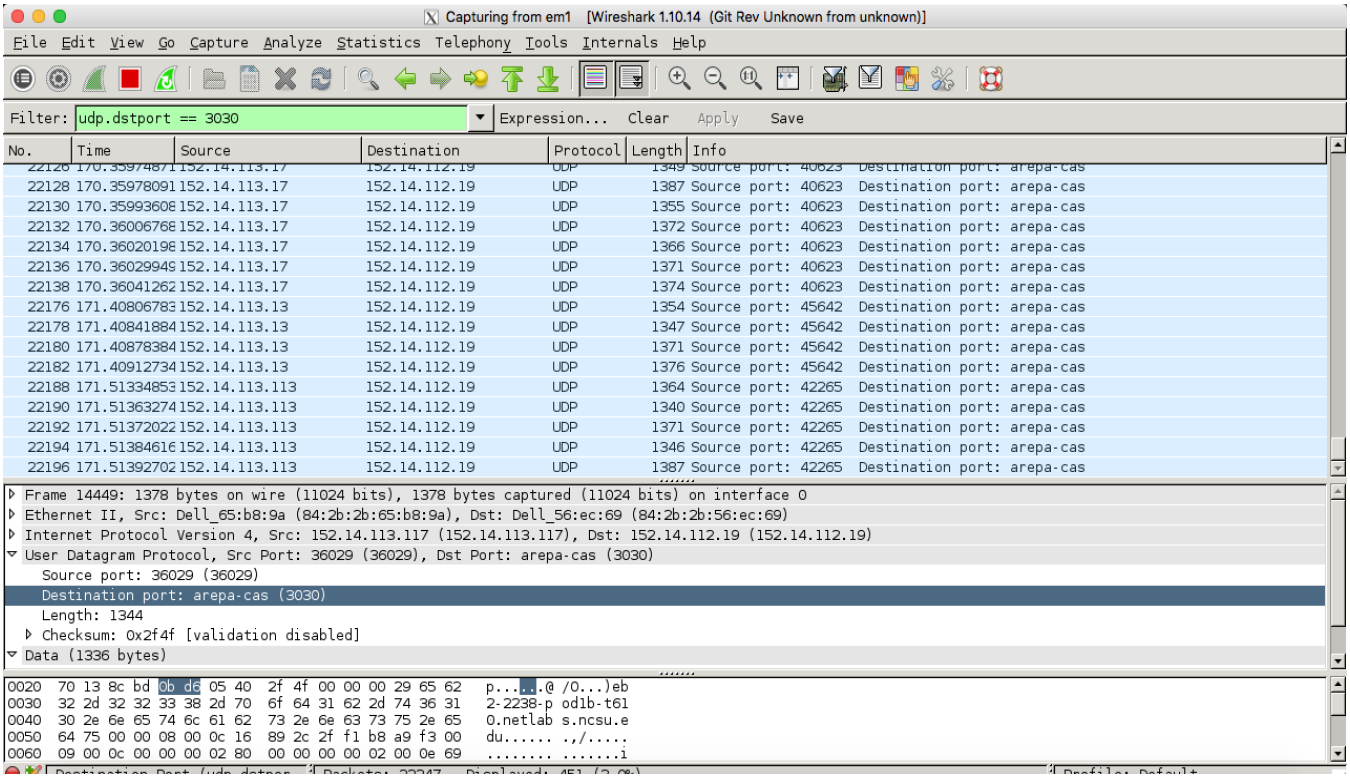


Figure 17: Capture of traffic destined to port 3030 on InfluxDB server

Test No.	Description	Expected Result	Actual Result	Requirement
T1.11	Verify grafana installation	Accessing the url http://localhost:3000/login should bring up Grafana homepage	Web page can be seen	NFR5

Table 8: Test Cases (Cont.)

The following figure 18 shows Grafana GUI.

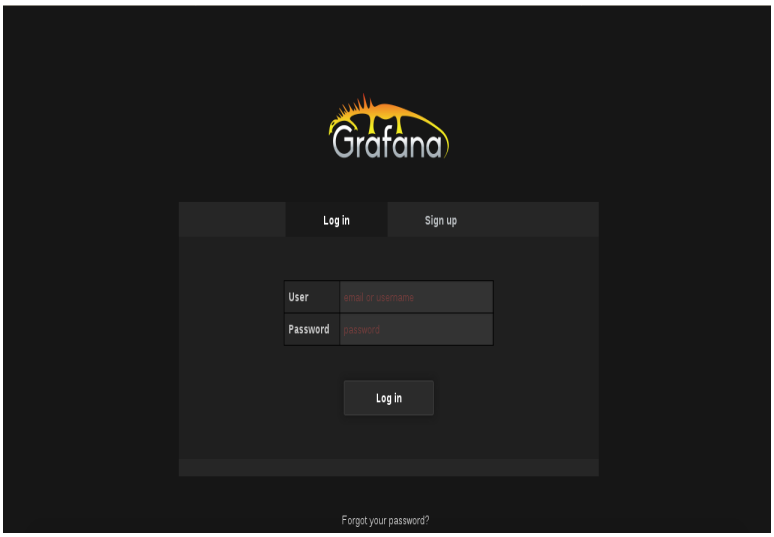


Figure 18: Grafana GUI

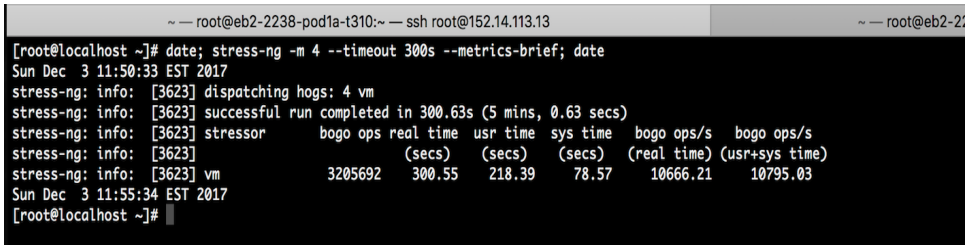
Test No.	Description	Expected Result	Actual Result	Requirement
T2.1	Running 'stress-ng' on containers and VMs to compare CPU performance	We expect the Containers to give better CPU performance than VMs	We see that the Containers do give better performance than VMs	FR3
T2.2	Running 'stress-ng' on containers and VMs to compare system load	We expect the VMs to put more load on the system than Containers	We see that Containers put more load on the system than VMs.	FR3

Table 9: Test Cases (Cont.)

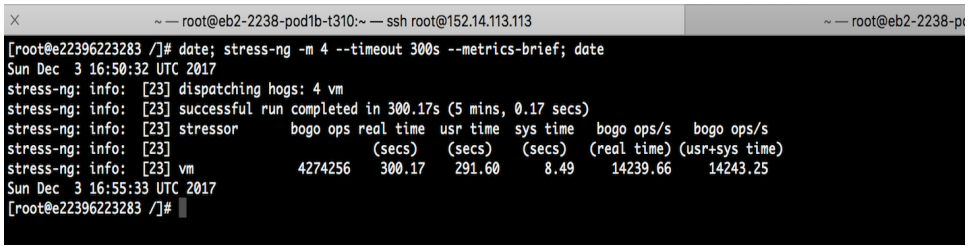
To test the comparative performance of Virtual Machines against Containers, we use the following command:

```
stress-ng -m 4 --timeout 300ms --metrics-brief
```

To measure CPU performance, we look at the floating point operations per second (FLOPS) value. The higher the FLOPS value the better the CPU performance. As we can see in the following figures 19 and 20, the containers give much better FLOPS value which implies that containers perform better with respect to the CPU performance.

