A Report submitted in partial fulfilment of
the regulations governing the award of

the Degree of

BSc (Honours) Computer Networks and Cyber Security

at the University of Northumbria at Newcastle

Project Report

Comparing Virtual Machine and Container Performance for Headless Servers in Computer Networks

James Poxon

2020/2021

General Computing Project

Declaration

I declare the following:

1. that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to ALL sources be they printed, electronic or personal.
2. the Word Count of this Dissertation is ⟨len⟩
   (result of shell command `texcount -total -inc Dissertation.tex`)
3. that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.
4. I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

   In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.
5. I have read the Northumbria University/Engineering and Environment Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED: . . . . . .

Acknowledgements

I would like to thank my dissertation supervisor Alun Moon for his role in support of my project. I would also like to thank my partner Alex for supplying me endless encouragement throughout the dissertation process.

Abstract

Virtualisation is a standard and de-facto technology within server operations, but virtualisation has it's pitfalls, with one of those being the large overheads introduced from virtualisation of a whole operating system. A possible solution to this problem is containerisation. Containerisation runs without a virtualised operating system, instead integrating directly with the host machine's operating system. This effectively removes the OS-component processing overhead required to run each instance of a server, which not only makes servers use less system resources, but theoretically could improve the speed of said servers.

An in-depth analysis into Virtual Machines and Containers was completed to ensure understanding of the problem domain before delving into comparison work of container and virtual machines. It became apparent that most previously published comparison work was specifically aimed at raw hardware performance and utilisation, which whilst useful, doesn't give results relevant to real-world performance.

Two separate but topologically identical computer networks were built. Said topology implemented a primary DNS, two secondary DNS, DHCP, Web, and MySQL servers. One network used VMware (Virtualisation), and the other used Docker (Containerisation) to host the servers. The network topology was designed to be reflective of a possible real-world internal network that we may expect to see in an SME (Small and Medium-sized Enterprise). Four separate benchmarks were then designed, using a mix of different techniques and software, such as JMeter and Sysbench, to accurately test every part of the network.

Impressive and substantial performance improvements found when moving from VMware to Docker. In some cases, Docker produced

over double the output that VMware did. Docker was also found to be far more stable than VMware.

Organisations looking to get extra performance out of ageing hardware could potentially use containers as an alternative to virtual machines for their server infrastructure. However, it is noted Docker is not intended to be used in the way we are applying it. As a result, it was decided that to better support container implementation of servers in the future and to support the transition from Virtualisation to Containerisation, a better, container-based solution for server management should be developed. The benefits of container-based servers are clear to see as a result of this research, but a more server-focused platform that can match the equivalent server-focused virtual machine based platforms that already exist is needed.

Contents

Chapter 1

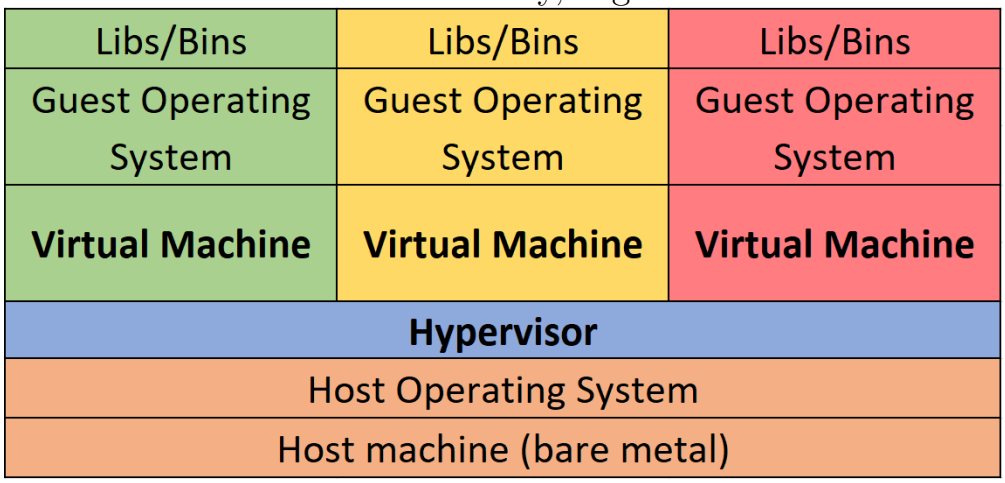
Introduction

1.1 Summary of the Current Situation

Within computer networking, and computing as a whole, we tend to move through paradigms. Generally, there is a set way of doing things and eventually there are breakthroughs that shift that standard. One such paradigm is the use and reliance of Virtual Machines in the hosting of servers. As written by Ameen and Hamo [2013], "Virtualization has rapidly become a go-to technology for increasing efficiency in the data center.". Ameen and Hamo were right when they wrote this in 2013 and more and more server infrastructure has moved to virtualisation since. If we look at uptake of virtualised networking, servers and storage more recently, we see can see that virtualisation has become one of the de-facto solutions for server management. Spiceworks [2020] found in "The 2020 State of Virtualization Technology" report that 92% of the companies that they surveyed already used server virtualisation, with a further 5% planning on adopting it within two years of that date [Spiceworks, 2020]. Some of the main reasons for this extremely high adoption are: savings in power and hardware (One physical machine can support multiple servers) and Logical Resource Consolidation [Wolf, 2007]. These savings however, don't come at no cost. One of the main stresses on any virtualised server infrastructure is the high resource usage that comes with hosting more than one server-based service on one physical machine. As server tasks become more strenuous, or when entirely new services need to be added to an already existing infrastructure, we can find that hardware (such as memory and

CPU usage) becomes overwhelmed [Wolf, 2007]. As the tech world moves forward, the infrastructure that supports that tech within organisations needs to do so too. This is why we are starting to see a large number of organisations moving to cloud-based infrastructure, or taking the leap to invest in upgraded server hardware, that can withstand the large workloads required for some modern-day server loads. These solutions however, come with their own problems and negate the key reasons that these organisations moved to virtualisation in the first place. Cloud-based solutions, whilst being able to take the onus away from an organisations infrastructure team, can also result in costly hosting fees, not to mention the fact that cloud services often can't be used in place of some internal network components, such as DHCP and internal DNS servers. The other solution; upgrading hardware, again misses the mark and undermines the reasoning for why virtualisation was implemented in the first place. As already discussed, virtualisation is preferred because it can save money on costly server infrastructure. Instead of having to purchase (and power) several physical machines, you have one powerful machine that can support all the same services. When we reach the ceiling in terms of output from that sole machine, an upgrade seems inviable, but at this point, how much money was actually saved? Once you factor in the extra overhead that running a virtual machine introduces as a result of Virtual Machine and OS virtualisation resource usage (Figure 2.1), we start to see that Virtualisation might not be the perfect solution it once was.

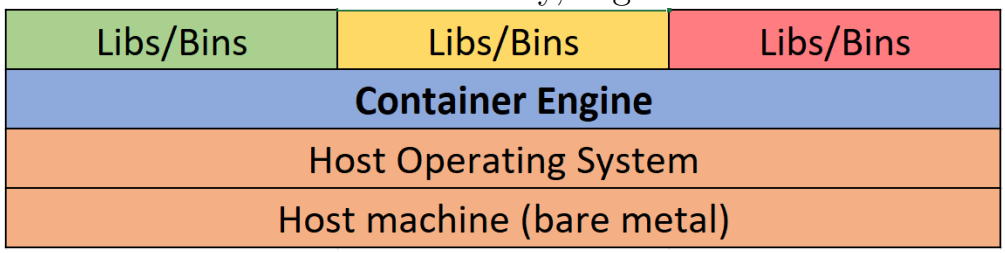
For reference only, Figure 2.1



1.2 A New Contender: Containers

One possible solution to this problem, is instead of upgrading hardware, organisations simply remove the Virtual Machine and virtualised OS components, along with their resource utilisation, whilst keeping the core functionality of the servers that they host intact, as shown in Figure 3.1. This is where containerisation comes into play. Containerisation, removes the need to virtualise a whole Operating System for each virtual machine, instead ingratiating with the host machine's Operating System directly, whilst keeping each container separated logically. These containers can still interface with real networks in much the same way that Virtual Machines do, but also reduce the overheads and the hardware resource utilisation dramatically. This allows SME's and other organisations currently using Virtualisation to squeeze more performance out of their already existing hardware without having to do costly upgrades to hardware.

For reference only, Figure 3.1



There has been a number of reports and published papers recently that demonstrate the improvements that containers offer over Virtual Machines for different use cases such as Dua et al. [2014] who found that Containers had a significant advantage over Virtual Machines in Platform as a Service environments. Another example of such findings is Joy's "Performance comparison Linux Containers and Virtual Machines" (2015). This research found, as expected, that containers were much faster at processing requests than virtual machines. Where this research falls down however, is the range of data. Testing was done across a few areas, but is not all-encompassing of any specific area. Rather, it is a first look into the benefits of containers, without a real and solid recommendation on who should move to containers, and exactly what the benefits would be to any real-world systems.

There are countless more studies that could be mentioned here in regards to the 'containerisation vs virtualisation' discussion, but the key finding around these studies tends to be that containers are better in doing a set task, but rarely is that task designed in a way that would test the real merit of containerisation within a real-world system. As a result, no one study appears to give a direct recommendation to anyone about moving to containers, and as a result, it seems we are figuratively 'stuck' in this paradigm of virtualisation.

The research in this project report aims to contribute to this possible paradigm shift by making the case for containerised server solutions. Containers may not be the replacement for large-scale cloud solutions that some want it to be, but in cases where servers must still remain internal like with DHCP and internal DNS (as discussed in section 1.1), containerisation offers a way to gain massive performance improvements along with decreased hardware utilisation.

1.3 Aims, Objectives, and How They Were Met

The key aim of this report was to compare Virtualisation and Containerisation in a test environment that better represents a real-world server system.

In order to do this, we set and then met a number of objectives. Firstly, we set out an analysis part, where we have discussed the current problem domain surrounding virtualisation, and why we need to find an alternative. Within this part we also explained the key differences between containerisation and virtualisation as this is very important to provide context to the research that is conducted in later parts.

As discussed in section 1.2 above, there is a lack of comparisons in this field that actually test real-world environments. As a result, the next step for this report was to carefully select both the hardware and software to be used in testing, and design an infrastructure topology that accurately reflects what could be a real-world internal server environment. Two separate systems were built, both using the exact same topology and configuration, but with one using a Container-based solution, and the other a Virtual-Machine solution.

Then a total of four tests (with five data sources) were conducted across the entire topology. These tests were created in order to accurately evaluate the performance of both as a real-world server topology. This means the tests accurately measure parts of the topology that are actually important in those environments; instead of measuring just raw computing performance, we measured performance on the network, and the ability for each technology to perform common server-environment tasks, such as MySQL database OLTP [Bog, 2013] and web server request latency. The Synthesis process was also conducted in a way that would allow us to pull information about the process of setting these systems up using our chosen tools. This information is important for our recommendation, as ease of use could be more important than performance for some end-users.

These tests then provided us with a large quantity of data that allowed us to compare the parallel performance of each server in either system. A detailed evaluation then followed, and with it we found that containers still hold up far better than virtual machines in a real server environment. As was planned, more detailed recommendations than seen from other similar studies were created. Through the method of creating the systems, we found that whilst creating both systems was relatively easy, there were some caveats in the process of using containers that hadn't previously been considered. This is exactly why matching research systems as closely as possible to real-world systems is so important. These findings are worked into the recommendations sufficiently.

There were of-course some planned limitations within the study that need to be addressed. Whilst we did look at areas other than just performance in our recommendation, the key and quantitative data was only performance oriented. There are other studies, such as Watada et al.'s "Emerging Trends, Techniques and Open Issues of Containerization: A Review" (2019), which highlight security and isolation issues around containers, that are equally as important to the conversation around containers (Discussed further in section 12.1.1). This was entirely out of the scope of our report though. The work that can be done in a one-person report has to be limited, and whilst further delving into an all-encompassing review of containerisation in server environments would have been something we would liked

to have done, the scope of the report had to remain reasonable.

Overall though, the project was success. The results speak for themselves, and hopefully can be useful for future research and development of containers and server infrastructure solutions.

Part I

Analysis

Chapter 2

Review of Virtualisation

2.1 Terminology & Definitions

2.1.1 Virtualisation

Virtualisation as a term originated in the 1960s when IBM workers began work on a project that would allow an IBM model-40 computer to segregate off its memory and allow up to 15 users to use the computer independently at once [Lindquist et al., 1966]. Each user would see their own abstract Operating System, separate from the others. Whilst virtualisation has continued in its development from this point onwards, the core functionality and architecture behind Hardware Virtualisation, remain much the same.

Each of these individual logical (as opposed to physical) devices is called a 'virtual machine'.

2.1.2 Hardware Virtualisation

There are a number of different types of virtualisation. This can open the scope for what can and can not be considered virtualisation, so we must be firm on our definition here. For the purpose of this report the term 'virtualisation' will refer specifically to 'hardware based virtualisation', whereby a 'virtual machine' is an operating system running on top of another operating system, with the virtualisation tasks being controlled by a 'hypervisor' (This is explained in more detail in the next subsection: 2.1.4).

This is an important definition to clarify, as sometimes containerisa-

tion (Chapter 3) can be viewed as a type of virtualisation (due to the similarities of what they provide). For the purposes of this report, the two definitions must be distinguished as separate things, much like in the report "Autonomic Orchestration of Containers: Problem Definition and Research Challenges" [Casalicchio, 2016], where a clear and defined difference between Hardware Virtualisation and Containerisation is made. In the following sections we shall provide more in-depth description of what a virtualisation is, and what containerisation is, so as to reduce any possible confusion.

2.1.3 Host Machine

Though this applies to containerisation also, it is important to define the host-machine here. The host-machine is the bare-metal physical machine that runs instances (whether that be virtual machines or containers). The host-machine Operating System (OS) is the OS that is installed at the lowest level on the hardware.

2.1.4 Hypervisor

Hardware Virtualisation relies on an underlying software that runs on the host-machine's Operating System in order to manage each instance/OS. This software can have a number of different names depending on the origin of the work, and the context. In early work on virtualisation, this software was often referred to simply as a "control program" [Creasy, 1981], but for the purposes of this research, this software will be referred to as a 'Hypervisor'. This term is often preferred in practical settings, such as in VMware's online Glossary [VMware, n.d.], or in Red Hat's "What is a hypervisor" [Hat, 2021], and has become the de-facto term for this type of software environment.

The hypervisor acts as a "manager" for virtual machines, performing tasks like logically splitting up hardware on the host machine so that virtual machines can make use of it, such as RAM, CPU cores, Disk space, network management or more [Fragni et al., 2010]. It is also usually responsible for managing user interface into the operating system, whether that be through a graphical input, or other means.

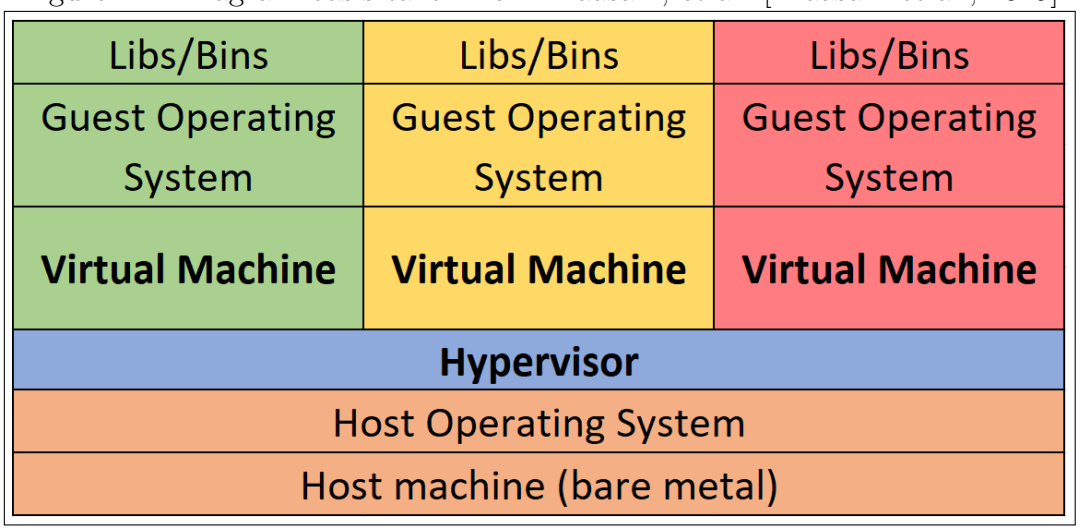
Some hypervisors also offer different abilities. One example is network interface and management that allow multiple virtual machines to communicate with each other [VMware, 2021d].

2.2 The workings of virtualisation

Virtualisation can work in a number of ways, and as mentioned by Thiruvathukal, et al. there could be whole reports looking into the intricate details of exactly what virtualisation is [Thiruvathukal et al., 2010] and how it works on different machines, or operating systems. However, most virtual machines (and the one's we will be using later in our research) follow some version of what will now be described. Firstly, the host operating system (installed on the host machine) provides interface to the hardware of the machine. Then, a hypervisor (as discussed above in subsection 2.1.4) bridges the gap between the host machine's operating system, and the the operating systems that are going to be virtualised [Fragni et al., 2010].

Figure 2.1 shows a diagram to better illustrate the layers that go into creating a virtual machine. This diagram shows a hypervisor that is supporting three virtual machines. The layers start from the bottom and move up the diagram. The 'Libs/Bins' part of the diagram is the binaries [The Linux Documentation Project, 2021a] and libraries [The Linux Documentation Project, 2021b] that are required to allow certain applications to run on operating systems.