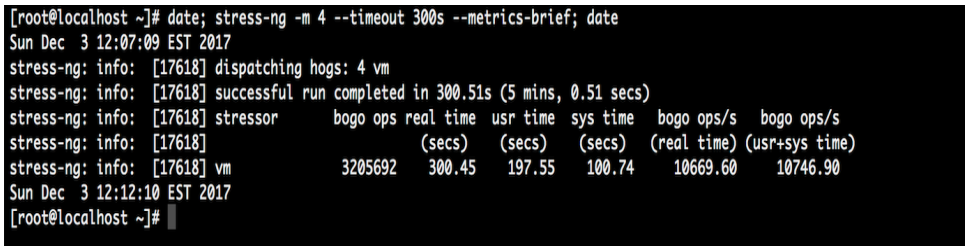


(a) VM

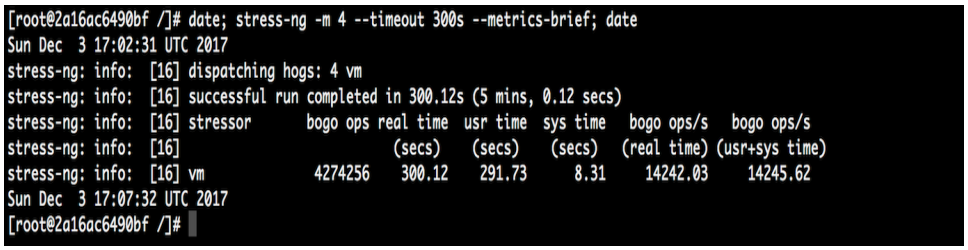


(b) Container

Figure 19: Running 'stress-ng' with 4 workers on t310



(a) VM



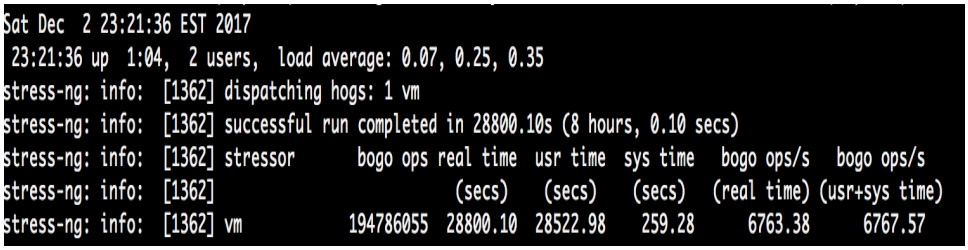
(b) Container

Figure 20: Running 'stress-ng' with 4 workers on t610

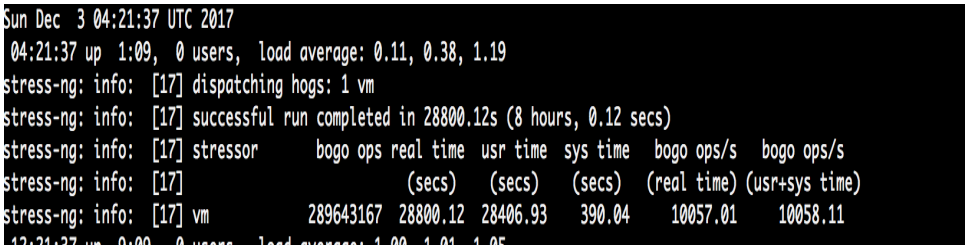
To account for long term fluctuations, we conducted a similar test for a period of 8 hours. We used the following command for a long run:

```
stress-ng -m 1 --vm-bytes 700M --timeout 8h --metrics-brief
```

We saw that the results were consistent and that the containers gave higher FLOPS over a large period of time as well. This can be seen from the figure 21.



(a) VM

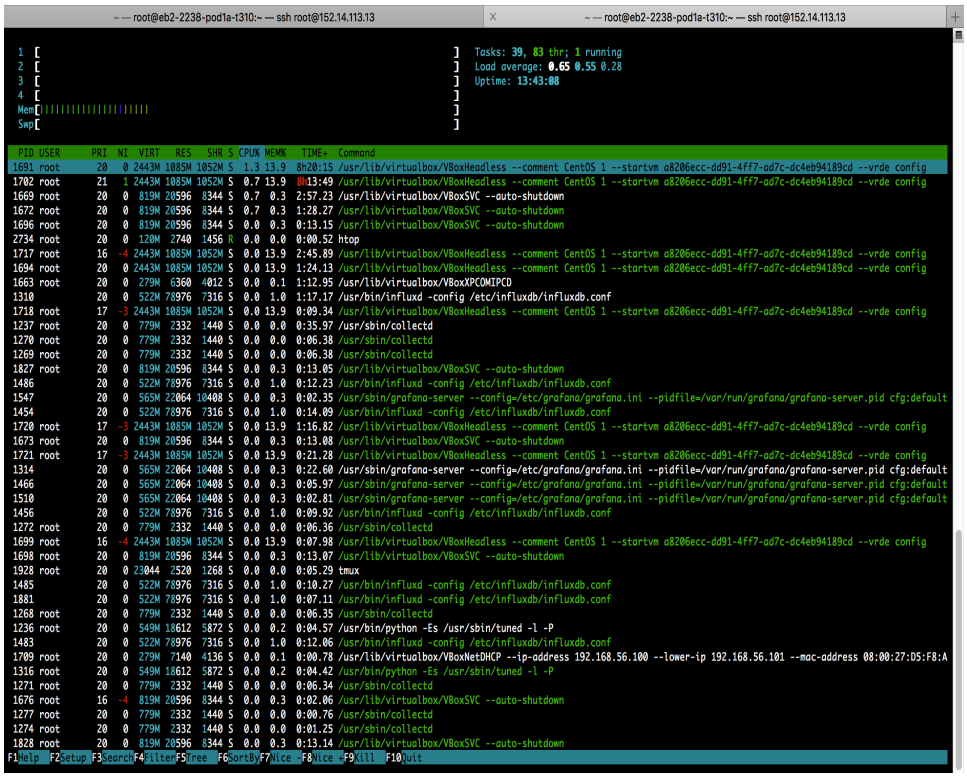


(b) Container

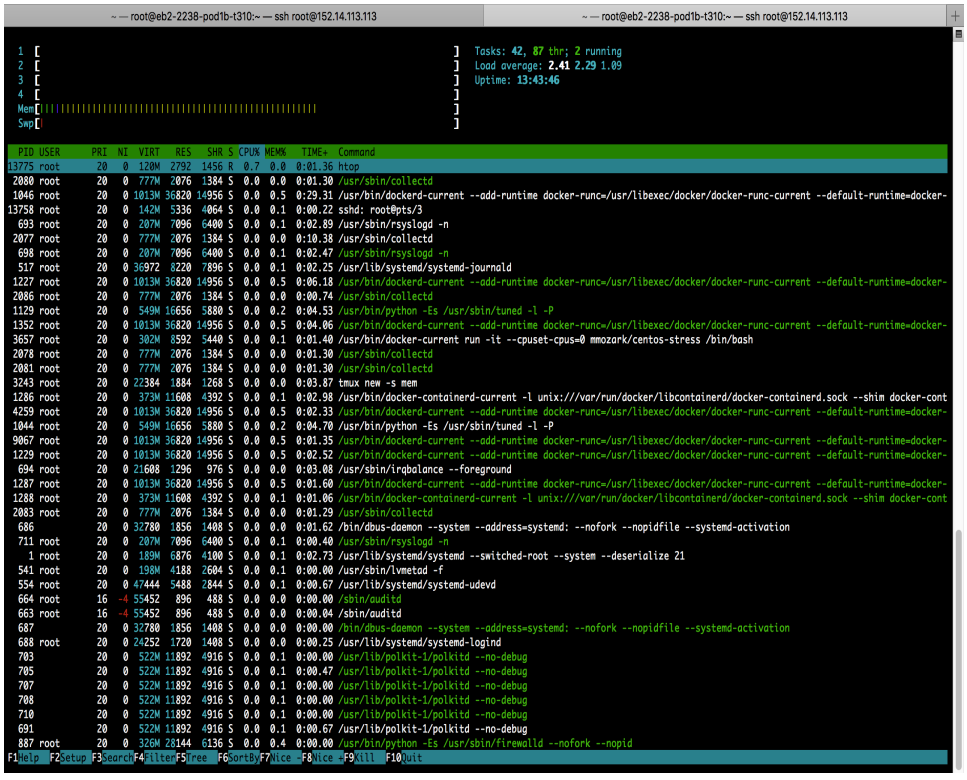
Figure 21: Display of htop just after running 'stress-ng' with 4 workers on t310

We test the system load on our system just after running the above 'stress-ng' application. We check the system load using 'uptime' command. This command displays 3 average loads for the last 1 min, 5 min and 15 mins respectively.

We expected than the VMs would put more load on the system than containers, however we unexpectedly see that the system load is much higher in the systems hosting the containers as compared to the systems hosting virtual machines. We can see the 'uptime' values in figures 22 and 23

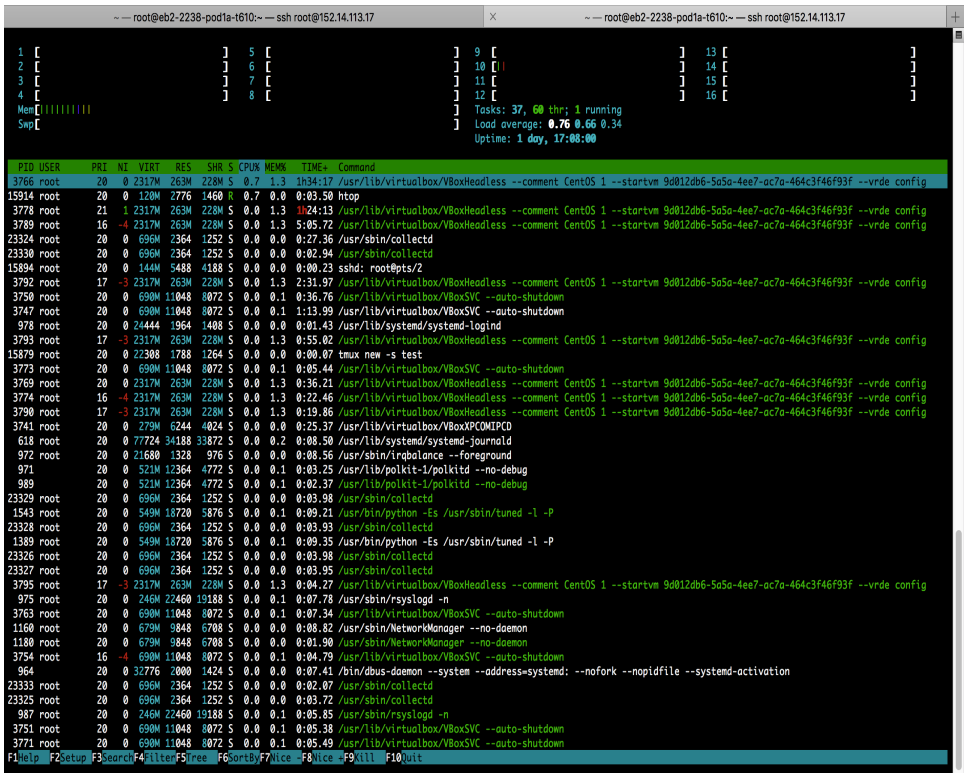


(a) VM

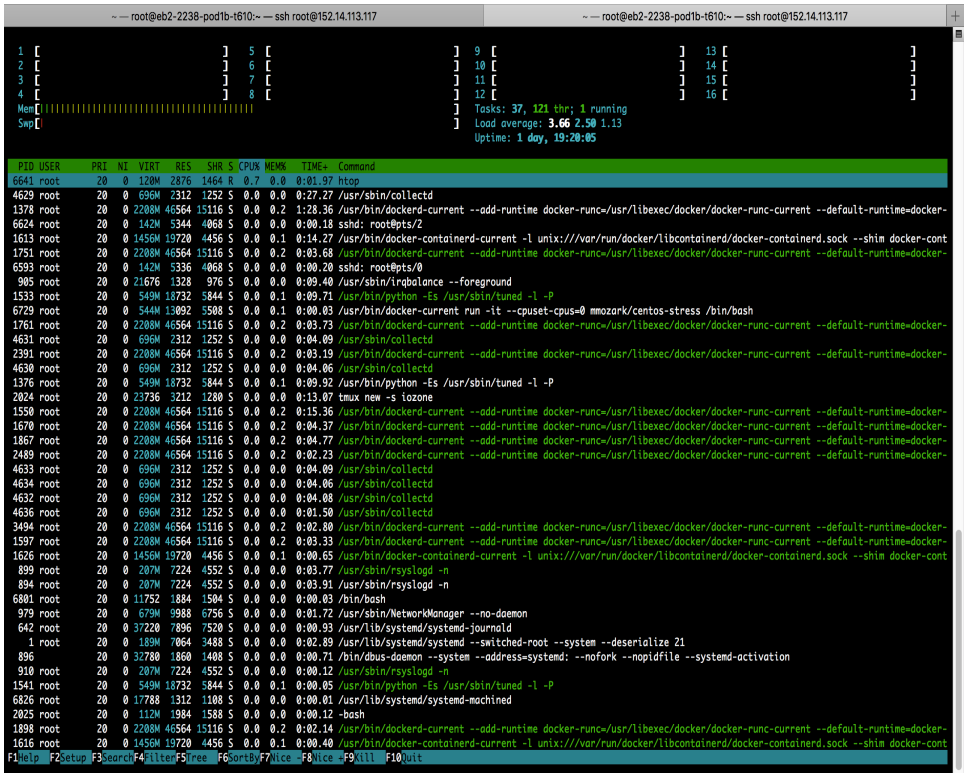


(b) Container

Figure 22: Display of htop just after running ‘stress-ng’ with 4 workers on t310



(a) VM



(b) Container

Figure 23: Display of htop just after running 'stress-ng' with 4 workers on t610

Test No.	Description	Expected Result	Actual Result	Requirement
T2.3	Running 'bonnie++' to compare disk i/o read-/write performance as well as i/o seek operation	We expected the containers to perform i/o read/write operations faster than the VMs	We see that the VMs give consistently better performance for i/o read and write as well as i/o seek operations.	FR3
T2.4	Running 'bonnie++' to compare disk CPU performance at a more granular level	Based on the previous result, we expected the containers to give better CPU performance than containers all the time	We see that VMs give better CPU performance for sequential input and output. And for others like random seeks and sequential as well as random create, read and delete, we see the containers give better CPU performance as expected	FR3

Table 10: Test Cases (Cont.)

We ran the same command as before, that is:

```
bonnie++ -u root -r NB
```

This time the focus is on the metrics value that is displayed by the command itself. We see that the i/o read, write and seek operations are consistently faster on VM than container.

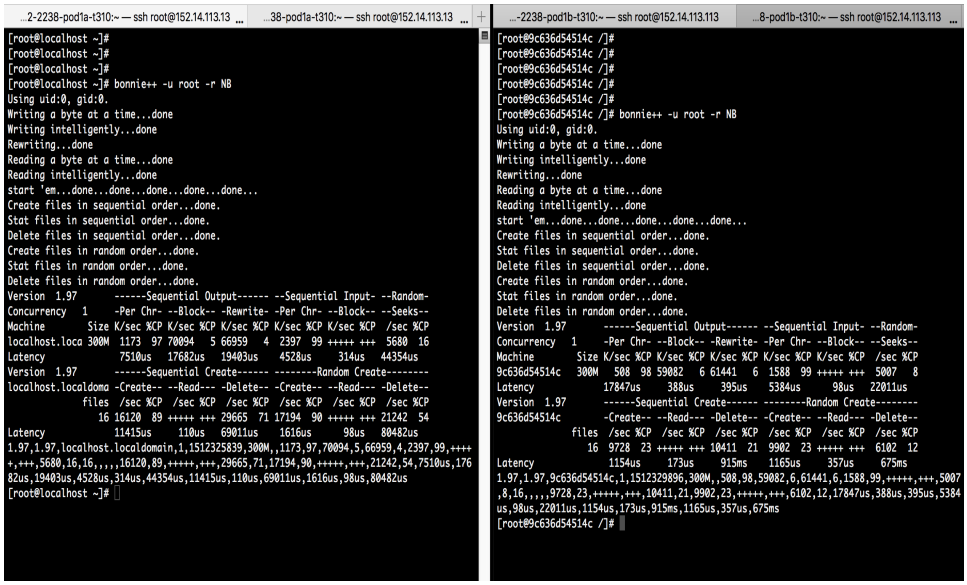


Figure 24: Bonnie++ Metrics Output

Now, moving onto the CPU utilization. The values displayed under %CP is the amount of CPU being utilized. Thus, based on the values, we can give a more granular CPU performance of CPU on VMs and containers. We also have an option to save the results as an Excel file and view the results graphically.

We see that VMs perform slightly better in case of sequential inputs and outputs. However, Containers give better CPU utilization in case of random seeks as well as sequential and random read, create and delete operations.

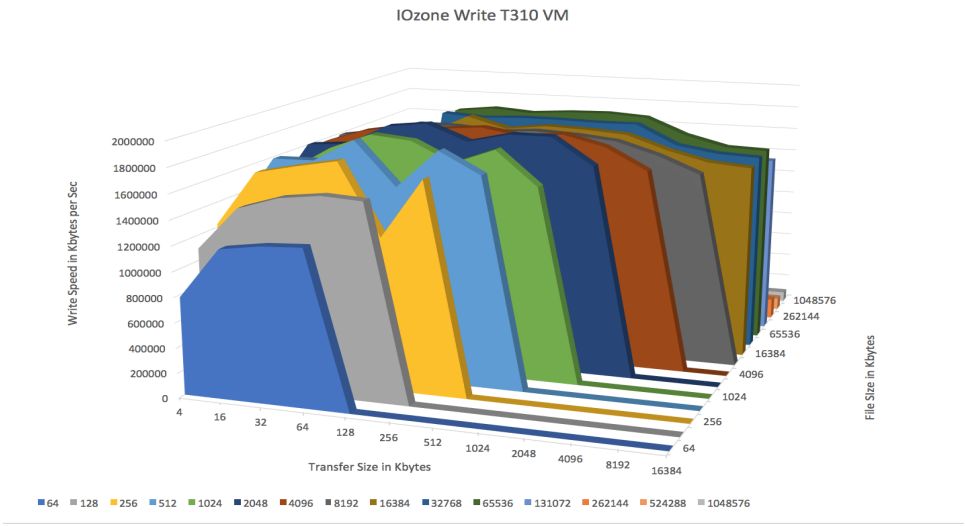
Test No.	Description	Expected Result	Actual Result	Requirement
T2.5	Running IOzone on containers and VMs to compare File Read and Write speeds	We expect the containers to have higher Read and Write speeds	We observe that VMs have generally higher Read and Write speeds as compared to containers Result	FR3
T2.6	Running IOzone on containers and VMs to compare CPU utilization for File Read and Write operations	We expect the containers to consume less CPU resources for Read, Write operations since they are lightweight	We observe that containers have low CPU usage whereas VMs have high CPU usage for Read and Write operations	FR3

Table 11: Test Cases (Cont.)