Figure 2.1: Diagram basis taken from Alaasam, et al. [Alaasam et al., 2019]



2.3 Choosing a Virtualisation Platform

There are a number of different hypervisors available for the practical work we will be doing in this project. Each one has different benefits and compromises. With this in mind, we must go through a process of deciding which technology we want to use to support our infrastructure. It is important to pick a virtualisation platform that is used in the industry I am aiming my research towards (that being SMEs with a need to host servers internally).

2.3.1 VirtualBox

VirtualBox is a popular virtualisation platform provided by Oracle [Oracle, 2021b]. It includes virtual network cards that allow virtual machines to communicate with each other [Oracle, 2021a, 1.3. Features Overview], which is beneficial to us when hosting interconnected servers. It also offers various networking options that allow for internal networks, as well as forwarding data onto a real interface [Oracle, 2021a, 6.2. Introduction to Networking Modes]. This means servers could be connected in a virtual network, or could forward their traffic to a hardware network interface card, which means we can do our testing, and also apply the findings to real-world networks.

However, VirtualBox has recently come under scrutiny for a reduction in stability of their virtual machines, with some users reporting crashes, and slow-downs in operation [Bradford, 2020]. When wanting to find a good contender for a comparative piece of research, it is important to stay away from something that could result in unfair results. As using VirtualBox may lead to slow-downs that are not a result of the Virtualisation technology itself, it wouldn't be a fair comparison to compare VirtualBox Virtualisation with a Containerisation, as differences in results could actually stem from something other than purely the raw performance we would expect to see on a virtual machine. VirtualBox is also considered a desktop-virtualisation package primarily, and doesn't offer any server-specific virtualisation packages, which also seems unfair given the great number of other virtualisation options available that *do* have server-oriented versions.

2.3.2 Hyper-V

Hyper-V is a hypervisor product developed by Microsoft. Unlike VirtualBox with its lack of server-oriented packages, Hyper-V has two primary versions (Though there are sub-versions within these), one is a server-specific version designed to work on windows server, and the other is a stand-alone version designed to work on windows desktop environments (Like windows 10) [Microsoft, 2018].

Hyper-V also supports three different virtual switch methods, those being "Private" (traffic can flow between VMs only), "Internal" (traffic can flow between VMs and the host, alternatively NAT can also be configured to allow access to the internet), and "External" (traffic is bound to a physical NIC and forwarded onto a real-world network) [Lee, 2017]. For our testing, we would be able to use the internal switch with NAT so that our VMs could access the internet. Results could then be applied to real world networks using the external switch.

Like VirtualBox it supports multiple Operating Systems, including Linux (which we will need for our testing, see Section 5.2). However, wider Operating System support isn't one of Hyper-Vs strong points [Collins, 2019], and this support for 'Linux' is actually only actually full support for specific Linux operating systems, such as Ubuntu, Debian or CentOS [Microsoft, 2016]. Whilst this wouldn't necessarily be a problem for *our* tests, it does somewhat damage our ability to use it as a recommended system, as some end users and organisations might not be able to make use of it due to requiring specialist operating systems that are simply not supported within Hyper-V.

2.3.3 VMware

VMware is a well-known virtualisation package and hypervisor. Whilst Hyper-V had a relatively small guest OS support; VMware has a relatively huge support base for various different Operating Systems [Collins, 2019] [VMware, 2021b]. This is important because it ensures that if relevant organisations want to use this research, they have a greater chance of being able to move over *other* server functions that aren't necessarily covered in this report.

VMware provides hypervisor technology for a number of different applications, including server infrastructure virtualisation and desktop virtualisation. The core hypervisor technology across these different versions however, remains much the same [VMware, 2021g].

The other key issue picked up with other virtualisation packages (namely VirtualBox) was stability. VMware on the other hand, is considered very stable [Pavlik et al., 2012], and is certainly more stable than VirtualBox [Bradford, 2020]. This ensures that our results are reflective of virtual machine performance, and not shifted by hitches in the hypervisor programme.

2.3.4 Decision

With the above in mind, in order to provide virtual machines for testing in this research we will be using VMware Workstation Pro [VMware, 2021h]. This choice comes for a number of reasons, one being because it is one of the leading virtual machine hypervisors for the desktop environment [G2, 2021], which makes application to real-world server infrastructure much easier. Another main reason we decided to use VMware is because of its many versions despite using the same technology across said versions. As a result, data we get from this research in relation to VMware Workstation Pro, should be applicable to other VMware applications, such as VMware ESXi which is used in enterprise server applications [VMware, 2021i]. This crossover between server variants and the desktop variant of VMware gives us a unique advantage as we can easily create and manage virtual machines in our desktop testing environment, whilst not damaging the credibility of our results.

VMware also provides us the exact network options we require, whilst also giving us the ability to alter and change networks using the built-in network manager [VMware, 2021d] (this is important for section 7.4 later in this report).

Chapter 3

Review of Containerisation

3.1 Terminology & Definitions

3.1.1 Containerisation

Containerisation has become the de-facto term to describe what has also been described as OS-level virtualisation [Hogg, 2014]. For the purposes of this report, I will be referring to this technology only as Containerisation, and each individual instance shall be referred to as a container (instead of virtual machine). This is the same approach towards defining containers as taken by Dua et al [Dua et al., 2014] who have made their own distinction between Containers and Virtual Machines in much the same way that we have.

The next sections will cover what containerisation is, and how it works. This is important so that the difference between virtualisation and containerisation is fully realised.

3.1.2 Container Engine

What a hypervisor is to a virtual machine, a container engine is to a container. A container engine sits on the base operating system, much the same as a hypervisor. Where a difference is found however, is in the way it interacts with said base operating system.

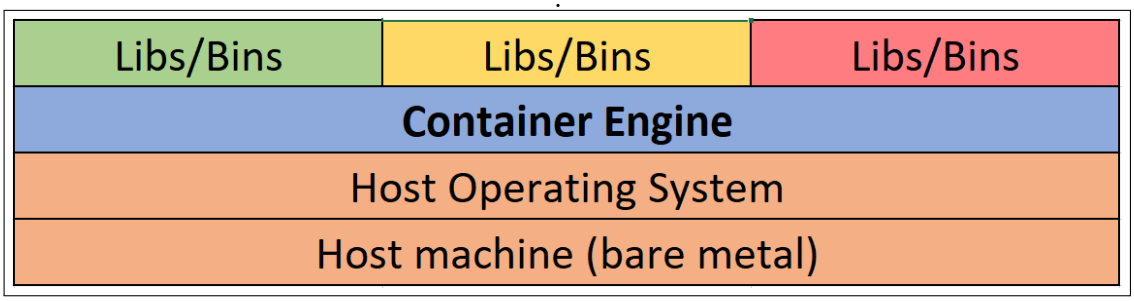
Whilst a hypervisor works by running a full operating system on top of the existing one, and then managing the transfer and allocation of resources from there, a container engine works without that upper operating system by using the host machine's operating system as

the OS component for each container. Segregation of containers and resource allocation is still managed by the container engine, so that each container is allocated separate and isolated resources. This can be done dynamically or be static, depending on the use-case (much the same as virtualisation).

This effectively removes part of the overhead that was previously required for each virtual machine. In theory, this should allow for improved performance over Virtual Machines, especially when system resources are scarce or under a large load.

Figure 3.1 shows diagram that illustrates how the containers and container engine work in relation to the host machine. When compared with the diagram in figure 2.1, we can see that containers have far less overheads in comparison to virtual machines.

Figure 3.1: Diagram basis taken from Jenny Fong (Docker Blog)[Fong, 2018]



3.2 Choosing a Containerisation Platform

Whilst containerisation is still relatively new, there are still a few options for us when deciding which containerisation package to use. Similarly to how we decided which virtualisation package we are using, we need to do a little bit of analysis into the various alternatives available to us, and decide which one is best for our own uses.

3.2.1 FreeBSD Jails

FreeBSD is an operating system that descended from the Unix family, much the same way that Linux did. BSD relates to "Berkley Software Distribution" which was a (now defunct) Operating System. When BSD stopped being supported in 1995, a number of organ-

isations continued developing their own versions of BSD. By 2005, FreeBSD was the most popular by far [BSD Certification Group, 2005], and remains one of the most popular versions today.

FreeBSD has been used to produce systems for a number of different use cases, but of those, server infrastructure is the one that would make most sense to our research. FreeBSD can support a number of different server functions, such as DNS, Web and FTP servers [FreeBSD, Section 1.2.1.].

Whilst not named "containers" by the FreeBSD team, what FreeBSD calls Jails *are* a form of OS-level virtualization (which for our research, we are considering to be containerisation, as discussed in subsection 3.1.1). Jails segregate functions logically, but still use the base kernel and Operating System in order to function. A unique benefit to FreeBSD jails is benefit that a long history of support provides. FreeBSD Jails were first introduced in 2000 [FreeBSD, 2000], and have received continuos updates and development cycles, making it one of the most sturdy container systems to date.

However, to make use of FreeBSD Jails, it is a hard-requirement that the Operating System and Kernel *must* be FreeBSD. There is no way of supporting FreeBSD Jails on anything other that FreeBSD as it is baked into the FreeBSD environment itself. This takes away from the autonomy of businesses that may want to use the product, as they can't choose to use a different operating system (such as windows or Linux). This could also be a problem as an organisation might see the outcome of this research and decide they want to move to a container-based system, only to be stopped in their tracks due to being unable to integrate the FreeBSD Operating System in their current system.

3.2.2 LXC

Another option is LXC, also known simply as 'Linux Containers', it was released in 2008. LXC makes use of a number of features of the Linux Kernel such as namespaces (explained further in subsection 3.3.1), Control Groups (also explained in subsection 3.3.1). These are Linux specific features that allow individual containers to access the same base Linux kernel, but logically run as separate Linux machines

[Linux Containers, 2021a].

Linux Containers also have a project known as LXD, which can be thought of as an extension to LXC [Jain, 2016]. LXD is developed primarily by Canonical [Linux Containers, 2021b], who also develop the Linux Ubuntu Operating System. This gives LXC and LXD a big advantage as it has support from some industry leading organisations.

Similarly to FreeBSD Jails, LXC can support various network modes (when LXD is used). Of those, we have 'macvlan' which gives each container a MAC address, and allows it to communicate using a hardware NIC, this makes it suitable for server environments [Linux Containers, 2021c].

The technology and backing behind LXC and LXD seems like it should be a perfectly good contender for our container roll, however, it still suffers from the same problem as FreeBSD, that being that is requires a set Operating System and Kernel (Linux) in order to work. When looking at Virtual Machines, we found a Hypervisor (VMware) that could support multiple Operating Systems. It seems then, that our container program should at the very least be able to make use of more than one Operating System too. For that reason, LXC still isn't the container solution for us.

3.2.3 Docker

An increasingly more popular implementation of container technology is a software known as Docker. This package is primarily used by (and aimed at) developers, who can use the platform to quickly create and deploy applications in container environments for testing. The use of containers is supposed to make managing these applications much easier, as it ensures compatibility along the whole development process. For example; a developer can code a program, then pass their code *and* container onto QA who can test the code in exactly the same environment [Docker, 2021d].

Somewhat importantly, is that when Docker first released in 2013, it used LXC as its foundation [Osnat, 2020]. Docker has since moved on from this, and now offers far more features than LXC, such as

a shared library of container images, similar to GitHub repositories named 'Docker Hub' (Explained in more detail in subsection 3.3.1) and version control similar to Git Versioning [UpGuard Team, 2021]. Docker Hub contains numerous images that are maintained by organisations that are in charge of specific technologies. For example; there is an official Docker Image in the Docker Hub for Ubuntu [Canonical and tianon, 2021]. It's no surprise then, that Datanyze (a technology market usage group) reported that Docker is currently the second most used Containerisation platform, with 25.34% market share in Containerisation [Datanyze, n.d.].

Possibly due to its LXC heritage, Docker offers networking options that are very similar, even using the name 'macvlan' [Docker, 2021c], which works perfectly for our current needs.

Where Docker strays from the other Container engines mentioned, is it's multi-OS support. Docker not only supports Linux Operating Systems, but also now supports Windows and Mac Operating Systems [Docker, 2021b]. This is great news for those that want to implement Docker on systems they may already have implemented, without need an Operating System change. Furthermore, Docker also provides versions that are suitable for implementation on AWS (Amazon Web Services) and Microsoft Azure [Docker, 2021b], which are both cloud services. This makes docker perfect for our comparison, as these cloud-capable versions allow Docker to go head to head with VMware's 'VMware Cloud' [VMware, 2021a], and other Virtual Machine solutions which work across various platforms.

For the above mentioned reasons, we will be working will use Docker as the container component in our testing for the rest of this report.

3.3 Using Docker

3.3.1 How Docker works

Docker's Container engine is aptly named Docker Engine, and utilises a client-server architecture [Docker, 2021, Section: Docker architecture]. The server portion of the Docker System is known as the 'docker daemon', the Client uses a command line interface to interact with one or more docker daemons. The client and dae-

mon communicate using a 'REST API' [Docker, 2021, Section: The Docker daemon]. REST (Representational State Transfer) provides an architecture for web services [Booth et al., 2004] that allows them to communicate over HTTP. The protocols within REST are stateless, this means there is no set 'state' or session control within the protocol; each command sent to a daemon can be understood as it is, without the need for any outside context to the command being sent.

The Docker Daemon manages 'Objects' [Docker, 2021, Section: Docker objects] required for a full Docker system. 'Objects' refers to the containers, the images (Docker images are the instruction sets for Docker containers, not to be confused with OS images, though often the Docker images will include which OS image is to be used).

Docker images are stored in a Docker Registry [Docker, 2021, Section: Docker registries]. By default, the registry is configured to use "Docker Hub" which is a public, open registry that already contains a large number of complete images for use. This registry can be changed however, to point to any location. This behaviour allows a registry to be setup part of a network, and the images can then be 'pulled' or 'run' from the registry using the "docker pull" and "docker run" commands [Docker, 2021, Section: Docker registries]. An image can also be configured on a live container, and then that image pushed to the registry using the "docker push" command [Docker, 2021, Section: Docker registries].

Docker is written using the 'Go Programming language' [Docker, 2021, Section: The underlying technology]. Go is developed and maintained by Google on an open source license, and is roughly based on the C programming language [The Go Authors, 2021a]. Being based on C gives the programming language the same benefits of any other low level language, being able to make use of functions that are integral to the kernel and operating system. Where go differs is that it is also designed to be be more intuitive and "clutter free" [The Go Authors, 2021b].

Namespaces

Docker makes use of a feature of Go that allows further use of a Linux kernel feature known as namespaces [Docker, 2021, Section: Namespaces]. When new containers are created, a set of namespaces are created specifically for that container. This means that programs that might otherwise be considered by the kernel to be entirely separate, are processed together, and vice versa. This in turn allows multiple containers to run processes in isolation that otherwise would have been processed by the kernel together, and also process a number of actions as one whole unit, that otherwise would have been considered separate tasks to the operating system. This is key to the function of containers, as it allows each container to process tasks as separate entities, but make use of the same kernel and operating system across all containers.

Control Groups

Control groups is another feature of the Linux kernel used by Docker. Control groups allow hardware resources to be segregated in a way that limits and further isolates them [Corbet, 2007]. In docker, this is used to segregate parts of memory, logical processors, drive space and network access so that there is no crossover in hardware utilisation between different containers [Docker, 2021a]. This is similar to how a hypervisor would segregate resources, but instead this is managed entirely by the kernel.

Chapter 4

Virtual Machine and Container Comparisons

4.1 Research comparing both methods

4.1.1 To support PaaS

Research has already been conducted comparing containers and virtual machines, such as Dua, et al.'s study into using containerisation and virtualisation in PaaS (Platform as a service)[Dua et al., 2014]. In this study, it was concluded that containers performed better in almost every way for online applications, but they also concluded that adoption of containers was still relatively low, and that there were some issues with standardisation in the field [Dua et al., 2014].

That study was conducted in 2014 however, and since that study took place, we have had a whole host of changes that could have had substantial impact in the field. For example; since this study, Microsoft has released Windows 10 (2015) [Myerson, 2015], and then later, Windows Subsystem for Linux (WSL), which provides the ability to run Linux applications on Windows [Loewen, 2019]. WSL2 has also been released, using a real Linux kernel [Loewen, 2020], and has been adopted by container applications such as Docker [Docker, 2021c], allowing them to run better on windows machines, could this be the start of the standardisation that Dua, et al. were looking for?

4.1.2 A more recent review

In 2019, Watada, et al. did a review of containerisation and found that containers were faster in virtually every deployment when compared to tradition virtual machines [Watada et al., 2019]. This study was thorough in looking at the workings of various Virtual Machine and Container technologies, but most of these were tested on cloud applications, such as AWS (Amazon Web Services). The results in this study focused more on seeing how containers were already deployed in working environments [Watada et al., 2019, VII.], and the benefits of using them in those situations. Whilst recommendations were made as to where containers should and shouldn't be used, the study makes mention that complex networking is difficult to achieve using containers [Watada et al., 2019, VIII. A.].

The main issues broached by Watanda, et al. regarding complex networking with containers is IP address application, making note that most most containers make use of NAT networking [Watada et al., 2019, VIII. A.]. It is stated, however, that NAT networking can be avoided by assigning individual containers directly to host network interfaces [Watada et al., 2019, VIII. A.].