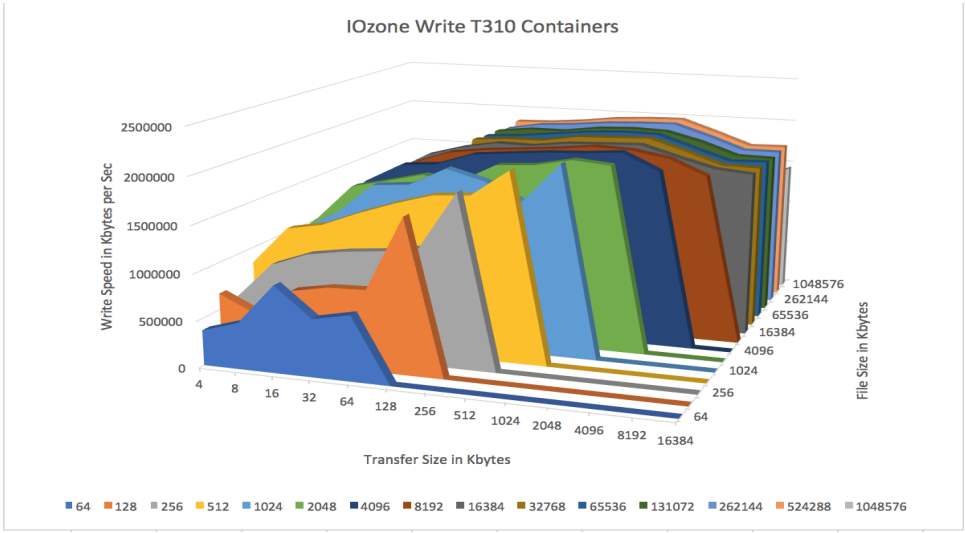
In order to test the File Read-Write speeds and CPU Utilization of Virtual Machines against Containers, we use the following command:

```
iozone -a -g 1g -+u -b output_filename.xls
```

This command would perform all 13 essential IOzone tests from Read, Write, Sequential Read, Sequential Writes, etc. for file sizes from 64 Kbs to 1 Gb with varying file transfer sizes. It would then measure the Read, Write speeds and corresponding CPU utilization. We also have an option to save the results in an Excel spreadsheet so that we can view the results later graphically. The results for all the IOzone tests can be observed in the below graphs.



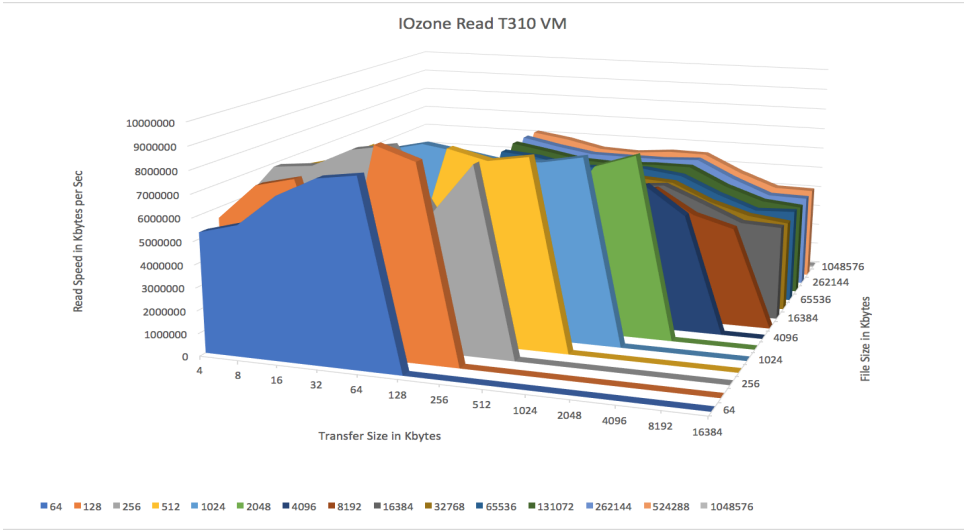
(a) VM



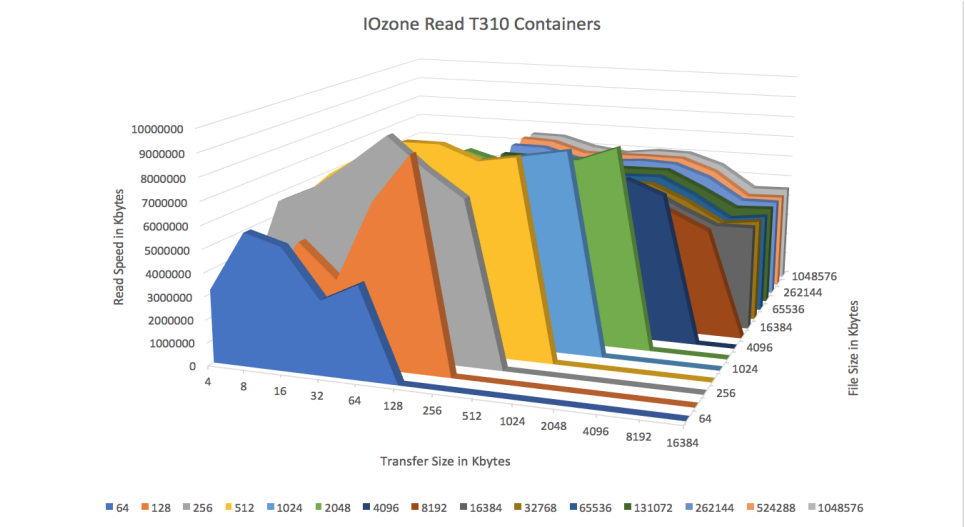
(b) Container

Figure 25: Running 'iozone' for write operations on t310

The above figures show that the Write speeds are generally higher for VMs as compared to containers. Although for some instances of file size and file transfer chunk size, the Write speeds are more for containers.



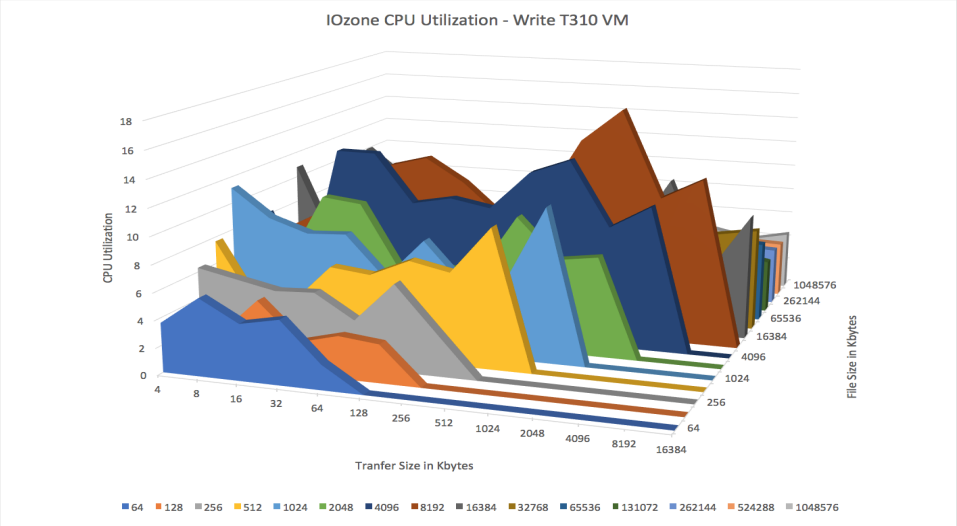
(a) VM



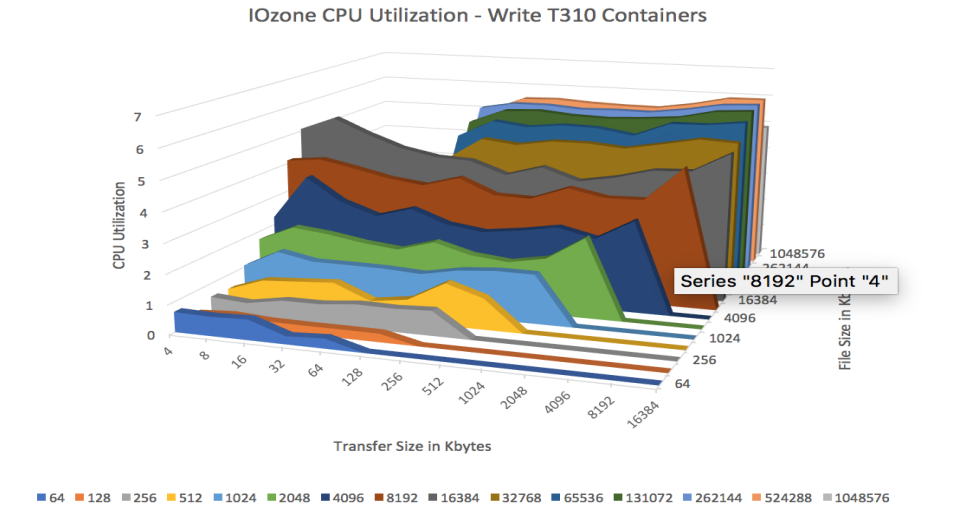
(b) Container

Figure 26: Running 'iozone' for read operations on t310

The above figures show that the Read speeds are also generally higher for VMs as compared to containers. Although for some instances of file size and file transfer chunk size, the Read speeds are more for containers.



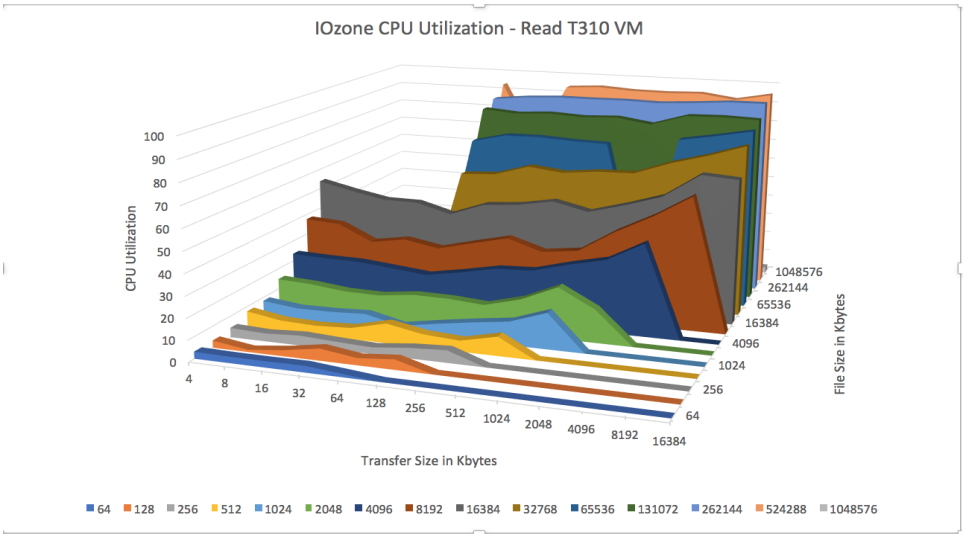
(a) VM



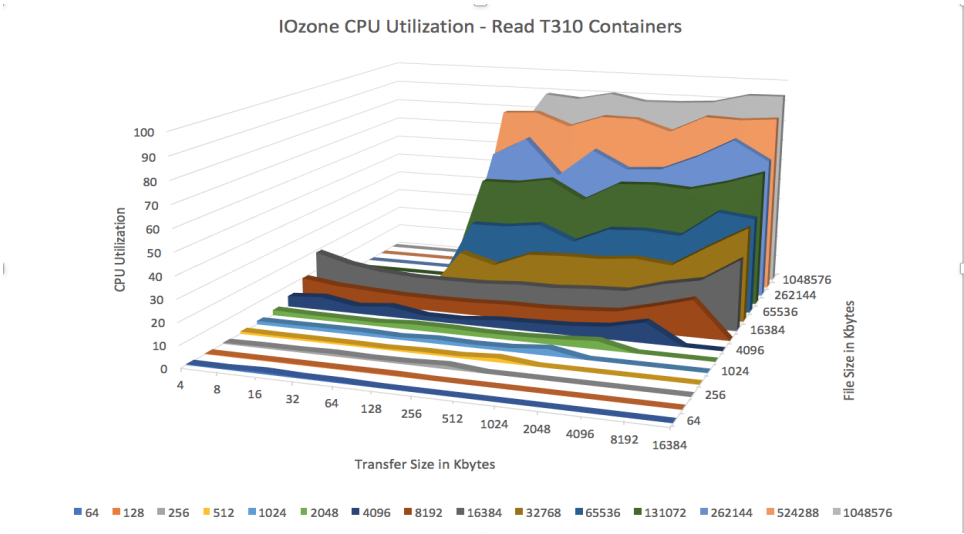
(b) Container

Figure 27: Running 'iozone' for CPU utilization of write operations on t310

The above figures show that the CPU usage for Write operations are generally more for VMs as compared to containers. This also explains why the Write speeds are observed to be more for VMs as compared to containers.



(a) VM



(b) Container

Figure 28: Running 'iozone' for CPU utilization of read operations on t310

The above figures show that the CPU usage for Read operations are also generally higher for VMs as compared to containers. This would also explain for the higher Read speeds that are observed for VMs as compared to containers.

On aggregating the above test cases for Bonnie++ and IOzone applications (Test Cases T2.4, T2.5, T2.6, T2.7), we can conclude that - VMs generally have better performance in terms of Read and Write operations as compared to containers. Furthermore, VMs also have higher CPU utilization for disk operations (excluding some operations) while containers have lower CPU utilization values. This explains high read-write speeds in VMs and low read-write speeds in containers. This is also evident from the fact that applications running on VMs have access to virtual host resources, therefore they can utilize more CPU cycles and achieve better read-write speeds. Whereas applications running on containers are isolated and have restricted access to resources, therefore, they tend to have lower read-write speeds. [21]

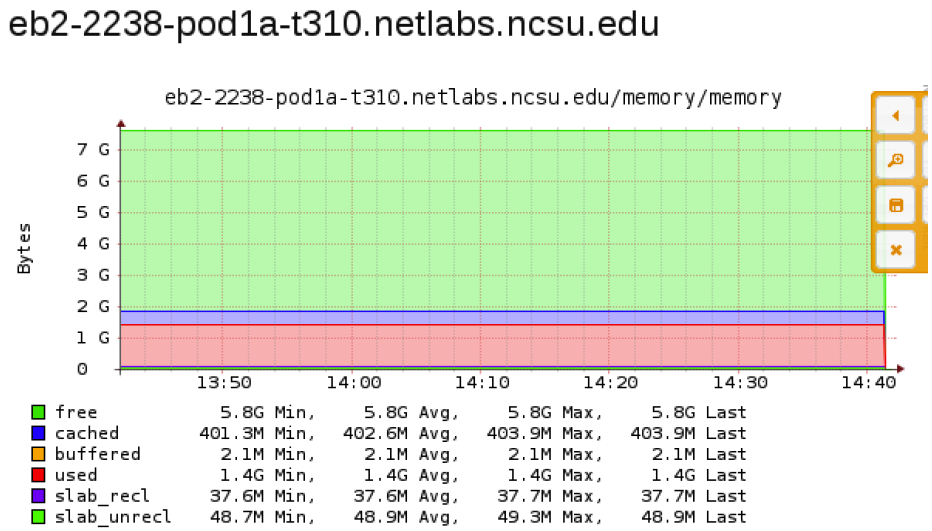
Test No.	Description	Expected Result	Actual Result	Requirement
T2.7	Compare memory utilization when one instances of VM is running vs one instance of container is running as well when multiple instances of VMs are running against multiple instances of containers are running	We expect containers to utilize less memory than VMs	We notice than for multiple VMs memory utilization is higher than the containers, but for just one instance we notice higher memory utilization by containers	FR3
T3.1	Scalability (Number of containers/VMs that can be run on one system)	We expect the containers to be highly scalable whereas the scalability of VMs is expected to be limited	We find that the VMs are less scalable than container	NFR2

Table 12: Test Cases (Cont.)

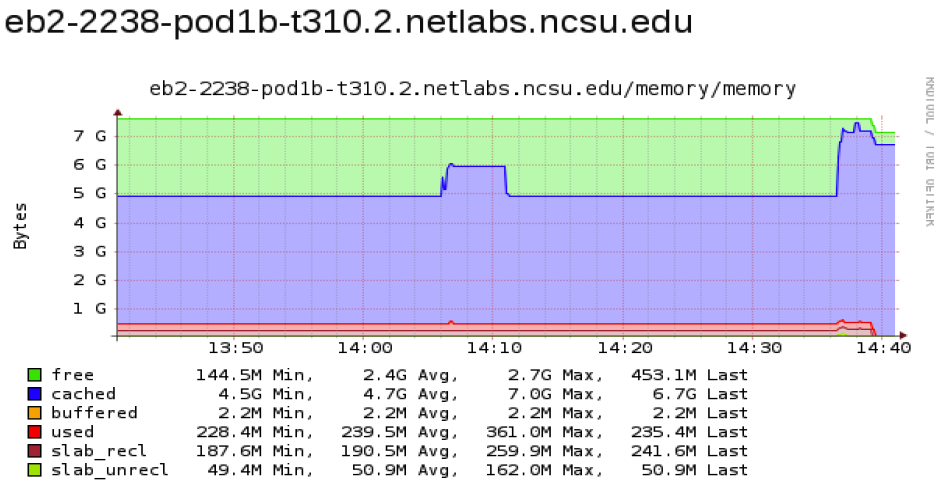
While testing for just one instance of VM and container, we also ran the following command to stress the memory utilization

```
bonnie++ -u root -r NB
```

We see from the fig 29 that the memory utilization on the OS hosting Virtual Machine stays constant whereas the one hosting container machine we see reduction in the amount of free memory. The overall memory utilization by container is seen to be higher in this case.