4.2 Full Virtualised/Containerised Networking

4.2.1 Virtual networks

We have already discussed the many applications of virtual machines, and among these is the ability to run multiple networked VMs on the same hardware, keeping various parts of the network logically separate. One example of this is VMware's network manager, whereby virtual machines can be made to appear as separate entities on a physical network [VMware, 2021d], whilst all sitting on the same base machine.

4.2.2 Containerised Networks

The studies and research mentioned above in section show that containers, in most situations, are faster than Virtual Machines. Where containers are said to fall down is their ability to interface and be

deployed easily in real world situations. This is why most SME's still rely on Virtualisation for a large portion of their network infrastructure, despite the fact they could probably prolong their hardware, or get more raw power from existing hardware by moving to containers.

In this report we will be aiming to create a typical network topology that could easily be deployed on virtual machine infrastructure, and then, we will be applying this same exact topology to a containerised network, and then measure that actual performance benefits in order to generate a recommendation to those that are still using virtual machines to host their infrastructure.

Chapter 5

System design and definition

5.1 Maintaining scientific method

To ensure that results are scientific, variables must be controlled between both of the systems. The first step in this, is ensuring that the topology and configurations for both systems are the same. This can be done by copying the configuration files from one system to the other, ensuring that the system works in the same way for both systems. As the underlying operating system should be a version of Linux for both the VMware and the Docker system, this should be relatively easy.

To further ensure that variables are controlled, we need to ensure that the same benchmarks are maintained throughout the testing process, when being used on the *same part* of the system. This means that if one benchmark is used to measure, for example, network latency on a web-server, then the same benchmark should be used to measure the network latency on that same web-server on the mirrored system. It may be necessary to use different benchmarks across the whole system, but this is acceptable as long as all testing is done to a parallel across both systems. The benchmarks to be used will be discussed in more detail in section 5.3 (Benchmarking).

5.2 LAMP System

I will need to make sure that the test system is as accurate to a real-world system as possible. To do this I will be employing a LAMP

stack topology for both the Docker system and the Virtual Machine system. LAMP is the acronym for Linux, Apache, MySQL and PHP, and is a common way of organising server infrastructure [IBM, 2019]. Whilst this can technically be run all on one system, it is common to separate various functions out onto different machines (logically or physically), this generally makes management of these systems easier.

For the Linux section of the LAMP topology, I will be employing Ubuntu Server as it is headless (resulting in a lower overhead), and because Ubuntu is one of the most widely used Linux operating systems for website infrastructure as of 2021 [Q-Success, 2021b]. Ubuntu even has its own images stored officially on Docker hub, which is updated regularly to stay in line with Ubuntu's Long Term Service version [Canonical, 2020]. Some of the images hosted on Docker Hub by Ubuntu [Ubuntu and Docker, 2021] are already configured to contain some of the parts required for the deployment, such as Apache2 and MySQL. These may be useful when implementing a Docker setup in the synthesis of this report, though for the sake of ensuring a fair-test, I may instead push images to docker that use the exact setup used by my virtual-machine testing. This could remove a possible source of extraneous variables from the testing.

5.3 Benchmarking

Benchmarking software and tools are designed to create a standard output measurement for performance of a computer-based system [Fleming and Wallace, 1986].

There are a number of benchmarking tools available, but for this research these can be split in discussion along lines of what they are designed to measure. That being said, the main split for this research is the measuring of computing performance, and of network performance.

5.3.1 Computing Performance

Computing performance in this case relates to the performance as a result of the computers ability to process information, and at what

rate. Whilst this is tied directly to the processor, RAM, and other hardware, the overhead of the Operating system can affect these components and as a result, have a large affect on the performance of a system. This effect is known as 'operating system overhead'. Based on previous research on containers and virtual machines it can be hypothesised that in this research, the total Operating System overhead for a container-based system will be smaller than that of Virtual Machines *(when using the same operating system)*. This is due to the way that containers utilise one OS for their function (as discussed in subsection 3.1.2).

5.3.2 Network Performance

Further to the Computing Performance; Networking Performance is the measured performance on the network between various nodes. These nodes are the servers, clients and other infrastructure that are configured to receive and send traffic on a network. One of the best ways to measure network performance is delay. Compute performance may change the network delay in some areas, but not all. To explore this, we should further break network delay into its four main components:

- Processing Delay: The amount of time it takes for a node (router, server, client, etc.) to process the header of a packet [Ramaswamy et al., 2004]. This is usually affected by the CPU performance, which is linked to the compute performance as discussed previously.
- Queuing Delay: The time spent after being processed or produced by the node, and then actually being pushed onto the line [Institute for telecommunication sciences, 1996]. This is called queuing delay, because due to other data already being pushed out onto the line. As a result, our data is waiting in a 'queue' behind other data, waiting for its turn to be pushed onto the line. This is affected by the amount of traffic being sent from a node, along with the speed at which the node can process multiple packets. This does tie in with Computing performance in a similar fashion to Processing Delay.
- Transmission Delay: The amount of time it takes for a node to

'push' packets (bit by bit) onto the line [Chen, 2005, Chapter 7]. This delay is a result of the bandwidth on a line, and as a result, the change in transmission delay between virtualisation and containerisation can't be hypothesised. This is due to differences in the way that both methods manage network traffic. There is no physical line between the servers, so differences in bandwidth are entirely down to the relevant solution (Containerisation, or Virtualisation).

- Propagation Delay: The time it takes for packets to travel over a line/medium (such as copper cable) [Messer, 2021]. This could be affected by the the change from virtualisation to containerisation but this is again down to the individual ways that each of the solutions I choose manage network traffic between logical machines. As such, I can't generate a hypothesis for how this would be affected, much the same as Transmission Delay.

5.3.3 Other key findings

Whilst the purpose of this project will be primarily to find differences in speed between the two methods, there may be other performance, or even logistical improvements of one over the other. The final output from this research should be a recommendation to those that might be considering using containers in order to run their infrastructure, so whilst performance might be the key focus of this report, I will ensure to mention in evaluation anything else that could be considered important to those that might be looking to containerisation as a 'step forward' for infrastructure. For example, in subsection 3.3.1 (Docker) I discuss the Docker Hub, which is an public repository of images that anyone can push to or pull from. This system might make the deployment of containers easier than the deployment of virtual machines, so whilst this isn't directly related to the performance of the system itself, it may be worth mentioning as a possible point of interest.

We can see from exploring these various types of delay on the network, that network delay can be directly tied to Computing Performance in *some* areas. As a result of this, it could be expected that the changes to overhead, and the possibility of streamlining processes as

discussed in chapters 2 and 3 result in better network performance.

5.4 Requirements List: Infrastructure

To create a realistic server infrastructure, a number of different services will need to be implemented. The following section lists the node requirements for our two server networks.

5.4.1 Webserver

The network will contain an intranet website in order to best simulate the intranets that are often operated by enterprises of various sizes across many different areas. Intranets are an integral part of a large number of businesses, and aim to serve as a central node for a large number of operations within businesses. These intranets can often host a number of important services for day-to-day working of a business [Jacoby et al., 2005], such as time-sheet and clock-in access, information sharing, and policy documentation hosting, to name a few.

Apache

The Apache HTTP server (Apache) is an open source HTTP server software developed by the apache foundation. It is (as of current) the most widely used HTTP server in the world with 34.5% of known websites using Apache to host their infrastructure [Q-Success, 2021a]. This makes it the perfect tool for simulating a real-world enterprise network, and should give fairly realistic results from benchmarks.

5.4.2 DNS

The architecture will require a DNS system in order to resolve hostnames outside of our own control (Recursive DNS), and to to maintain the control over the domains for the intranet discussed in subsection 5.4.1. Two secondary DNS servers will be configured to interact with clients, whilst the primary (authoritative) server will be hidden from clients by having no hostname. Having secondary servers acts both as a way to load balance DNS requests from the clients, and to act as a redundancy should one of the servers fail.

It is important to mention that for the purposes of testing, Reverse DNS (rDNS) will also be configured [Mockapetris, 1987]. This service essentially converts from IP addresses to hostnames, and is useful for using test commands such as Ping and Traceroute. These commands wont appear in the testing, but will be useful for troubleshooting when the systems are being created.

The primary and secondary forwarding addresses for queries unknown to the primary DNS server will be Google's Public DNS servers (8.8.8.8 and 8.8.4.4) [Google, 2021].

BIND

As the machines running the network infrastructure will be using Linux, the best fit for DNS architecture is BIND. This is because BIND is the de-facto standard for DNS on linux, and because it allows us to run the DNS server both as a recursive (for finding DNS queries for the outside network) and an authoritative (for managing the intranet domain) DNS server. BIND is maintained by the ISC [Goldlust, 2021], who also maintain a Docker Image for the application in docker hub.

5.4.3 DHCP

The network will also require a DHCP server in order to allocate IP addresses to clients, as well as inform said clients of the IP addresses for the two secondary DNS servers, and the gateway IP.

ISC-DHCP

Another ISC solution [Goldlust, 2021], this will be used in order to manage DHCP. This service is simply known as ISC-DHCP and is another de-facto service in enterprise environments. These programs are in line with the testing I am doing, as they are commonly used in real world environments, which is where my research aims to replicate and implicate.

5.4.4 MySQL

As discussed in 5.2, a MySQL server will be used in order to host a database. There are a number of reasons an organisation might want to host a database on their internal network, such as storing employee information for access on the intranet. Databases can take a lot of resources to maintain, and are often an important part of key infrastructure. Any increase in database performance from virtualisation to containerisation would be a clear success, and a good reason to move from Virtual Machines over to Containers. MySQL has its own Ubuntu Server Package, making the server portion of the installation easy.

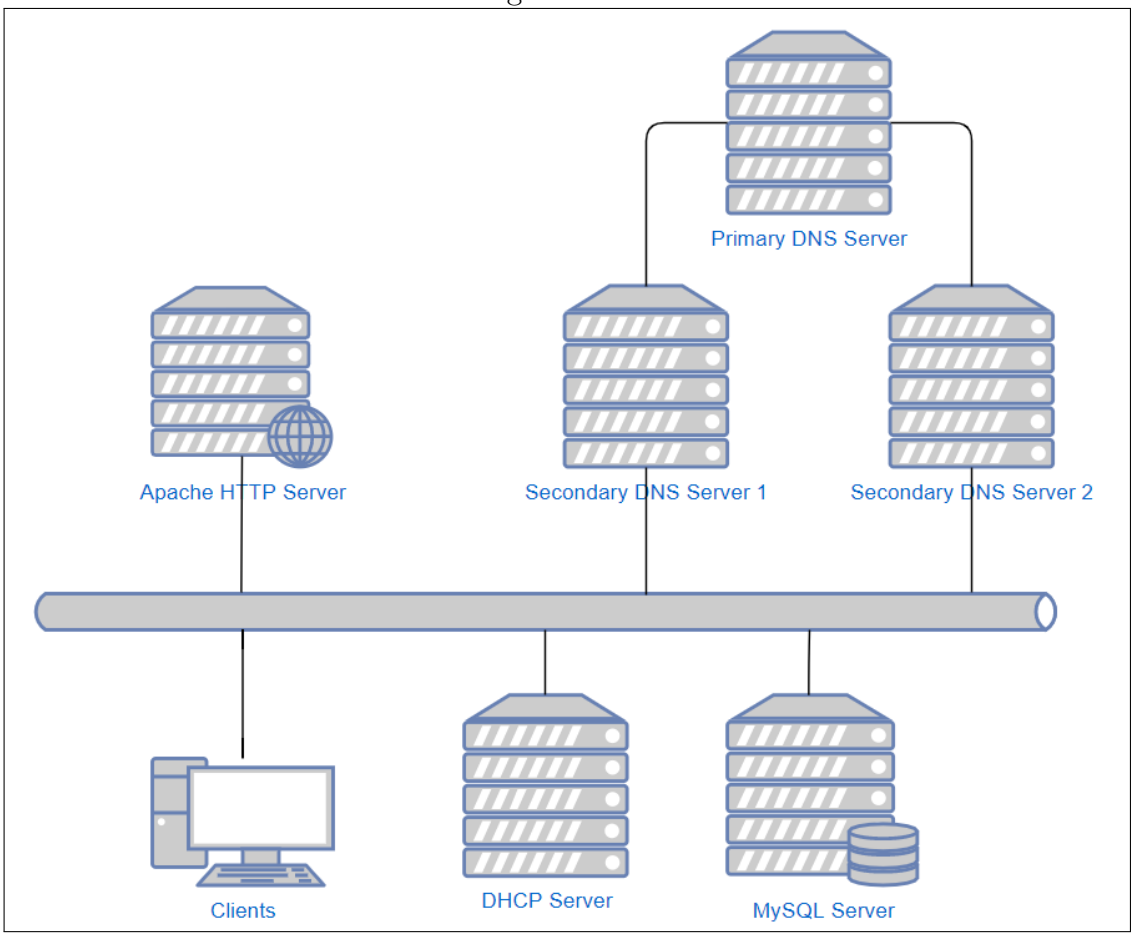
5.4.5 Clients

Whilst not technically a service, the network will have to include at least one client in order to test that the services listed above actually work as intended. These sorts of internal networks are made entirely to serve clients, so without testing performance on the client end, the testing would be somewhat redundant. Ubuntu Desktop will be used, as it is also open source and can be deployed easily on both systems. The client machine used in testing will be a VMware machine when testing *both* systems. This way, the same system resources will be allocated to the client in both testing environments. Whilst there will be an impact on the performance of the machine during the testing due to the client being on the same host machine, it should have the same impact on the host machine's system resources across both systems, and shouldn't change the legitimacy of the output.

5.4.6 Topology Diagram

Figure 5.1 is a mock-up of what the final topology should look like logically. Here we can see the dual secondary server DNS architecture, and how each machine sits on the network. The bar in the middle logically could represent a virtual switch.

Figure 5.1:


Primary DNS Server
Apache HTTP Server
Secondary DNS Server 1
Secondary DNS Server 2
Clients
DHCP Server
MySQL Server

Chapter 6

How will the system be measured

6.1 Requirements list: Benchmarks to be used

The system will need to be measured using standardised benchmarks in order to ensure that results are comparable.

To ensure that all relevant elements of the system are measured, I will do four tests as follows.

6.1.1 Test one - iPerf3

iPerf is a tool used for measuring the total bandwidth achievable over IP networks [Dugan et al., 2021a]. In this test, the throughput will be tested between the client and the Apache server, as realistically, this is the server that would receive the most traffic in a real deployment. It can be expected that any server to client bandwidth would be the same however, given that the connections are all virtual and should work much the same way, regardless of which server is being used.

Measuring throughput for the web server is important because a higher throughput should mean a greater number of users being able to use the website without long wait times for example.

6.1.2 Test two - Sysbench MySQL

Sysbench is a benchmarking tool that comes with a number of different tests for measuring processing, memory usage, Input/Output speed, and database performance[Akopytov, 2020]. Whilst the tool is extremely versatile, with tests for a number of different functions,

the test that is of interest to this work is the 'oltp_read_write' test supplied. This test performs a number of random read and write queries and transactions to a pre-prepared MySQL database using OLTP (Online Transaction Processing), which is a system which facilitates small-scale, multi-user transactions and processes to a database[Bog, 2013].

This test will be performed on the MySQL server outlined in subsection 5.4.4, with the client machine being used to send the queries and initiate reads and writes.

The output from this test should give the number of queries that were successfully performed, along with the total latency for each transaction that takes place. These measurements are important because lower latency and a higher number of transactions in the same amount of time in a real deployment would result in better load times for end users for any web pages that require database manipulation. Even databases that don't face any users could see benefit from increased performance however, as lower latency allows for greater workloads [Eyada et al., 2020], which would allow heavier, more demanding work to be allocated to the same system.

6.1.3 Test three - Namebench

Namebench is an old and archived, but still fairly standard DNS benchmarking utility developed by Thomas Stromberg as part of Google's 20% program [Stromberg, 2010]. The project is using the Apache Open-Source License, and is still available to download and compile for use on Linux.

The tool's original purpose was for testing DNS response times for multiple websites against well-known, public DNS servers. The output from the test was designed to give the user a good idea as to which servers to use as their primary and secondary DNS servers in order to achieve the best response times possible when browsing the web. However, the outputs given can also be extremely useful for testing internal DNS server speeds. For example; the average, maximum and minimum latency given for each server. When this test is done for both deployments (Docker and VMware) the outputs can be compared to determine which DNS architecture gives less latency.

DNS latency directly affects how quickly websites load, as time is needed to resolve the server address from the IP. Overstressed DNS architecture can result in larger query latency[Liang et al., 2013] so we should see some interesting results from stress-testing this part of the infrastructure.

6.1.4 Test four - Apache JMeter and Performance Measurements using Netdata

Apache JMeter is an open-source application provided by the Apache Software Foundation designed to allow load-testing of various servers [Apache Software Foundation, 2021]. It was initially designed for testing of web-applications, and whilst it can now be used to test a number of different protocols, we will be using it for testing of HTTP/HTTPS.

JMeter is fully user customisable, but under a fairly standard and basic use, it allows for the simulation of a number of clients to a remote server in order to simulate what a different loads would look like if the server were to be in a real deployment[Rodrigues et al., 2019].

This test will be performed against the Apache server laid out in subsection 5.4.1, and as with the other tests, the client machine will be used to initiate the test, so that the traffic uses the network environment.