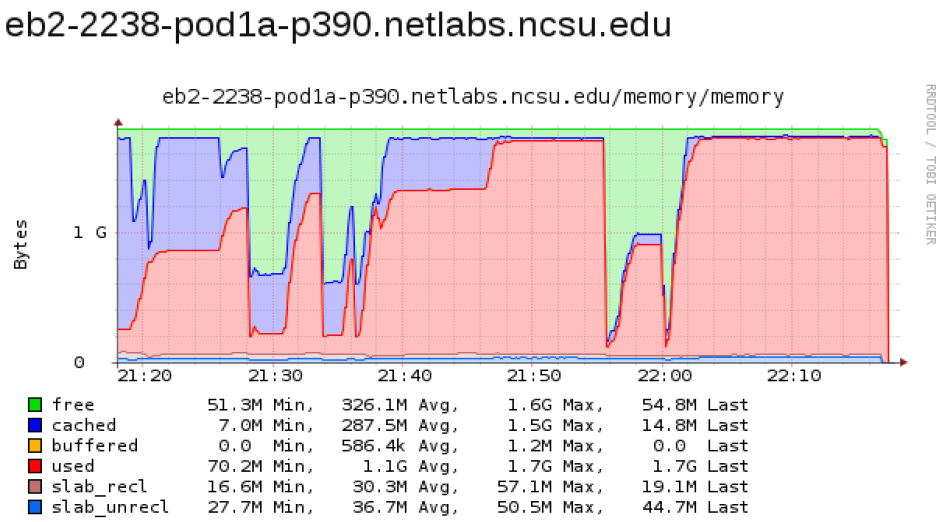
(a) VM



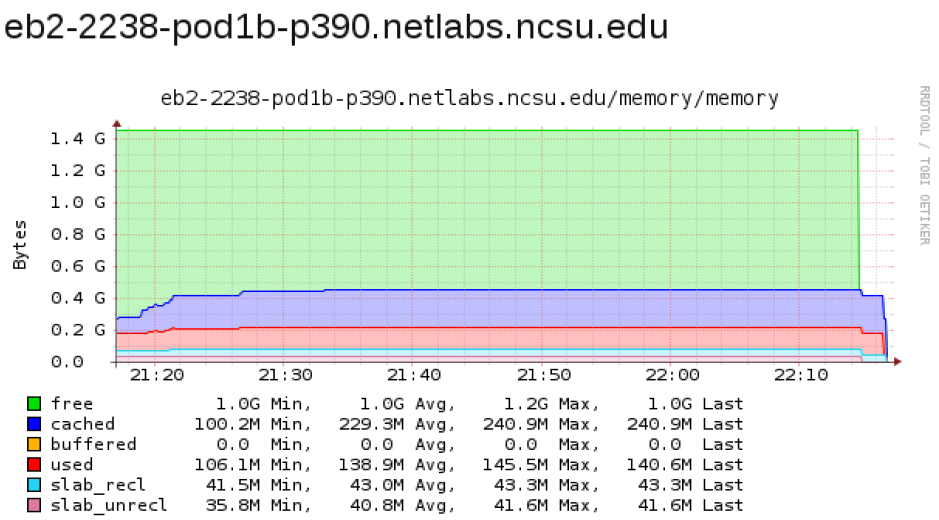
(b) Container

Figure 29: Memory utilization graph just after performing 'bonnie++' operation

Next, we ran six VMs on Pod1A-p390 machines and noticed the memory utilization increased drastically. Whereas, we have around thirteen containers running on the Pod1B-p390 and memory utilization is seen to be reasonable. It seems like the containers adjust the memory utilization across each instance of container to maintain a decent level of total memory utilization. This can be seen in following figures 30a and 30b.



(a) Memory utilization when 6 VMs are running



(b) Container

Figure 30: Memory utilization when 13 containers are running

Test No.	Description	Expected Result	Actual Result	Requirement
T3.2	Ease of deployment	We expect the deployment of containers to be easier and faster than that of VMs	Initial installation of Docker container took 4-5 mins as compared to VMs which took 20-25mins	NFR1

Table 13: Test Cases (Cont.)

The time taken to set up and install a container is much less than a virtual machine, this roots back to the reason that containers do not perform any emulation of the hardware, nor do they boot an OS. Containers simply provide an isolated environment to run an application, without worrying about the underlying hardware. Thus, we expect Containers to perform better.

Test No.	Description	Expected Result	Actual Result	Requirement
T3.3	Flexibility	We expect Virtual Machines to be more flexible than containers. Containers restrict the choice of OS for a user, if the application is not supported for the underlying	VMs are seen to be more flexible than containers as expected	NFR3

Table 14: Test Cases (Cont.)

We can install different types of VMs on the same host OS. However, with containers, the container cannot be run on a different OS base. For example, we can run linux as well as windows VM on the linux host OS as seen in figure 31. However, since the container utilize the system calls of the underlying host OS, the guest OS needs to be compatible with the host OS [22].

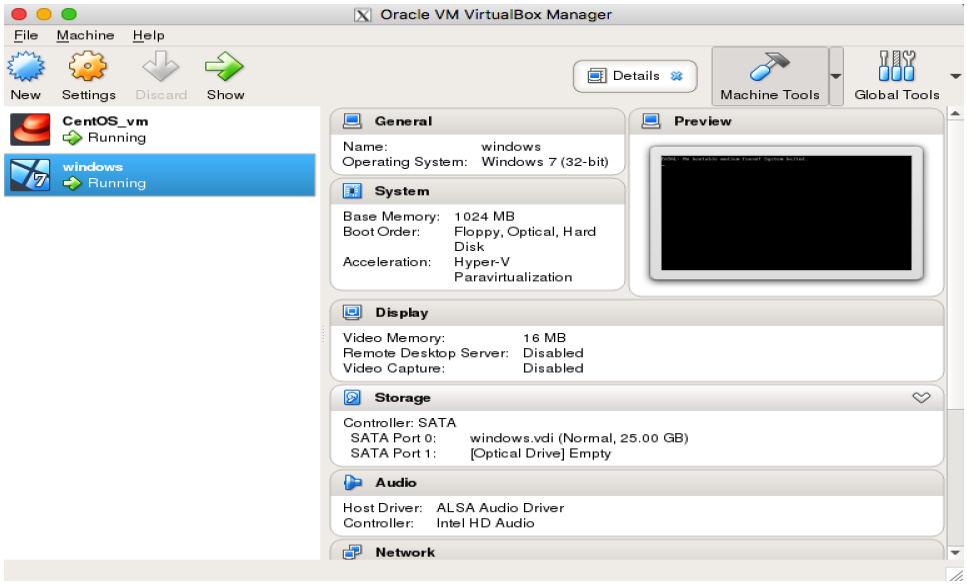


Figure 31: Multiple different types of VMson linux machine

Test No.	Description	Expected Result	Actual Result	Requirement
T3.4	Automation of Benchmark Testing	Performing benchmark tests for different performance metrics on all hardware machines should be simple and user friendly	By implementing a simple front-end interface which automates the triggering of benchmark tests on VM and container machines, users need not ssh into individual machines and manually start them.	NFR4

Table 15: Test Cases (Cont.)

In order to execute any benchmark test, the python front end application can be run in a local machine and the desired performance test can be started on a VM or a container.

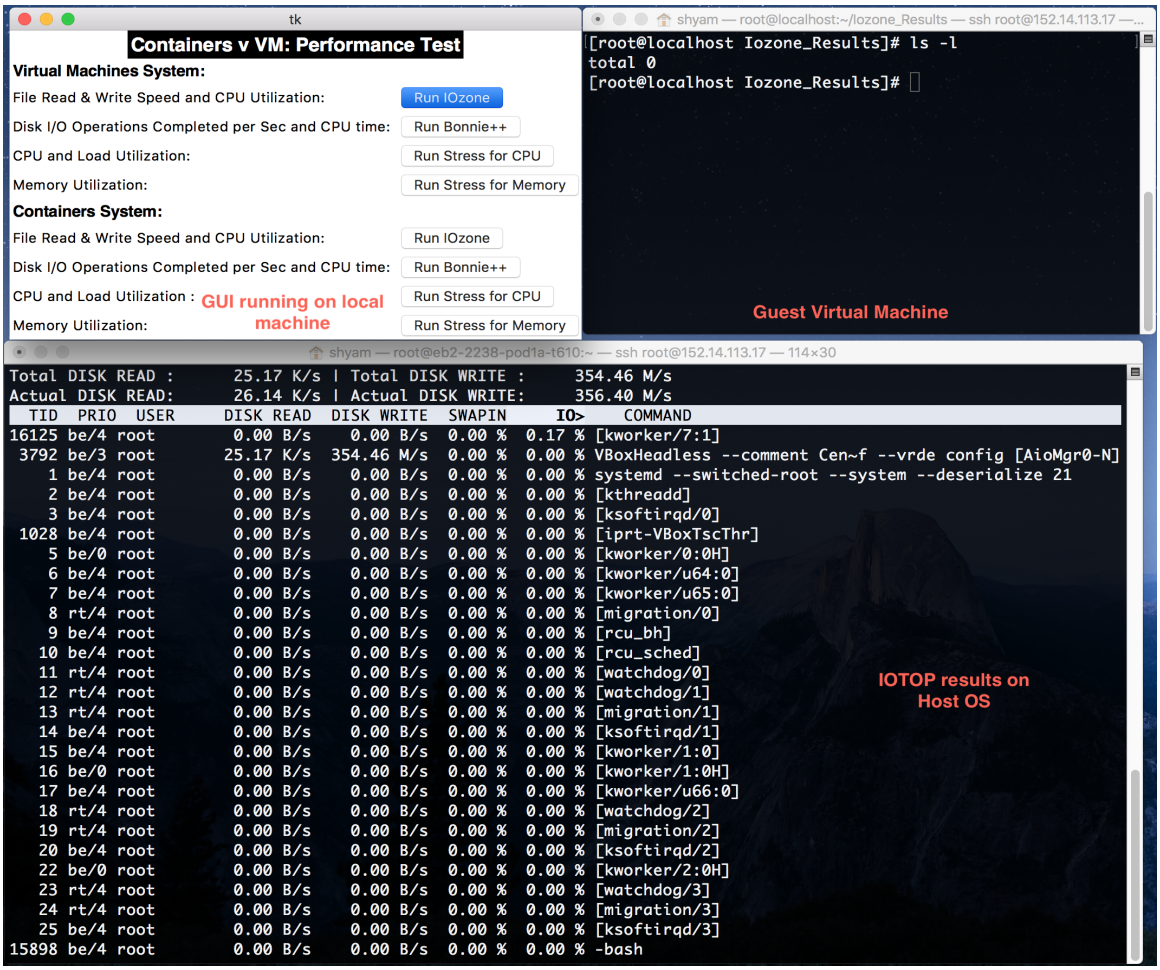


Figure 32: Automated Testing using Front End GUI

In the above figure, the IOzone benchmark test has been triggered on the GUI application running on a local machine. Using 'iotop' command's statistics, we can observe the disk read, write values on Host machine, which shows that the test application is currently running on the guest virtual machine.

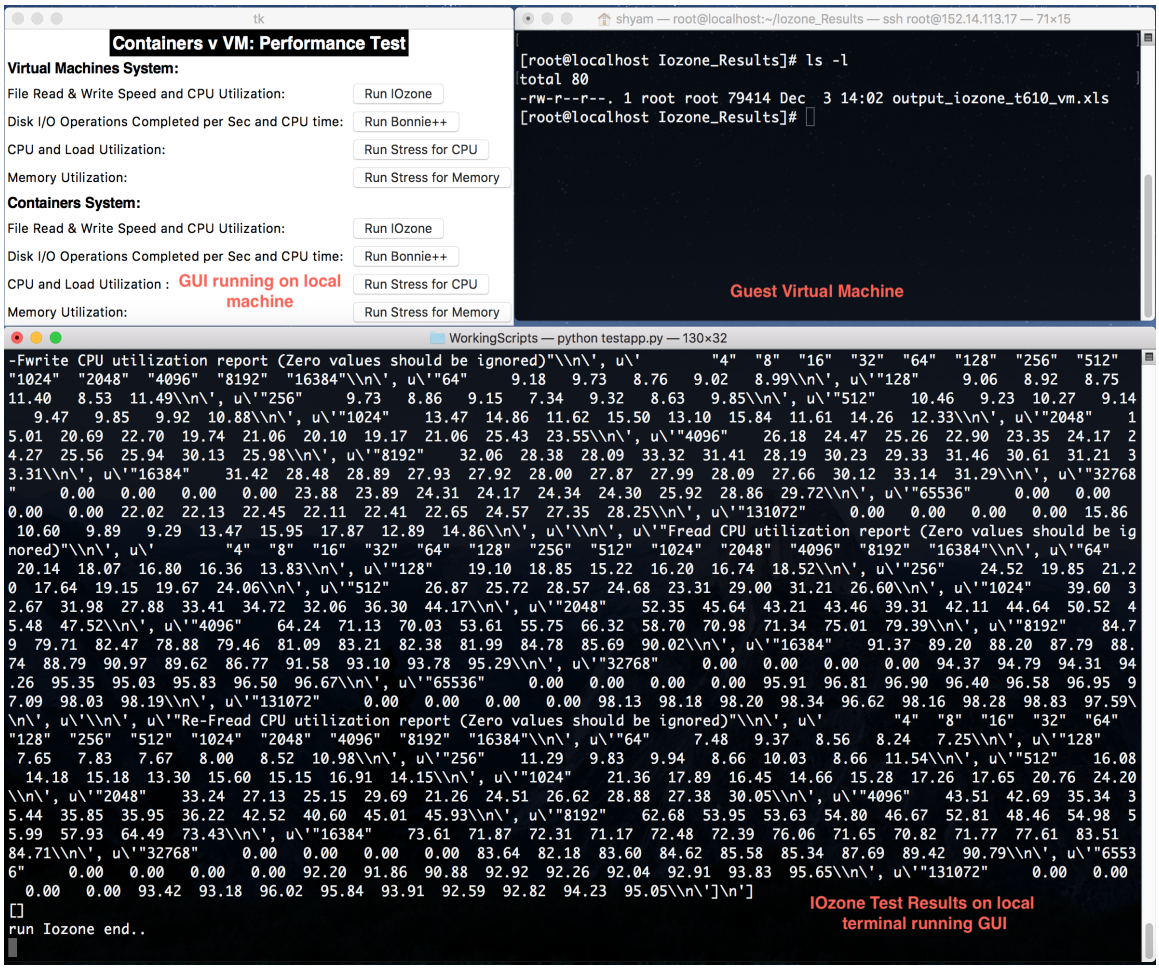


Figure 33: Automated Test Results

After test completion, we can observe the IOzone test results on the local machine terminal running the GUI application. Moreover, an excel spreadsheet containing the IOzone test results can also be found on the guest virtual machine. Therefore we observe that we have enabled users to trigger benchmark applications automatically, without needing manual effort.

7 Schedule and Personnel

No.	Tasks	Owner	Contributors	Start Date	End Date
1	Understanding Container Technology	Team		10/17/17	11/03/17
2	Understanding VM Technology	Team		10/17/17	11/03/17
3	Researching Topology Design	Team		11/04/17	11/06/17
4	Researching Compute intensive applications	Jaspreet	Shyam	11/8/17	11/9/17
5	Researching I/O intensive applications	Lahari	Abhimanyu	11/9/17	11/10/17
6	CoreOS(host) installation for Container System	Mrunal	Shyam, Jaspreet	11/13/17	11/14/17
7	CentOS (host) installation for VM System	Abhimanyu	Lahari, David	11/13/17	11/14/17
8	Docker Setup	Shyam	Mrunal, Jaspreet	11/15/17	11/15/17
9	VirtualBox Setup	Lahari	David,Abhimanyu	11/15/17	11/15/17
10	Container Setup for individual Applications	Jaspreet	Shyam	11/15/17	11/29/17
11	CentOS VM(guest) installation for VM System	Abhimanyu	Lahari, David	11/16/17	11/17/17
12	Installing Applications on CentOS VM	David		11/18/17	11/18/17
13	CentOS (host) installation for container System	David	Abhimanyu	11/28/17	11/28/17
14	Setup CollectD clients and server	Lahari	Abhimanyu	11/29/17	11/30/17
15	Setup InfluxDB server	Abhimanyu	Lahari	11/29/17	11/30/17
16	Setup Grafana dashboard and add data sources	Abhimanyu	David	12/01/17	12/01/17
17	Setup CollectD-Web GUI	Mrunal	Jaspreet	12/01/17	12/01/17
18	Perform Stress and Stress-ng tests	David	Mrunal	12/02/17	12/03/17
19	Perform IOZone test	Mrunal		12/02/17	12/03/17
20	Perform Bonnie++ test	Shyam		12/02/17	12/03/17
21	Verify and Validate Test Results	Jaspreet	Mrunal	12/03/17	12/03/17
22	Front-end application to run tests	Shyam		12/02/17	12/03/17
23	Document and compile report	Lahari	Team	11/20/17	12/03/17

8 Results and Future Scope

The above tests show that both Virtual Machines and Containers perform fairly well depending upon the scenario. In general, we observed that VMs performed better when the metrics under consideration was system load and disk i/o whereas containers performed better with respect to CPU and memory utilization. Moreover, containers are easier to deploy and highly scalable even on limited resources. On the other hand, VMs are much more flexible. These test cases can form the base on which advanced test cases like constraints on memory and other system resources, can be implemented. Our test cases along with any advanced test cases can enable a cloud computing provider and customer in obtaining a detailed insight while selecting a particular XaaS.