However, JMeter only gives performance results that are perceived by the clients that it simulates, for example, query latency. It doesn't show the impact on performance of the machine itself during these stress tests. So to measure the computing performance (as laid out in subsection 5.3.1), Netdata will be implemented to run on the host machine in order to monitor the performance metrics.

Netdata is open source software that is designed to pull numerous performance metrics such as CPU and RAM usage in real-time from nodes and end-points, whilst maintaining as small of a footprint on that node as possible [Netdata, 2021]. These statistics are displayed in a visual 'dashboard' which display a number of charts showing usage. A key feature of this dashboard is the ability to export and

import snapshots [netdata, 2021]. This feature will allow a snapshot to be created for each scenario, allowing for historical access to the exact performance metrics from each test. This allows for in-depth analysis even after the testing has finished.

Part II

Synthesis

Chapter 7

System creation and setup

7.1 The host machine

7.1.1 Hardware

To do the testing, an older desktop machine with limited resources will be used rather than a modern server machine. This should better represent one of the target groups of this research; that being SMEs that may have aging hardware, attempting to get the most out of what they currently have available.

The specification of the machine is as follows:

- Processor: Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz
- RAM: Corsair Vengeance 2x8GB DDR3 @ 1600Mhz (CAS latency of 10 clocks)
- Motherboard: Gigabyte H81M-S2PV (Dual Channel Memory)
- Graphics Card: NVIDIA GeForce GTX 1060 (3GB VRAM)

This processor has four cores, and is single threaded; it was released in 2013 and support from Intel discontinued in 2017 [Intel, 2017]. Nevertheless, it is still a fairly decent CPU and is reflective of what we might see is some legacy servers running in some real-world systems.

This motherboard supports up to a maximum of 16GB of RAM, and a frequency of 1600Mhz in dual channel [GIGA-BYTE Technology Co, 2013] which we are taking full advantage of with two 8GB sticks of 1600Mhz RAM. The CAS latency is important here

too. CAS means "Column Address Strobe", and relates to the number of times the memory clock cycles between the 'read' command and the data actually being read [Jacob, 2004]. Generally the lower the CAS latency, the better for real-time speed. The intricacies of memory profiles and CAS latencies is far beyond the scope of this report, but it is important to mention in the event anyone decides to re-create or re-test this study, and wants to be able to have a comparison of hardware within their own research in the future. As with any benchmarking test, small changes in hardware can generate big consequences towards our results, so being as transparent as possible with regards to the hardware specification is within our best interest. CAS latency *can* make a noticeable difference in real-world speed of RAM, so even if two RAM sticks are of the same frequency, having different CAS latencies *could* shift results, though that is not to say that they absolutely would do.

The speed of the graphics card should make little difference to the results, but is worth mentioning given the graphical output will be driven entirely by this card for the whole of the testing, thus removing any strain that would be put onto the Chipset from using onboard graphics.

7.1.2 Operating System

The Host Machine will be using a fresh installation of Ubuntu LTS 20.4 Desktop [Zemczak and Canonical, 2021]. Using a fresh installation is important to ensure that results are as close to stock as is possible.

Ubuntu is being used because the infrastructure that we will be building uses solely Linux-based solutions (BIND9, ISC-DHCP, Apache, etc). Though this can be supported for containers on Windows, it can only be done using Windows Subsystem for Linux 2 (WSL 2). The decision was made to avoid WSL 2 as whilst WSL 2 does load a real Linux kernel [Microsoft, 2020] (making it have less overhead than a traditional Virtual Machine), it still runs that kernel through a windows environment, so there's no guarantee that this wouldn't be detriment to the performance of the containers. Using Ubuntu as the base operating system ensures the containers we run

interface directly with our base operating system as explained in the Analysis, see Figure 3.1.

When considering VMWare, having the host machine's operating system as ubuntu is acceptable, as VMware Workstation supports Linux as a host operating system [VMware, 2021f].

7.2 Servers

7.2.1 Operating system

Both systems were created from scratch, using the latest LTS versions of Ubuntu Server (20.04, Focal Fossa Server [Ubuntu Documentation Team, 2021]).

For VMware, the images were downloaded directly from the Ubuntu Website and the virtual machines were created using VMware's workstation Virtual Machine creation wizard, whereas for Docker, the Ubuntu Server image was taken from Ubuntu's official Docker Image, hosted on Docker Hub [Canonical and tianon, 2021]. Each docker image was then created using a Dockerfile, which lays out which version to use; in this case, ubuntu:latest.

7.2.2 Software

The servers were created to use the software and topology laid out in the requirements within section 5.4.

For both VMware and Docker this meant using the Advanced Package Tool (which is the default package manager on ubuntu) to install the various software needed.

For VMware this was done once each server was installed and booted. However, for Docker, through the use of Dockerfiles, this can integrated directly into the Docker image, along with any configuration files that may be required, such as local zone files for the primary DNS server (DNS1). An example of a Dockerfile for DNS1 is shown in figure 7.1 below. This shows how the base image is selected, along with the addition of the software and tools necessary to make bind9 work. The 'COPY' command takes the completed configuration files and places them in the folders required for BIND to work.

Figure 7.1: Dockerfile

```
FROM ubuntu:latest

RUN apt-get update \
  && apt-get install -y \
  bind9 \
  bind9utils \
  bind9-doc


COPY named.conf.options /etc/bind/
COPY named.conf.local /etc/bind/
COPY db.intranet.co.uk /etc/bind/
COPY db.72.168.192.in-addr.arpa /etc/bind/

CMD ["/bin/bash", "-c", "while :; do sleep 10; done"]
```

Both methods of creating the machines and grabbing software are straight forward, though I would argue that Docker, whilst maybe having a slight learning curve, is faster once the user knows what they are doing. Writing Docker files for each machine was quick and easy.

The one caveat with using Dockerfiles, however, was that during setup of phpMyAdmin on the MySQL machine, the installation process asks for user input. The Dockerfile creation process isn't intelligent enough to return these to the user so the creation of the Docker image fails in this instance. Instead, the MySQL Server Dockerfile was created without phpMyAdmin and then once the container was created and running we could install phpMyAdmin manually. Once this is done the container in its running state must be 'commited' to a new image using the '`docker commit`' command [Docker, 2021a] (much like in a Git Commit [GitHub, 2021]), otherwise the changes wont be saved for the next time the container is launched. Instead, without a commit, it would launch fresh as if phpMyAdmin was never installed.

7.3 Client

7.3.1 Operating System

The client machine was created using VMware Workstation Pro, using Ubuntu's latest LTS desktop version (20.04, Focal Fossa Desktop [Zemczak and Canonical, 2021])

The same client machine was used for both the VMware system and the Docker System. This was to ensure that the client used the same amount of resources (RAM, CPU and Network usage) on the host machine for both the VMware tests and the Docker tests, as in a real environment, the clients would be remote devices on the network, not on the same node as the server. By using the same virtual machine as the client in both tests, any impact to the outputs of the tests should be mitigated.

The Client machine was configured to use up to 4GB of RAM, and up to two cores using VMware Workstation Pro.

7.3.2 Software

The client machine had all the testing and benchmarking software required installed before any testing took place, so that the machine was the same between testing.

The software installed was as described in the requirements list in section 6.1. As mentioned in that section, Netdata is installed on the host machine during the final test, not the client machine. This is to ensure that usage statistics represent the whole machine, not just the client, or the endpoint (in this case, the web server).

7.4 The Network

For both tests, VMware Workstation's NAT networking mode was used [VMware, 2021c]. The same network was used for both in order to ensure a fair test. Using the same network provided by VMware's NAT setting ensures that any differences in network performance measured in the tests is due to factors other than the network infrastructure itself; more specifically, the difference between Docker and

VMware's core performance in a network setting.

The configuration of this network was edited using VMware's "Virtual Network Settings" [VMware, 2021d] to disable the built in DHCP server, so that it didn't conflict with the DHCP server created for the testing (see subsection 5.4.3).

Setting up VMware Virtual Machines to use the NAT (VMnet8) network is as straight-forward as selecting this network for the virtual machine before booting. On the contrary, to do the Docker test more work was required. Firstly, a Custom Docker Network named "CustomNet" was set up using Docker's "macvlan" network driver [Docker, 2021c]. This network driver is designed to be bound to a physical network, similar to the "bridged" mode found in VMware by giving each Container an individual MAC address. When creating a "macvlan" network, the interface to be used must be specified. This is where VMware's Virtual Network Adapter was used [VMware, 2021e], which is designed to allow the host machine to communicate with virtual machines over a virtualised IP interface that is installed on the host machine. This allowed the Docker Network "CustomNet" to be bound to the VMnet8 interface, thus allowing the Docker containers to communicate with each other over the VMware network. This also made it possible for the VMware client to be used in the Docker test as explained in subsection 7.3.1.

Table 7.1 shows the IP addressing and hostname scheme that was used in both tests.

Table 7.1:

	IP Address	**DNS Hostname**
DNS1	192.168.72.3/24	No hostname
DNS2	192.168.72.4/24	ns1.intranet.co.uk
DNS3	192.168.72.5/24	ns2.intranet.co.uk
DHCP	192.168.72.6/24	dhcp.intranet.co.uk
Apache	192.168.72.150	www.intranet.co.uk
MySQL	192.168.72.151	mysql.intranet.co.uk
Client Addresses	192.168.72.50/24 - 192.168.72.100/24	No hostname

For testing, when a server needed to be specified, the hostname was used over the IP address where applicable so that the process would make use of the DNS infrastructure. For example,

'www.intranet.co.uk' is used in the Jmeter test, instead of '192.168.72.150'.

7.4.1 Difficulties with Docker and DHCP

As mentioned above, Docker was using the 'macvlan' Docker network driver. This network driver is not the recommended driver for use by Docker, as generally Containers in Docker are designed to run within the constraints of the machine they are running on, and not to face out onto a physical network. Due to this, there was an occasion during the creation of the Docker system whereby the DHCP server was not giving our IP addresses even though it was configured to do so.

After trying various troubleshooting methods, and after rebooting several times, it seemed there was no solution to the problem. It was decided that we would come back to the issue the following day with a fresh mind. After booting up Docker fresh the following day, the DHCP server started working as intended. I believe the issue was due to the way the Docker handles replies over IP, as it seemed to be dropping important packets headed into the DHCP container that were part of the DHCP handshake process (The process would restart from the beginning again, and then the returning packets would drop in a continuous loop). This is however, merely hypothesis, as if I had the real solution to the problem I was facing I would certainly explain it in more detail, but this erroneous behaviour was never observed again.

Even after the Docker containers were saved to .tar files, and then reloaded onto different hardware, the DHCP server ran entirely as expected. I would suggest this speaks to the fact that this work is on the edge of what Docker is really designed to do, so your own tests and observations on your own hardware, may in-fact differ, or you may experience similar problems.

Chapter 8

Testing & Benchmarks

Introductory explanations for each of these tests and benchmarks are included in the requirements in section 6.1.

8.1 Test 1 - iPerf3 Throughput

8.1.1 Test parameters

In this test, the throughput is measured between the Apache server and the Client. The listener for iPerf3 was setup on the Apache server using port 5201 (the default port for iPerf3 [Dugan et al., 2021b]). The client then handles the initiation process with the server, and tells the server which format to record throughput with. For this testing, the format used was 'M', meaning Megabytes (as it is specified by the iPerf3 Documentation [Dugan et al., 2021b]).

The test runs for ten seconds, taking measurements of total throughput in every second, resulting in ten points that can be converted into a graph showing the transfer rate (throughput) over time in Megabytes per second. An average can also be taken from this. A separate throughput is calculated on each side of the transaction (at the server, and at the client).

8.1.2 Results

The average throughputs for both tests, and for clients and severs, are shown in table 8.1 below.

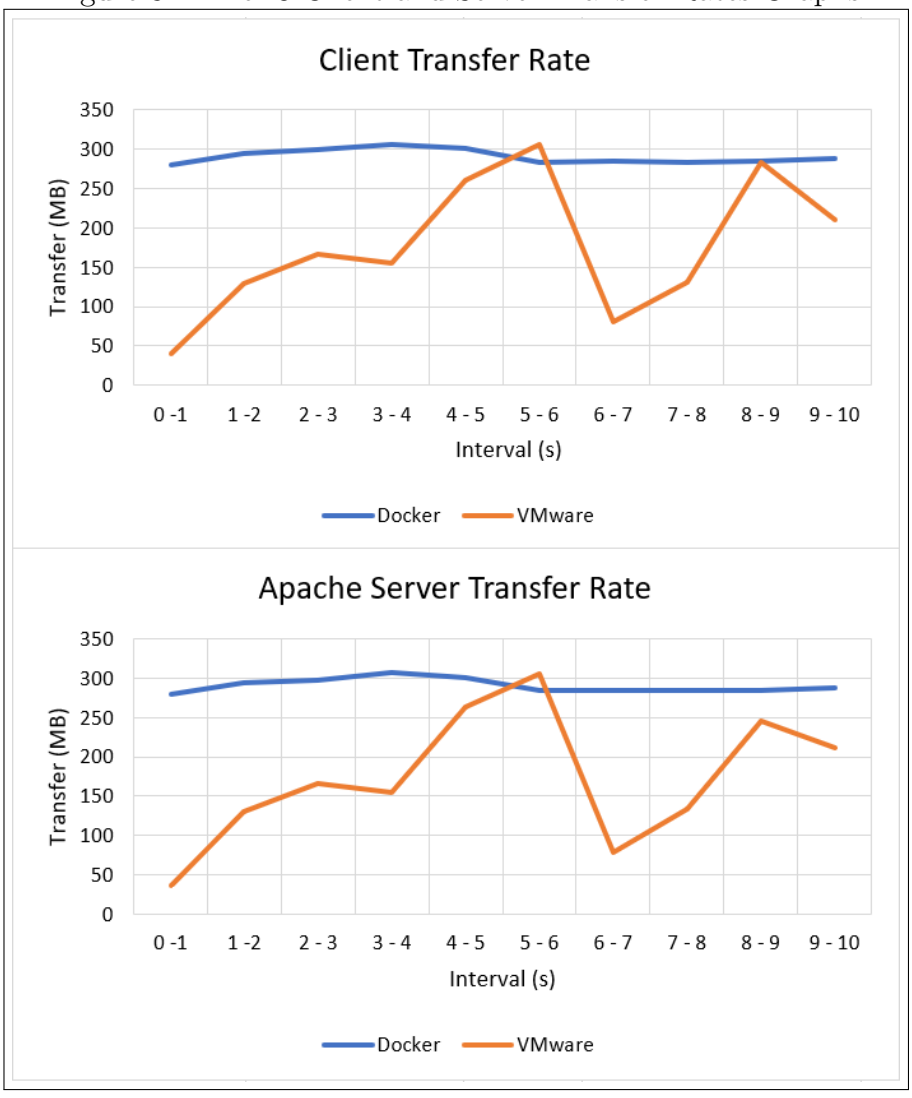
Table 8.1: Table showing the average transfer rate in a 10 second period

x	VMware Average (MBytes/s)	Docker Average (MBytes/s)
Client	176.37	290.70
Apache Server	172.61	290.40

The averages shown in this table clearly show a higher throughput for the Docker system, with an improvement (averaged between the Client and the Apache Server) of 116.06 MegaBytes per Second.

Figure 8.1 shows the transfer rate over time for both Client and Server.

Figure 8.1: iPerf3 Client and Server Transfer Rates Graphs



This shows that Docker is stable, whilst VMware's throughput result fluctuates far more. This would suggest that the VMware system is

unstable; something we will investigate further in the evaluation part of this report.

8.2 Test 2 - Sysbench MySQL Input/Output

8.2.1 Test parameters

This test required a test database with random data to be created on the MySQL server. Luckily, Sysbench includes a 'prepare' tool, which allows a user to specify parameters for the table that match the test they want to conduct [Akopytov, 2020]. Sysbench then creates, or *prepares*, a database that is suitable for said test. A test user account with all permissions was also created for Sysbench to use. When the prepare command is run, it creates the database and tables, and then fills the tables with the amount and type of data specified.

The database we created for testing had the following parameters:

- The database had two tables.
- Each table had a size of 500,000.
- Table size in sysbench relates to rows. Meaning across the two tables, there was a total of one million rows of data.
- The test was specified as '`oltp_read_write`'.

OLTP is Online Transaction Processing[Bog, 2013], which is a term used to describe online databases that generally have high throughput and connectivity from many users [Cyran, 1999]. The `oltp_read_write` test attempts to simulate an OLTP workload by performing various different SELECT, DELETE, INSERT and UPDATE queries on the data [Ksiazek, 2018].

Once the above table was created, the actual test could be performed. The client machine runs the command that starts the test, and the domain name for the MySQL server (mysql.intranet.co.uk) was used to specify where the server and database were located.

The test was run using the following parameters:

- The port was set as 3306 (the default for MySQL).

- Two threads were used. This simulates the database being accessed and changed by two entities at the same time. This doesn't mean two different users however, as in a real system, just one of these 'entities' could be in charge of processing multiple user queries.
- 'Time' was set as 300. This is in seconds, meaning the scenario ran for five minutes.
- 'Events' was set as 0. This setting would allow the user to specify that the test should stop after a certain number of events have been reached. By setting this to '0', the test will continue to run until the 'Time' setting is reached.

8.2.2 Results

Figure 8.2 shows the total number of Read, Write and Other queries performed by Sysbench in a 5 minute period. The results for both systems are compared side by side.

This graph shows a clear win for Docker, with the VMware system managing to perform just over a third of the number of Reads that Docker managed in the same time frame.

Figure 8.2: MySQL Queries Performed in Five Minutes Graph

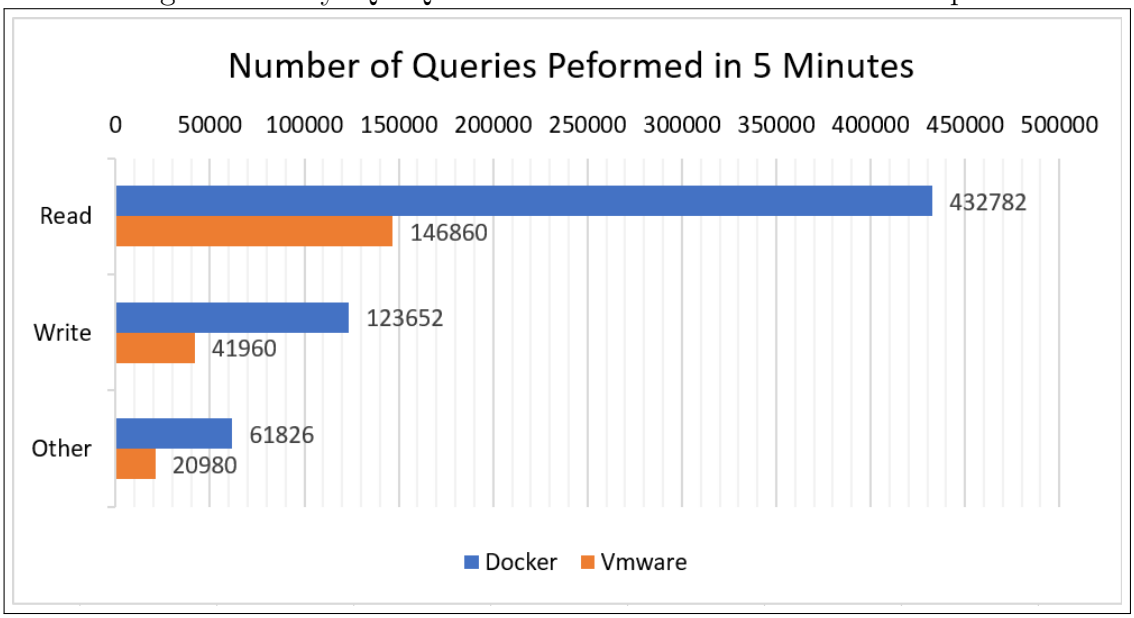
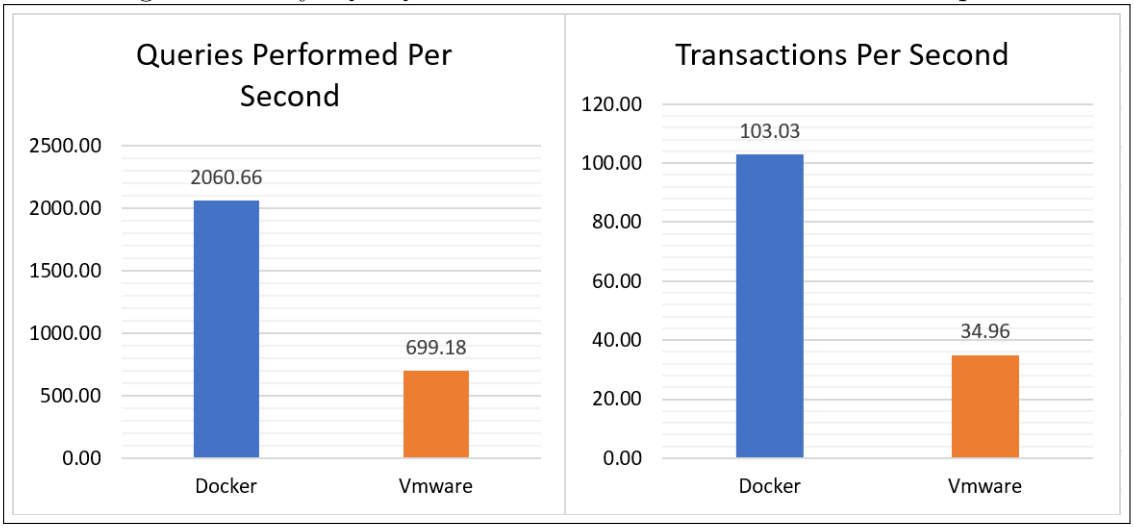


Figure 8.3 shows both the total queries per second, and the total

transactions per second for both systems. A query is defined as being a single statement such as SELECT, INSERT or UPDATE [Oracle, 2021c]; whereas a transaction is a combination of multiple of these statements within one command [Oracle, 2021d], typically in order to manipulate multiple rows of data.

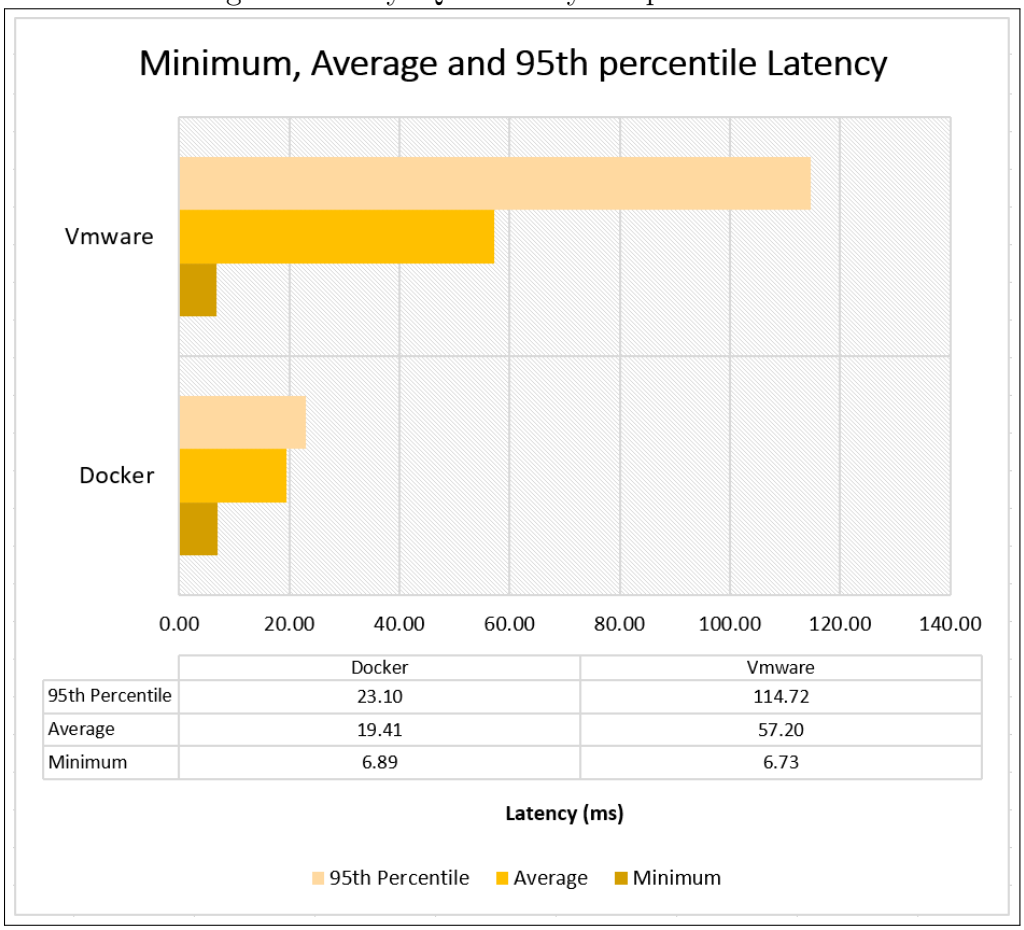
This further shows just how much faster the Docker system was at data manipulation.

Figure 8.3: MySQL Queries and Transactions Per Second Graphs



The latency between queries was also measured by Sysbench. Figure 8.4 shows a latency bar graph, with the minimum, average, and 95th percentile latencies in Milliseconds. The 95th percentile was used rather than the maximum, because both tests experienced a maximum latency that would have dwarfed the other results. These maximum latencies were also most likely errors (which were to be expected with such a high number of queries being performed [Bog, 2013]).

Figure 8.4: MySQL Latency Graph and Table



	Docker	Vmware
95th Percentile	23.10	114.72
Average	19.41	57.20
Minimum	6.89	6.73

We can see from these results, that Docker far out performs VMware in terms of latency, to such an extreme extent that Dockers' top 95th percentile results are lower than VMware's average latency.

8.3 Test 3 - Namebench

8.3.1 Test parameters

This test compares the performance of the DNS servers and their ability to perform DNS queries for various websites.

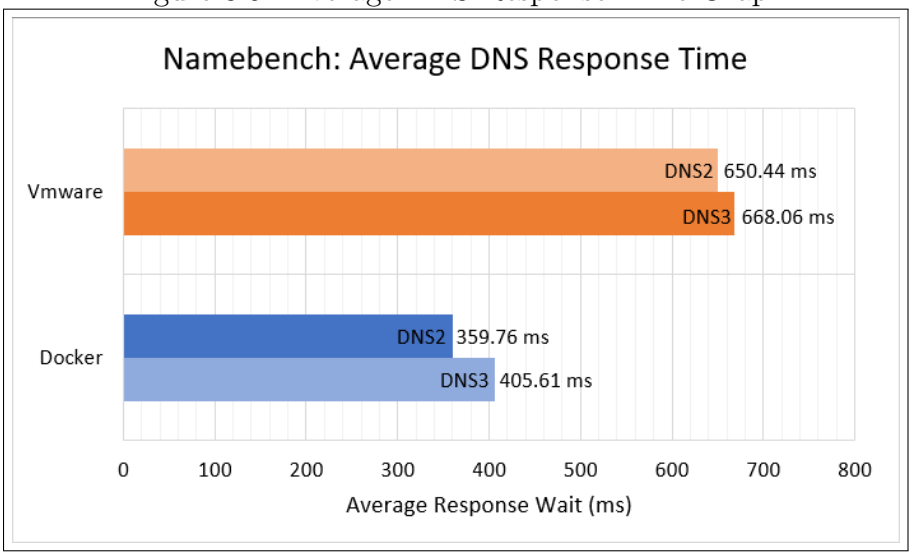
Namebench works by going through a list (specified by the user) of websites and recording the latency for the DNS server to perform the query [Stromberg, 2010]. Before the test commenced, the caches on all of the DNS servers in our infrastructure were wiped to ensure a fair test between the two systems. In our testing, the following parameters were set:

- The list to be used was specified as 'alexa'. This option uses Amazon's Alexa Top Sites list [Alexa Internet, 2021]. Around 38,000 different website addresses were tested. Having a high number of addresses to test against is important because it gives a good average.
- Both DNS2 and DNS3 were specified (using their domain names: ns1.intranet.co.uk and ns2.intranet.co.uk respectively) to be tested. DNS1 is a primary server, and is hidden in the network topology, only to be used as the master for the 'intranet.co.uk' domain, so it isn't tested here.

8.3.2 Results

Figure 8.5 shows a graph of the average DNS response times. We can see from this graph that Docker has smaller average DNS response when compared to VMware. An interesting, but unrelated behaviour here is that DNS2 in both tests out performed DNS3; this is unexpected given that both servers share exactly the same configuration. This is outside the scope of this report, but perhaps this is a characteristic of the way that BIND9 handles secondary DNS servers? Interesting nonetheless!

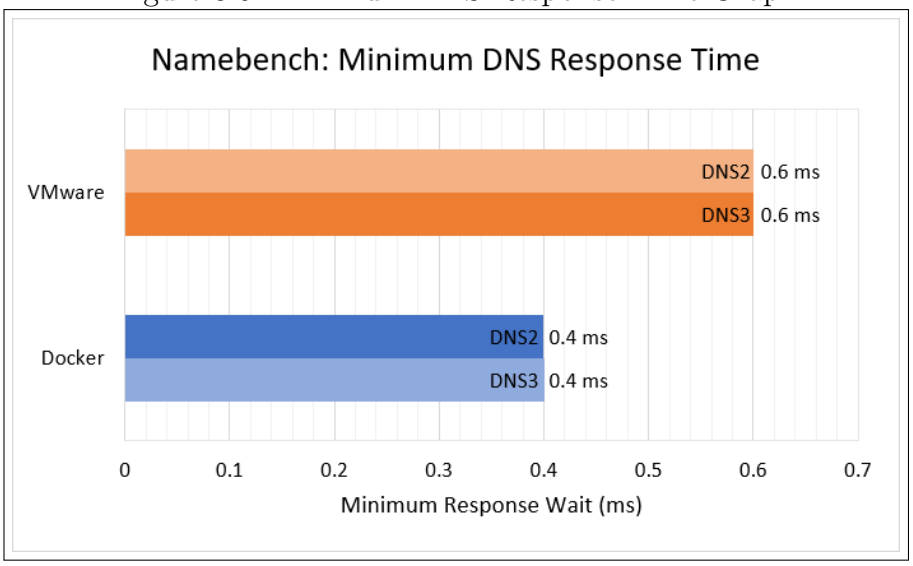
Figure 8.5: Average DNS Response Time Graph



We can see the minimum latency response in figure 8.6. Here, Docker takes the win again, with a similar relative margin between results

as with the average latency shown above. Here we can see that the latency for each server is the same within the same system; this helps us to rule out an element of chance to the results (ie. the results are more likely to be genuine given that they are the same across the two severs for the same system).

Figure 8.6: Minimum DNS Response Time Graph



No maximum latency is given in a graph as in both tests, at least one request hit the maximum wait time for a response (3500ms). This is most likely due to an error with authoritative servers outside of our control. Furthermore, 95th percentile data could not be used (such as in our last test in subsection 8.2.2) because Namebench compiles the the minimum, average and maximum data itself, without allowing the user to access raw data. This is most likely because the domain name list provided by Alexa [Alexa Internet, 2021] is propriety, so providing access to this data could be a breach of Alexa's usage agreements (This is speculation, not to be treated as fact).

8.4 Test 4 - Apache JMeter and Netdata

8.4.1 Test parameters

This test is slightly different, in that the data we get back from the workload (JMeter) isn't the most important piece of data we receive by doing the test. Instead, we are using the workload that is

generated using JMeter [Apache Software Foundation, 2021], in order to measure the stress on the machine (though we do still look at the data JMeter found). The three areas of Computing performance we will be looking into here are the CPU usage, RAM usage and the 'Load' on the machine. These were measured using Netdata[Netdata, 2021], so that the data could be monitored on another PC without affecting the performance of the PC running the Containers and Virtual Machines. Netdata also creates real-time graphs that can can then be saved in snapshots, so that we can investigate the results in depth.

Idle data was also collected from Netdata. In these idle tests, the whole system was live, but without the client machine running. When the test with the JMeter load was done, the whole system, *including* the client was active. The reason for taking idle data, was so that we have a baseline figure to compare to for when the system was under stress.

JMeter was used to test the Apache server, and the JMeter tool was installed and run from the client (the same as the other tests). JMeter works by the user creating a test plan in which a number of different parameters can be set to create a test workload [Apache Software Foundation, 2020b] that is exactly what the user requires. The below parameters were specified for JMeter:

- A new thread group was created. A thread group is essentially a group of simulated users, with each individual 'thread' being one user [Apache Software Foundation, 2020b, 3.1 Thread Group], that to Apache, will appear as a different client. This is what allows JMeter to do stress testing.

- Within this thread group, we set the number of 'threads' to 100. The ramp-up period was also set as 100. The ramp up period is the amount of time (in seconds) that Jmeter takes to start all threads [Apache Software Foundation, 2020b, Section 3.1]. The time between starting one thread and starting the next is equal to $\frac{RampUpPeriod(s)}{ThreadCount}$ [Apache Software Foundation, 2020b, Section 3.1]. In this case, $\frac{100}{100} = 1$. So the time between each thread start is one second. The loop count within this thread group was set to 10, this means each thread (user) will perform the

test 10 times before that thread closes.

- Now that we have our thread group, we need to assign it a task. This is done by adding both a 'sampler' and a 'configuration' element. A sampler allows us to do various requests[Apache Software Foundation, 2020b, Section 3.2.1] but in this case we will be doing a HTTP request. The configuration element, allows us to change the options within the sampler [Apache Software Foundation, 2020b, Section 3.6]. For our test we used the 'http request defaults' config element; the only change we make is to set the domain to `www.intranet.co.uk`.
- Finally, we need to add a listener. A listener is a way to collect the results that are generated by JMeter [Apache Software Foundation, 2020b, Section 3.3]. As we want to be able to compile the results ourselves, we used the 'view results in table' listener [Apache Software Foundation, 2020a, Section 18.3], which allows us to export the results to a CSV file for easy data manipulation later.

8.4.2 Results

JMeter

Table 8.2 shows the average latency for both the Docker System and the VMware system. We can see that Docker has a reduced average when compared to VMware; further strengthening support for the Docker system.