References

[1] Anthony Sostre. Containers vs Virtual Machines (VMs): Is There a Clear Winner?, May 2016. Available at https://www.serverpronto.com/spu/2016/05/containers-vs-virtual-machines-vms-is-there-a-clear-winner/.

[2] Eric SILLSb Patrick DREHER, Mladen A. VOUKb and Sam AVERITTb. Evidence for a cost effective cloud computing implementation based upon the nc state virtual computing laboratory model.

[3] A Beginner-Friendly Introduction to Containers, VMs and Docker. Available at https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b.

[4] About. Available at https://alpinelinux.org/about.

[5] Download centos. Available at https://www.centos.org/download.

[6] https://coreos.com/. Availabe at https://coreos.com/.

[7] Virtual box for windows vboxmanage and interface names, Jul 2013. Available at http://discoposse.com/2013/07/13/virtualbox-for-windows-vboxmanage-and-interface-names/.

[8] Functional requirements. Available at https://en.wikipedia.org/wiki/Functional_requirement.

[9] Non-functional requirement. Available at http://users.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html.

[10] Iozone. Available at https://linux.die.net/man/1/iozone.

[11] Bonnie++. Available at https://www.linux.com/news/using-bonnie-filesystem-performance-benchmarking.

[12] Bonnie++. Available at https://www.coker.com.au/bonnie++/.

[13] Stress. Available at https://www.hecticgeek.com/2012/11/stress-test-your-ubuntu-computer-with-stress.

[14] Stress-ng. Available at http://kernel.ubuntu.com/~cking/stress-ng.

[15] Virtualbox. Available at https://www.virtualbox.org.

[16] Virtualbox installation. Available at https://tecadmin.net/install-oracle-virtualbox\-on-centos-redhat-and-fedora/#.

[17] Monitoring hosts with collectd, influxdb and grafana. Available at https://jansipke.nl/monitoring-hosts-with-collectd-influxdb-and-grafana/.

[18] collectd the system statistics collection daemon. Available at https://collectd.org/.

[19] Influxdb. Available at https://en.wikipedia.org/wiki/InfluxDB.

[20] Grafana. Available at https://wiki.opennms.org/wiki/Grafana.

[21] Disk performance. Available at http://www.electronicdesign.com/dev-tools/what-s-difference-between-containers-and-virtual-machines.

[22] Docker community forums, Feb 2016. Available at https://forums.docker.com/t/can-i-run-windows-server-container-on-linux-host-operating-system/6713/2.

[23] Iozone installation. Available at http://www.thegeekstuff.com/2011/05/iozone-examples.

[24] List of iozone common commands. Available at http://www.iozone.org/docs/IOzone_msword_98.pdf.

[25] List of bonnie++ common commands. Available at https://manpages.debian.org/jessie/bonnie++/bonnie++.8.en.html.

[26] List of stress common commands. Available at https://www.hecticgeek.com/2012/11/stress-test-your-ubuntu-computer-with-stress.

[27] List of stress-ng common commands. Available at http://kernel.ubuntu.com/~cking/stress-ng.

Appendices

A XaaS Appendix

Since, in this project, we are concerned with Infrastructure-as-a-Service and Containers-as-a-Service, we limit our discussion to these two types of services in this appendix.

Infrastructure-as-a-Service (IaaS) is one of the service-oriented cloud computing model. It is a service model that delivers computer infrastructure on an outsourced basis to support enterprise operations. Typically, IaaS provides hardware, storage, servers and data center space or network components; it may also include software. In may instances, it is often called as Hardware as a Service. An IaaS provider provides policy-based services and is responsible for housing, operating and maintaining the equipment it provides to its client. Clients usually pay on a per-use or utility computing basis.

Containers-as-a-Service (CaaS) is an emerging cloud services offering for container-based virtualization in which providers offer a complete framework to customers for deploying and managing container clusters, and applications. In a CaaS model, containers are provided as a service that can be deployed in on-premises data centers or over the cloud. At the heart of a CaaS is a container orchestration platform, which is designed to handle operations such as container deployment and cluster management. Popular examples of container orchestration platforms are Google Kubernetes, Docker Swarm, OpenStacks nova-docker etc. Finally, report the findings and provide recommendations on which service is well suited for given set of application requirements.

B Tools Appendix

B.1 IOzone

To install IOzone on CentOS 7 Minimal on the Dell P390, T310 and T610 machines, follow the steps provided below[23]

- Download IOzone tar file

```
sudo yum install -y wget
wget http://www.iozone.org/src/current/iozone3_394.tar
```

- Extract IOzone tar file

```
tar xvf iozone3_394.tar
```

- Change directory

```
cd iozone3_394/src/current
```

- Build executable

```
make
make linux
```

List of common commands in IOZone[24]

- Naming your test (output)

```
iozone -b <File Name>
```

- Run all tested file operations for record sizes of 4k to 16M for file sizes of 64k to 512M

```
iozone -a
```

- To a particular test:

```
iozone -i <Test Number>
Test numbers:
    0 = write/rewrite
    1 = read/re-read
    2 = random-read/write
    3 = Read-backwards
    4 = Re-write-record
    5 = stride-read
    6 = fwrite/re-fwrite
    7 = fread/Re-fread,
    8 = random mix
    9 = pwrite/Re-pwrite
    10 = pread/Re-pread
    11 = pwritev/Re-pwritev
    12 = preadv/Re-preadv
```

- Create temporary files of size from a given file size (64k to 512M)

```
iozone -s <File Size>
```

- Include CPU Utilization

```
iozone -+u
```

- Increase file size (important if your system has more than 512MB of RAM)

```
iozone -g <File size>
```

B.2 Bonnie++

To install Bonnie++ on CentOS 7 Minimal on the Dell P390, T310 and T610 machines, run the command below:

```
sudo yum install -y epel-release
sudo yum install -y bonnie++
```

List of common commands[25]:

- Naming your test (output)

```
bonnie++ -m <Name of Test>
```

- Set directory to use for the test

```
bonnie++ -d <Directory Name>
```

- Run as a particular user

```
bonnie++ -u <Username>
```

- Testing IO performance measures with default file size of 8192 megabytes

```
bonnie++ -s
```

- Testing IO performance measures with given file size in megabytes

```
bonnie++ -s <File Size>
```

- Testing file creation (in multiples of 1024) with default file size of 0 bytes

```
bonnie++ -n <Number of Files>
```

- Testing file creation (in multiples of 1024) with given file size in bytes

```
bonnie++ -n <Number of Files>:<Maximum Size>:<Minimum Size>:<Number of Directo
```

- No write buffering after every write

```
bonnie++ -b
```

- Repeat Test multiple times

```
bonnie++ -x <Number of Test Runs>
```

B.3 Stress

To install Stress on CentOS 7 Minimal on the Dell P390, T310 and T610 machines, run the command below:

```
sudo yum install -y epel-release
sudo yum install -y stress
```

List of common commands[26]:

- Stressing the CPU

```
stress -c <Number of Cores>
```

- Stressing the Memory (RAM) with default process size of 256 MB

```
stress -m <Number of Processes>
```

- Stressing the Memory (RAM) with given process size in MB

```
stress -m <Number of Processes> --vm-bytes <Process Size>
```

- Stressing the Disk Drive with default 1GB file size

```
stress -d <Number of processes>
```

- Stressing the Disk Drive with a given file size

```
stress -d <Number of processes> --hdd-bytes <File Size>
```

- Setting up a timeout with time frame in seconds

```
stress -t <Time Frame>
```

Note: If -t option is not specified, user needs to manually exit from Stress

B.4 Stress-ng

To install Stress-ng on CentOS 7 Minimal on the Dell P390, T310 and T610 machines, run the command below:

```
sudo yum install -y epel-release
sudo yum install -y stress-ng
```

List of common commands[27]:

- Stressing the CPU

  ```
  stress-ng --cpu <Number of Cores>
  ```

- Stressing VM with default process size of 256 MB

  ```
  stress-ng --vm <Number of Processes>
  ```

- Stressing VM with given process size in Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g

  ```
  stress-ng --vm <Number of Processes> <Process Size>
  ```

- Stressing Disk Drive by continuously commiting buffer cache to disk with default 1GB file size

  ```
  stress-ng --io <Number of Processes>
  ```

- Stressing Disk Drive by continuously commiting buffer cache to disk with a given file size given in Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g

  ```
  stress-ng --io <Number of processes> --hdd-bytes <File Size>
  ```

- Setting up a timeout with time frame in seconds, minutes, hours, days or years with the suffix s, m, h, d or y

  ```
  stress-ng --timeout <Time Frame>
  ```

Note: If -t option is not specified, user needs to manually exit from Stress-ng