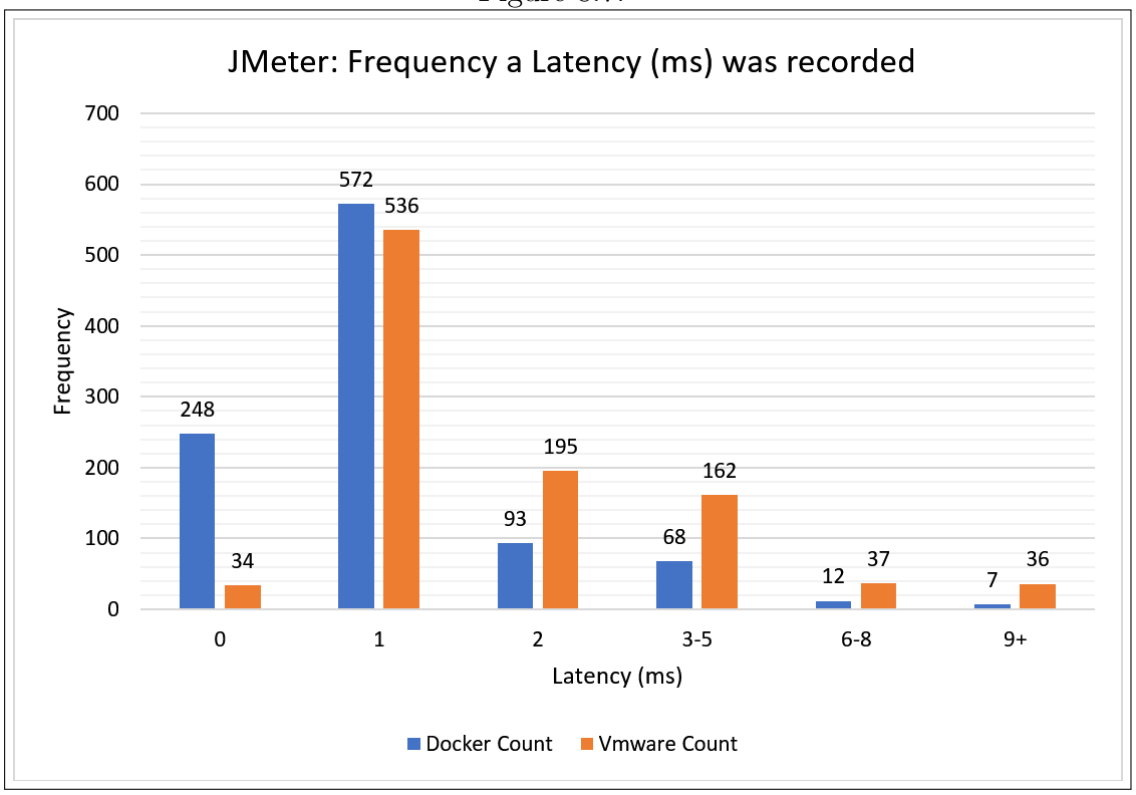
Table 8.2:

AVERAGE LATENCY (ms)	
Docker	VMware
5.85	8.88

The parameters we set gave us an output that had the latency for 1000 HTTP request transactions. To better illustrate the spread of latency, Figure 8.7 shows the frequency of these latencies. Frequency relates to the number of times a certain latency (in ms) was recorded. A latency of 0ms means the system took less than 1ms to process

the HTTP request. Some latencies have been grouped together as they became less frequent.

Figure 8.7:



Firstly, we can see that Docker had far more response times in the 0ms category. This suggests the Docker system often didn't have a backlog of requests due to be being saturated meaning it was still able to give almost instant replies. Moving to the next category, we can see that both Docker and VMware have similar numbers of requests in the 1ms range, however, Docker is still slightly ahead. Moving on to the 2ms range, we can see than VMware starts to take over as having more counts of this latency. Moving past this latency we can see that VMware has far more responses in the higher latency ranges, with 36 counts in $9ms+$ range, compared to only 7 counts for Docker.

If we dig deeper into these 9+ results, we can see that some of them are incredibly high in comparison to what we would expect. For Docker, of the seven results in the 9+ category, five of them are in in the magnitude of 10^1ms. One of the results is in the magnitude of 10^2ms but is also the first result in the series, so can be omitted

as a 'warmup' result. The one other result here is $4635ms$, which clearly shows an erroneous result.

To look at VMware's 36 results in the 9+ category it is easier to break it down into table 8.3.

Table 8.3: Magnitude of latencies for VMware that are over $9ms$

Latency Magnitude (ms)	**Count**
10^1	28
10^2	6
10^3	2

From this table, we can see that six results took between $100ms$ and $999ms$(One of these is also the first result, so should be considered a 'warmup' result, as we have done the same with Docker. A further two results take $1036ms$ and $3996ms$. When compared with Docker, the results demonstrate how these anomalous results or 'hitches' in the system were more common in VMware.

Netdata

CPU Usage: First, we will look at idle CPU usage. It is important to remember that in subsection 8.4.1 we discussed that the idle usages were taken without the client running. This means that the idle usages are reflective of the usage we would see in a live environment on this hardware, if the system was receiving no traffic.

Figure 8.8 shows Docker's CPU usage at idle

Figure 8.8: Docker Idle CPU

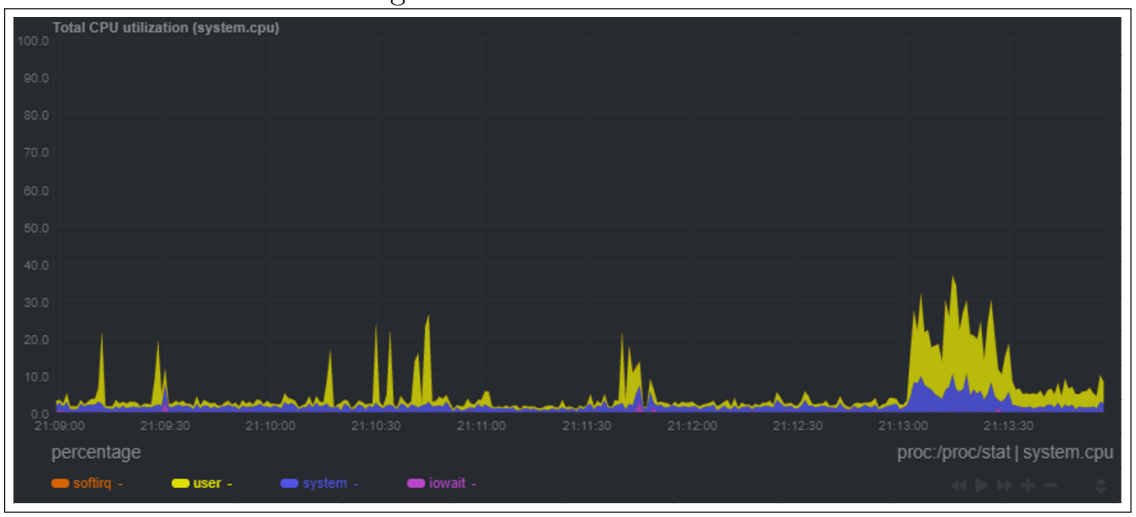
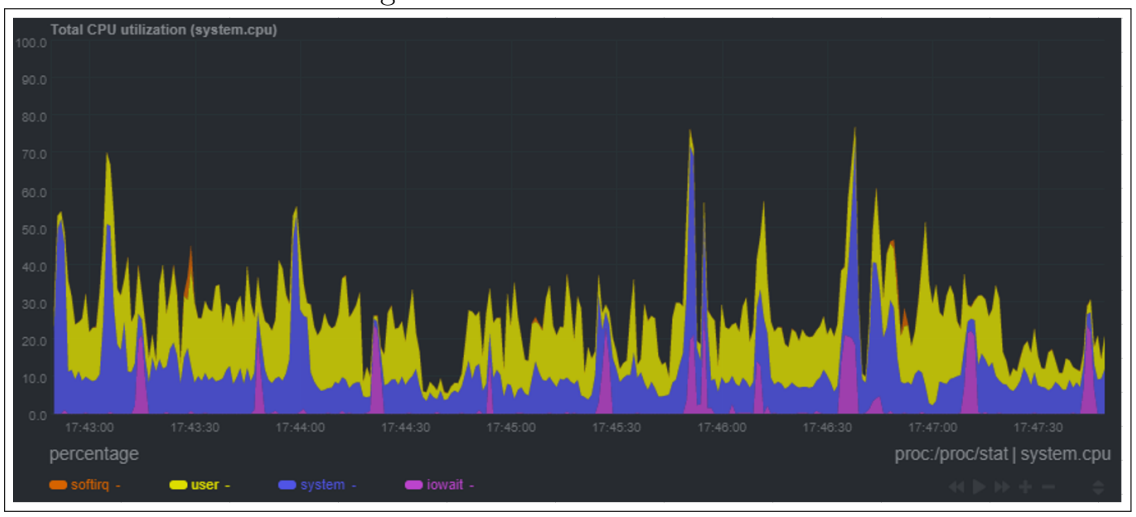


Figure8.9 shows VMware's CPU usage at idle.

Figure 8.9: VMware Idle CPU



Already we can see that VMware is using far more resources from the CPU when idle compared to Docker. Figure 8.10 shows the CPU usage of the Docker system for when the JMeter test was being performed.

Figure 8.10: Docker JMeter Workload CPU

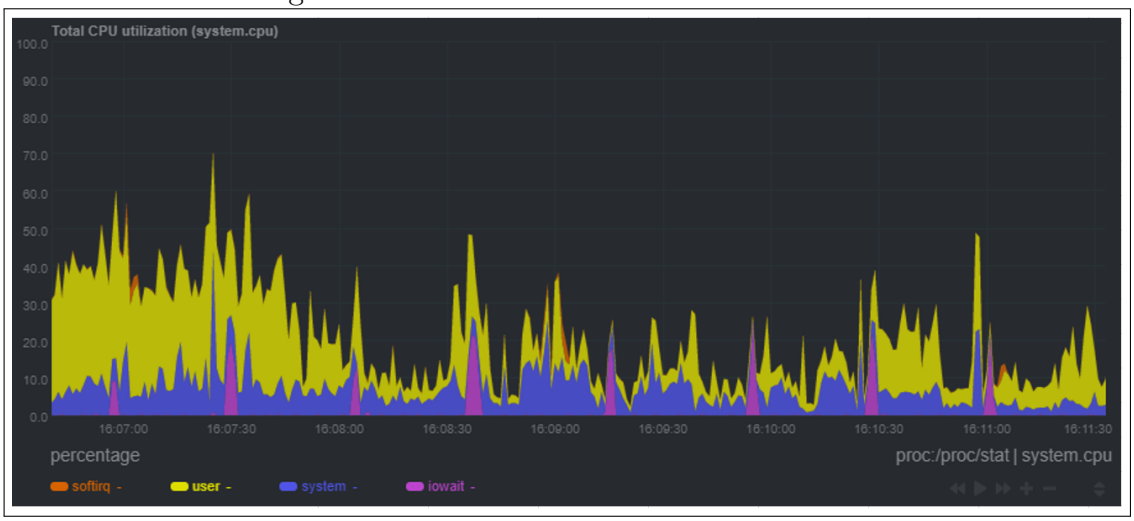
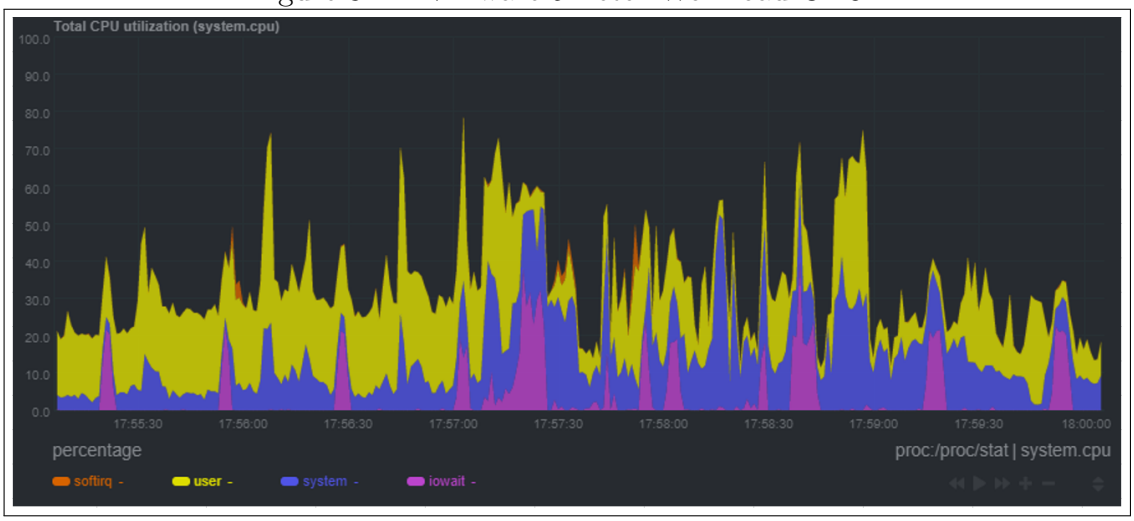


Figure 8.11 shows the CPU usage of the VMware system whilst under stress from the JMeter test.

Figure 8.11: VMware JMeter Workload CPU



Whilst the difference during a workload is less noticeable, it is still clear that VMware's CPU usage is more than Dockers. Interestingly, we can see that Docker's CPU usage ramps up at the start of the test, but begins to drop once the test has been running for some time. This might suggest that Docker manages consistent workloads more efficiently.

RAM Usage: Here we will investigate the RAM usage on the system. To start, Figure 8.12 shows the machine's RAM usage whilst

running the Docker system idle. We can see from this, that when idle, the docker system takes up very little RAM on the host machine.

Figure 8.12: Docker Idle RAM

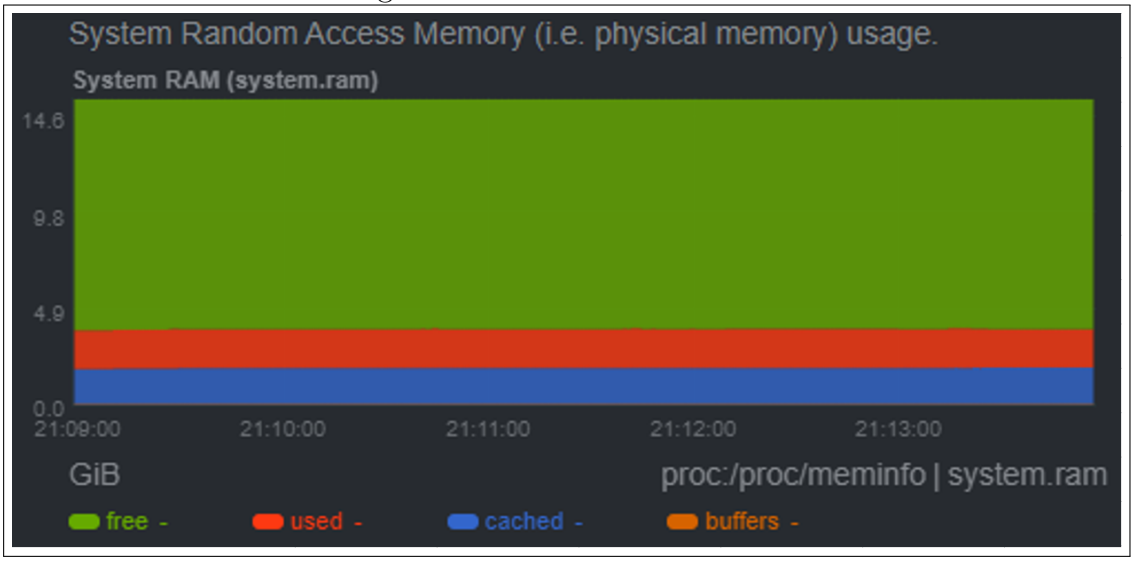
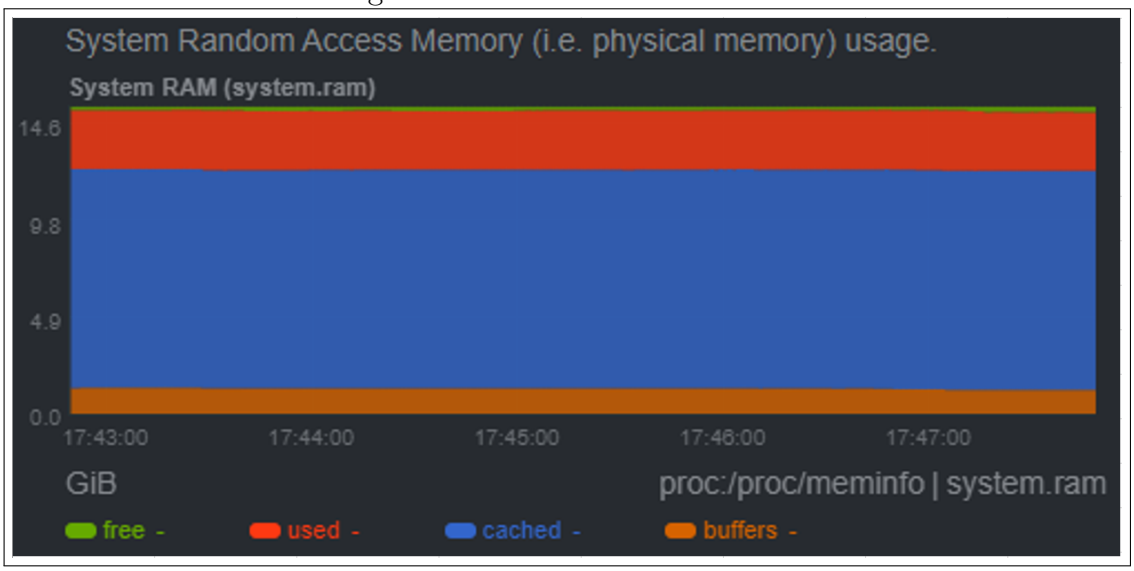


Figure 8.13 shows the RAM usage on the host machine when the VMware system is running in idle. We can see that there is a very tiny amount of free RAM, which could be a considerable bottleneck to the system, or could cause it to thrash [Denning, 1968]. However, the portion of RAM actively being 'used' is also fairly small (still more than the Docker system, but not unreasonable). Instead most of the RAM is is cached. This is most probably due to the way that VMware partitions away sections of RAM for its virtual machines. The ram we see cached is most likely being used to host each individual virtual machine.

Figure 8.13: VMware Idle RAM



Moving onto Docker under a load from the JMeter test (Figure 8.14), we see that there is a slight increase in 'used' RAM, and considerably more ram being cached. There is still a healthy amount of RAM left free for the system to use, so no thrashing should take place.

Figure 8.14: Docker JMeter Workload RAM

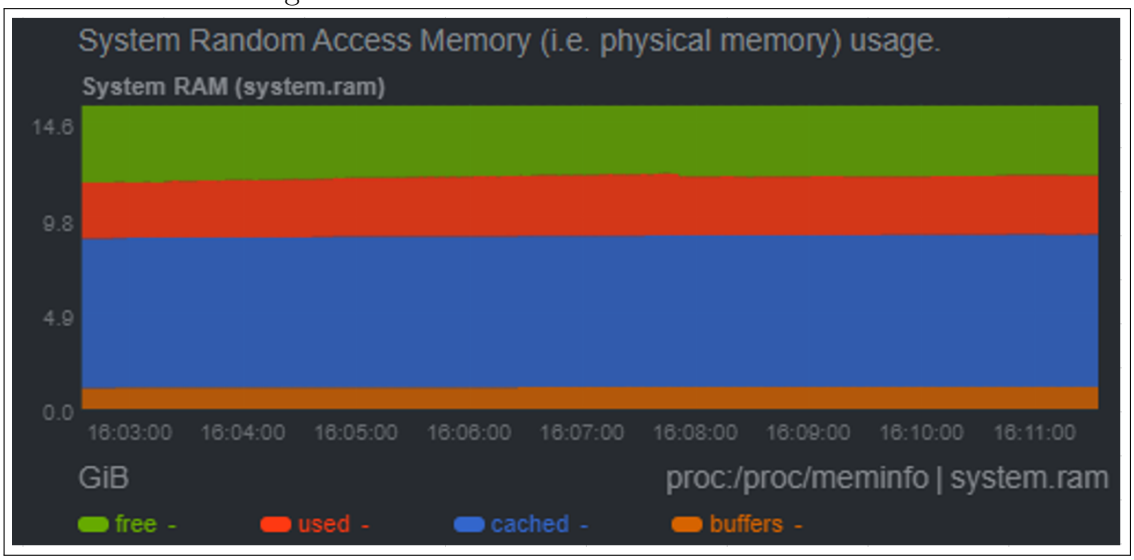
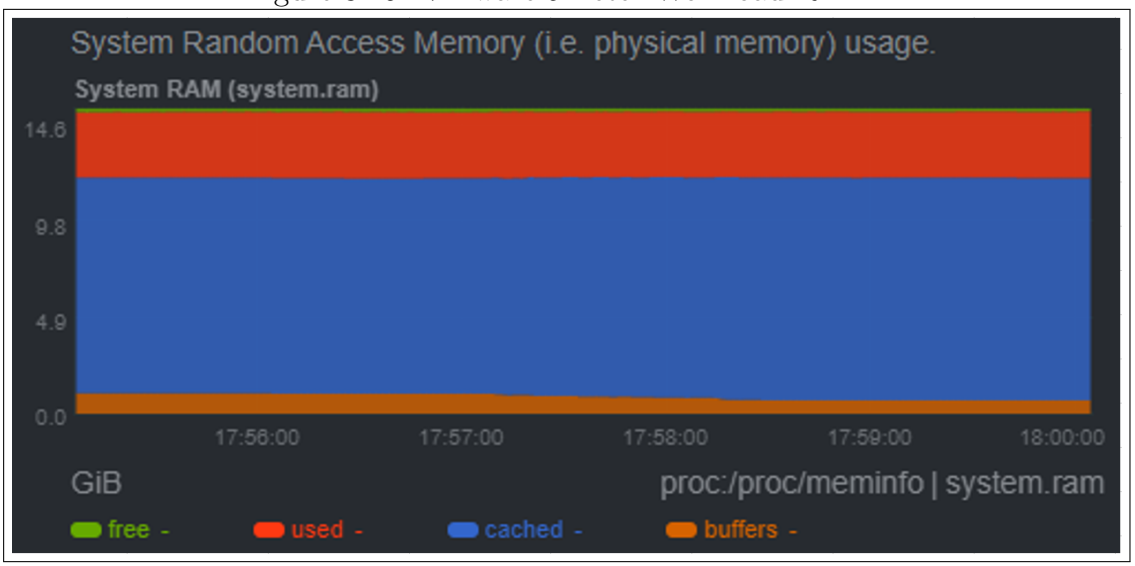


Figure 8.15 shows the RAM for the host machine whilst under a workload from the JMeter test. There is very little change between the usage here and the usage at idle (Figure 8.13), other than the RAM buffers being slightly reduced. This is because even at idle, the system is already fully committed; there is no free ram for the

machine to move into. This might explain why some of our previous results for VMware, such as in subsection 8.1.2 (Test 1 Results), are unstable.

Figure 8.15: VMware JMeter Workload RAM



Load: Whilst CPU and RAM usage are self explanatory, 'load' may seem more vague. Load as measured by Netdata, is defined in the same way that Ferrari et al. defines load, whereby load is based on the number of tasks using the CPU and I/O along with those that are waiting in a queue [Ferrari and Zhou, 1987]. This is as opposed just using CPU utilisation on its own, as this combined approach gives better results as to the actual extended load of the system. By measuring load in this way, we are also tying in other parts of the hardware, as load could increase with little CPU utilisation, if for example the machine was waiting for memory resources to be clear. Netadata accesses this data by using linux's built in "`/proc/loadavg`" file [Fedora, 2012].

By looking at load in this way, we effectively tie the two measurements above (CPU and RAM) with other hardware (for example; the disk) to see what the combined affect is to the machine.

Figure 8.16 shows the Netdata load graph for Docker whilst idle. In this graph, 'load1', 'load5' and 'load15' refer to the number of of tasks waiting over one, five and fifteen minute averages respectively to be 'done' (that is to mean, that task has been completed by the

computer). Generally low results across the board is best, but with waits in the higher ranges obviously being worse. We can see in this graph that 'load1' has a few spikes, however the scale of this graph is actually designed to show load at all times. If we look at the y axis, we can see that the axis is actually showing 0.70 as the highest number, which relatively speaking means less than one task a second is taking 5 minutes in the queue.

Figure 8.16: Docker Idle Load

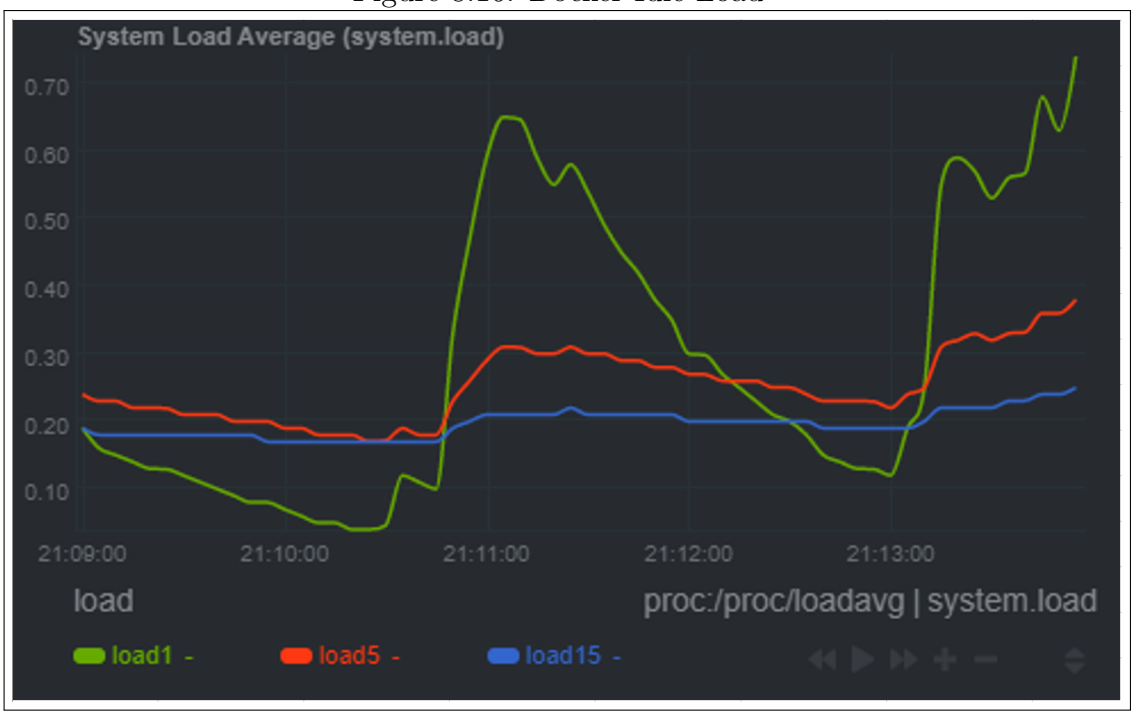


Figure 8.17 shows the idle load for VMware. Here, we can see just how much of an affect on task completion times having multiple virtual machines running has on the system. Again, notice the y axis goes much higher here, and that there are more tasks waiting 15 minutes, than there are 5 or 1. Around three and a half tasks are waiting more than fifteen minutes to be processed.

Figure 8.17: VMware Idle Load

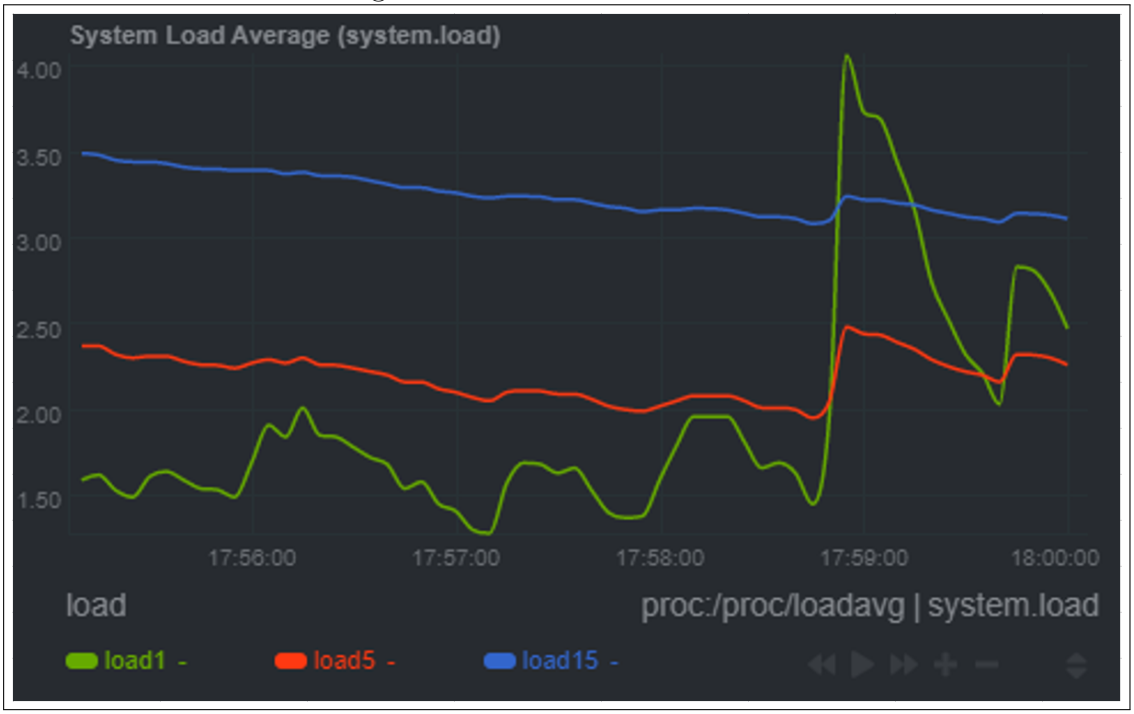


Figure 8.18 shows the Load on the host machine when docker was under stress from the JMeter test. We can see from the Y axis that around two tasks are waiting more than a minute to be completed at peak. There is around one task at worst waiting 15 minutes.

Figure 8.18: Docker Workload Load

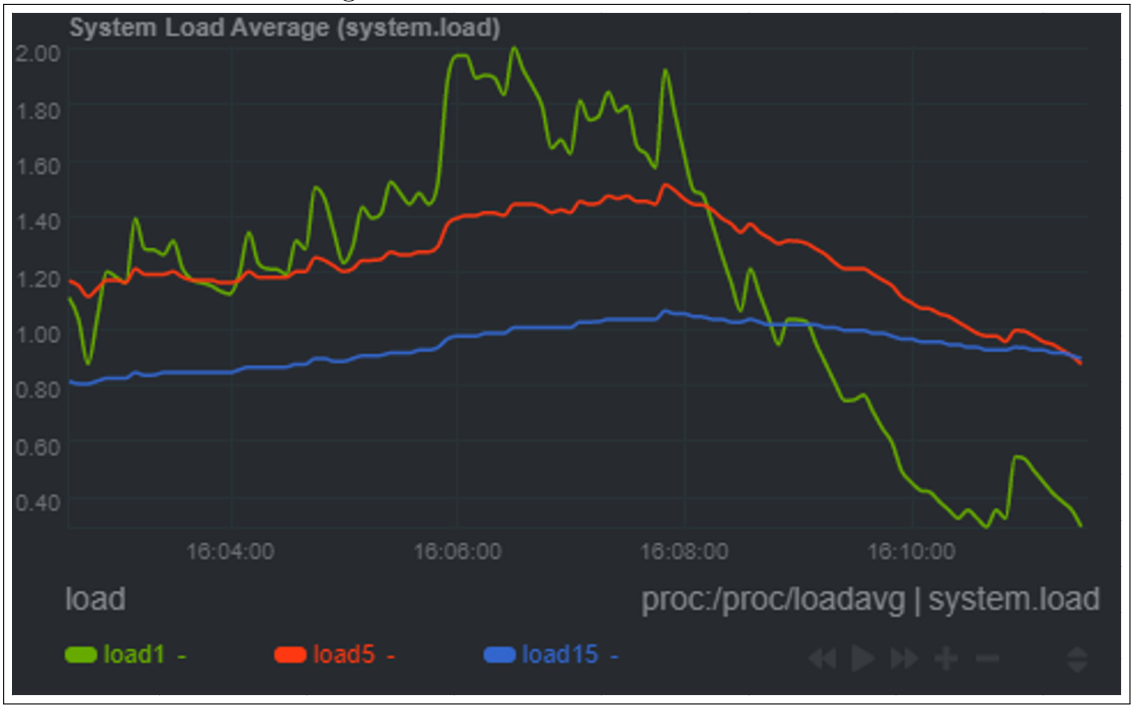
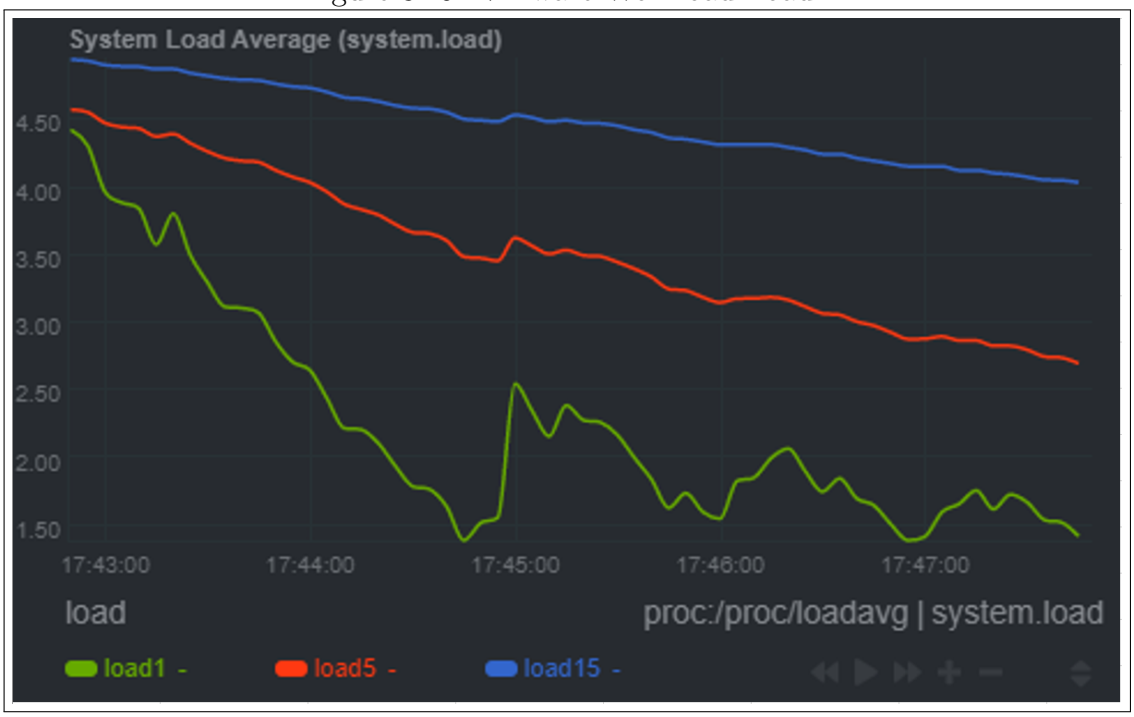


Figure 8.19 displays the Load during the same JMeter test, but on the VMware system. We can see here that more tasks are taking longer than 15 minutes than otherwise. Looking at the y axis, we can also see that the bars are sitting far higher than the previous graphs. Interestingly, we can see that load is on a downwards slope even during a workload. So whilst load here may be very high, we could maybe hypothesise that should the load continue long enough, that the machine may reach some sort of equilibrium.

Figure 8.19: VMware Workload Load



0

Part III

Evaluation

Chapter 9

Evaluation of the Product

9.1 Evaluation of the systems

9.1.1 Successes

Both systems (The VMware system and the Docker system) do exactly what was described in the analysis. The network topology for both of the real systems is exactly the same, with both of these networks matching the topology that was designed during the analysis section (Figure 5.1, in subsection 5.4.6). The system maintained a LAMP topology, as was specified, along with the various other components of the system such as DHCP and DNS. This is an important part of the research because it demonstrates that this kind of system can be supported by containers, and not just virtual machines, as is done historically. Furthermore, the configuration for matching services across the two systems (ie. DHCP on VMware; DHCP on Docker) are identical, ensuring that results remained comparable.

Separately, the VMware system is not particularly impressive, or groundbreaking, whilst the container based system on the other hand *is* impressive, and potentially groundbreaking. This is because there are not many examples of systems such as this one being built for use outside of small scale development operations (whereby the network infrastructure couldn't be supported on a real LAN). Together, the two systems combine to form an important base for research that could potentially impact the current paradigm of virtualisation in which we currently find ourselves in. The actual results, and their impact will be discussed further in section 9.2, but without the work

that was conducted to ensure that both systems worked exactly as designed, those results wouldn't have been useful. This is due to the fact that the testing was a direct comparison of systems; should there have been any disparity between the VMware network and the Docker network, the results would not be a fair comparison.

This required a level of perseverance when it came to finding something within the Docker system that was difficult. For example, getting phpMyAdmin to work on the Docker system wasn't as easy as the rest of the build (as mentioned in subsection 7.2.2), but a workaround was found to make it work, even though there was probably an alternative system that could have been implemented easier. Had the Docker system been the only system being built, and that had been the project basis, then using an alternative method would have been acceptable, and this could have been worked into the synthesis, but with both systems needing to be identical, this was just not an option. In the long run, I think this has made the project more of a success, because it has proven that taking systems that already exist within virtual environments, and converting them to container environments is an option without compromise for those that are in positions to make this move.

The scalability of the system we have created is very good. During development the system was actually entirely designed and created on a separate machine and then moved over to the testing machine after the fact with a fresh installation of Ubuntu. Setting the whole system up on a fresh machine was simple, and there was minimal hitches in the process. This suggests that both the containers and virtual machines were designed and created in an efficient manner. I would suggest that scaling up the number of machines, or adding new functions to the system later, would be an easy task.

9.1.2 Some notable Limitations

This project took place over what became a bizarre year for academia. Due to the Coronavirus pandemic, there was limited ability to access network labs and University infrastructure, so this project had to be scaled in a way that was reasonable for me to complete with my own personal hardware.

RAM and CPU usage

The computer that was used was a desktop PC, and as a result, the hardware reflects that of what is reasonable in a typical desktop machine. I think it would have been interesting, and possibly more reflective of the area I am trying to influence with this research, if the testing could have been done on a purpose built server machine, that may have had more than 16GB of RAM, and possibly a more server focused processor. For example; AMD have recently written a paper showing the use of their new EPYC line-up of server CPUs for hosting containers [AMD, 2019].

However, if we are looking to make recommendations for users that may still use old server hardware, I believe the use of older desktop hardware in this test may actually be *more* compelling to them, as the results may offer those users an alternative should they already be using virtualisation, and looking to get a little bit more usage out of the hardware before an upgrade.

Perhaps if the test was done over a few different systems, we could have compared the usefulness of containers and virtual machines across these systems. This however, would have made the project much larger, and would have been outside the scope of what could be considered a reasonable amount of work.

The Client

As already discussed, access to hardware switches, and with that, the ability to run the network infrastructure outside of a virtual network was not possible. This meant the client within the network had to be hosted on the same hardware as the rest of the network. Obviously, in a real-world environment we would expect a number of clients to be on the same network, but on separate host hardware.

It could be argued that by having the client sitting on the same machine, that the results are skewed, as delay to the processing as a result of the client using CPU time and RAM could affect how the other virtual machines or containers run. This affects objective nine within the Terms of Reference (appendix A), as it may result in inaccuracies within the data. Part of the way through the design

process, it was decided that to ensure that there was no run-off affect on the results, that the client would be the same VMware virtual machine with the same number of resources allocated to it for both tests. By doing so, we ensured that any offset to the results was the same for *both* systems, and thus partially mitigated the affects to objective nine.

There was still some possibility that having VMware running and Docker running at the same time could introduce some sort of fighting for resources between that could negatively affect the Docker results, but given the immense head run that Docker has on VMware, I feel any affect on the results here is negligible, and that we are still in line to have achieved objective nine.

Using VMware's VMnet8 Adapter

Another point of contention within the research is the use of VMware's NAT network adapter 'VMnet8'. The use of this adapter in a link-local state to simulate an environment for the Docker containers could both be seen as a way in which undermining of the results was avoided, but also the contrary.

This is because using the VMnet8 adapter allowed Docker to use the same network environment as the VMware tests. This in theory should remove any discrepancy in results that would result from the network medium instead of the distinction between VMware and Docker. However, it could also be argued that VMware virtual machines using the NAT network directly through VMware could have different performance to Docker containers using the adapter. It could be possible theoretically that the Docker containers are at a disadvantage by using the adapter, due to hidden variables such as bottlenecks in the pass through of traffic over the VMnet8 adapter onto the VMware NAT network.

When looking at the results however, it is clear that any disadvantage to Docker that may be present, won't have changed the main conclusive point of the data. That being that Docker is the better, faster and less resource heavy option for network management.

9.2 Evaluation of the test results

One of the most important parts of the tests in my view is the quantity of data that was generated. Testing was done across the whole system, and those tests focused on both the network performance, and the performance of the hardware itself. This makes the research very valuable, as it gives us a very detailed insight into how containerisation affects all parts of a network topology.

9.2.1 Understanding the test results

All of the tests we performed looked at some degree, into the performance of various network components, with Test 4 offering further insight into the hardware performance of the system whilst performing network tasks. To understand the relationship between the results, we will start with Test 1 (iPerf3, Section 8.1). Here, the tabled results showed (as with all other tests) a clear win for Docker, with an average transfer rate sitting at 116.06 MBytes/s higher than that of VMware's. Looking into the transfer rate over time however, we can see that performance over time for VMware and Docker was not a linear and parallel, like we may expect to see. Instead, whilst Docker's Apache server transfer rate *was* relatively linear, staying around the same level throughout the test with little deviation, we found that VMware's transfer fluctuated. At times, the transfer rate for the VMware Apache server reached and even surpassed Docker's own transfer rate in the same relative time period, but it would then quickly drop back down to levels that were less than half of what it had been.

This sort of behaviour is something we then started to see in further tests, such as Test 2 (Sysbench MySQL I/O, Section 8.2). In this test, Docker is again the clear winner when looking at averages, with VMware falling behind with around a third of the total queries performed in the same time frame. When we dig deeper into these results however, we see the same trend of fluctuation appearing again as we see that VMware has a minimum latency that is actually slightly smaller than Docker's minimum, but the average and 95th percentile latencies of VMware are far higher than the Docker Average's and 95th percentile latencies. This shows that the range of

results for VMware is considerably larger, suggesting fluctuation like we saw in test one, whilst Docker's results maintain a small range, which suggests those results are more consistent.

Moving onto Test 3 (Namebench, Section 8.3), we see that the Name Servers on the Docker systems can perform queries $\approx 72\%$ faster than the same Name Servers on the VMware system. Where this test falls down, is the inability to dig further into the results in the same way as we did for Test One and Test Two, due to the way that Namebench compiles and presents the results. The data crunching that takes places to find the averages is behind a screen on Namebench, and the raw data is never presented to the user, instead, only the Minimum, Average and Maximum results are given. This is most likely because the Alexa Top Sites data that the test uses is a propriety data set that may be protected under rules of usage. Nonetheless, this results in a smaller workable data pool when compared to the rest of the tests, and whilst the data collected here is still valuable, it would have been more valuable if we could have looked in detail to see if we could pinpoint if the tendency towards fluctuation in the VMware results, and stability in the Docker results, is present in this test also. Further disappointment in the results from this particular test comes from the maximum latency that was given, which was 3500ms for both VMware and Docker. This result is most likely due to errors on Authoritative servers for one or more of the domains that we tested against resulting in the request being dropped after this exact amount of time (3.5 seconds), and as such, should be treated as anomalous data. Without access to the raw data, we can't actually confirm this and thus it remains merely as hypothesis. This kind of error is to be expected given that the test was against $\approx 38,000$ domains, but access to the raw data would have allowed us to investigate whether or not it was the same domains giving this high output, or even if this error was more frequent on one system over another. The reason Namebench was chosen despite not giving much flexibility and control over the results, is merely due to the fact that there are so few DNS server performance tests. Perhaps the development of a more in-depth DNS benchmarking suite would be beneficial so that interesting behaviours such as those witnessed in this test can be explored in greater detail.

The final test was Test 4 (JMeter and Netdata, Section 8.4). Though we grouped this as one test, we essentially performed two tests in one, those being the networking performance test with JMeter, and then the host machine's hardware performance measured by Netdata during that test (alongside comparison idle data from Netdata also). To start, we will evaluate the network performance measured from the JMeter test. This test was much better than the one in Test Three (namebench) in terms of providing us with a real dataset. The full results from this test were output by JMeter, and this allowed a more in depth analysis of the results. To start though, and as with the other tests, the data was compiled to show the average latency where, unsurprisingly, Docker came out on top again, being on average 3.03ms faster than the equivalent Apache server on the VMware system. When taking a deeper look into these results, like in Test One and Two, we see that the spread of results for VMware is much higher. Whilst Docker and VMware both have most of their results in the 1ms category, we see that VMware has far more results spread across the categories for higher latencies. This is significant because it further shows this trend of fluctuating results that we have witnessed in Tests One and Two. We can also see that Docker has a number of latencies (248) in the less than 1ms category (0ms), whilst VMware only has 34, which further works to solidify that Docker is just faster than VMware in these high stress networking tasks.

Whilst Docker always remains the fastest option, these tests also demonstrate another key finding: that Docker, *in this system at least*, is far more **Reliable** than VMware. In Test One, Two and Three, we see a clear trend towards Docker not only performing well, but being consistent in doing so. VMware can often times pull results that match or surpass the Docker equivalent, it just doesn't maintain these results for any amount of time that could be considered reasonable for a live system in a critical environment and as anyone with Computer Networking knowledge should understand, reliability is a key factor with any live system, not to mention a high input and output OLTP database like the one we are trying to simulate. As discussed in the Analysis and Synthesis, OLTP systems are very common ways of maintaining databases with high user interaction, as as such they need to be able to manage under a large load, as to

not crash or develop errors during peak usage times.

A good way of seeing how often these errors take place for both systems is looking at results which appear to have excessively long latencies (as extremely high latencies would suggest the system is waiting for other resources, which further leads to I/O errors). When looking at Docker, we see than only one result appears to be due to an error, when on the VMware system there appears to be seven results that are due to errors. It is worth a note here mentioning that this method of determining errors is speculation; no actual MySQL error checking was used during the tests. Other reasons could be to blame, but it stands to reason that the most likely reason is I/O errors.

9.2.2 Understanding the fluctuations

By now it is clear that the results present Docker as faster on average, but we have not yet fully understood why the fluctuations on the VMware system that we are observing in tests one, two, and four take place.

This may be one of the areas that the research falls down, as no scientifically sound root cause analysis is possible with the results we have. Currently we can only hypothesise as to why the results present in this way. Despite this, we *can* make hypothesis as to the most likely and logical explanation, that explanation being linked to the RAM (and possibly, but perhaps less so, the CPU) usage. As discussed in subsection 9.1.2, testing was only performed on one Desktop PC with limited resources. If we then look to the second stage (Netdata results) within Test Four, we can the CPU and RAM usage, along with Load on the host machine during the JMeter test and when idle. Looking at RAM specifically, we can see that during the VMware testing there is little to no free RAM. It is most likely, by my own hypothesis, that this is the reasoning behind the seemingly random high latencies, and drops in throughput. Furthermore, the Load increase on the VMware system is considerable, and most likely due to the system needing to quickly to perform a lot of memory swapping (moving memory from RAM to Disk). When this process of swapping memory back and fourth becomes large, then it starts to have an affect on CPU times (reflected in load as measured

by Netdata), at this point, we are experiencing what is known as thrashing [Denning, 1968].

To summarise, whilst Docker is still more suitable than VMware (as it *does* uses less resources), it could be that our results around network performance are being unfair to VMware in that we are not providing enough resources for it to be able to handle the workload we are putting it under. Whilst it could be hypothesised that we would still see similar results without this RAM bottleneck, this research doesn't investigate this any further.

That is not to say that the research isn't still useful, as this recorded behaviour could actually be very useful to some users (Businesses, Network Administrators, IT Infrastructure Workers, etc.). As there may be a number of those users using outdated hardware along with virtualisation, in order to host networks, that are beginning to show signs of the issues we detailed here (i.e. Memory Swapping or Thrashing). For those users, the research in this paper details a possible solution (that being containerisation) that they could do right now, without yet having to update to newer hardware. This option could an appealing solution for those looking to stay in-line with new demands in the networking sector without spending large amounts of money on new hardware.

Nevertheless, should this research be done again I would want to look to do testing on other, more advanced hardware, to confirm that the results remain similar. Similar research that removed hardware bottlenecks that are present in this research could be extremely useful, as it could be better applied to those that are already running modern and fast server machines with ample memory. Sadly, with where the results we see in this paper lay, we are less useful to those very end users.

Chapter 10

Evaluation of the project and process

I believe that this project has been the most defining aspect of my university degree, and of my academic development to date. I have been confident in my writing ability throughout my academic journey, but this project has challenged my writing ability in such a way that I have been forced to self-critique and develop it further. This project has also made me develop numerous skills which I hope to take forward with me into my future work, weather that be in academia or elsewhere. The ways I have improved my abilities as a result of this year long project are numerous, but I hope to touch on them in the following sections.

It is also worth noting here that I hope to touch on my failings as well as my successes. This project has been a way for me to show my skills, and my ability to work on substantial projects; whilst I would love to be able to say that I am a perfect student that makes no mistakes, this would simply be untrue. A good evaluation of one's own ability requires that person to be open about their downfalls.

10.1 Development of skills

10.1.1 VMware

Working with Operating Systems using VMware was the starting point for my research idea. I had previously worked with VMware, and already felt confident in my ability to produce a VMware based

network to a good standard. As a result, the work done around VMware is of very good quality, and I had no issue in achieving the objective to build the virtual machine topology (as laid out in my Terms of Reference objectives). However, I do feel that I have still developed my skills surrounding VMware a considerable amount. The main area of learning being a deeper understanding of the way that VMware's various network modes operate. Before doing this research, I knew how to use these network modes to achieve different results, but I had taken no thought into how they actually work on an Operating Systems or hardware level. I had also never used VMware's Network manager to change aspects of these virtual networks such as disabling automatic DHCP or changing the default IP addressing schemes (both of which were required during the project). Now that I have finished the practical work in the project, I can use hindsight to say that though I did already have a good understanding of how to *use* VMware to create the VMware network, I didn't have the in-depth *knowledge of why* VMware worked, like I do now. When comparing this train of thought to my Terms of Reference Project Plan, it is clear than I should have factored this in. The plan makes little mention of learning VMware skills. Instead, it only makes mention of Docker. This perhaps was due to a level of naivety towards the scale of the project. That is to say, that I underestimated the level of knowledge I would require around VMware in order to fulfil the VMware tasks I set out to accomplish within the project.

Understanding how VMware networks operate also helped me when wanting to test the Docker network system. I had the unique problem of not having a way to network the various bits of traffic between docker containers, as Docker's own network modes didn't easily allow traffic which relies on Unicast, Multicast and Broadcast (Such as DHCP) unless routed to a real network using `macvlan`. My developed knowledge of VMware gave me the idea to route the Docker traffic, using `macvlan`, to the VMnet8 network adapter, and setting the adapter as a link-only network interface. This in-turn gave me more reliable results, as it allowed me to rule out latency of the network medium as a variable, as both systems ended up using the same VMware NAT network.

Had I not developed my VMware network knowledge, I wouldn't have known that this was an option, and the research as a whole would have suffered.

10.1.2 Docker

When starting the project, my understanding of Docker was very minimal. I roughly understood why containers were different to virtual machines, and I had heard of Docker being used for development purposes, but outside of this, I had no understanding of how Docker actually worked.

Taking a look at what I am capable of now, I am extremely pleased with my development of Docker knowledge and skills. I am now very confident in the writing of Dockerfiles, Docker image management, Docker Networking and more. As a direct result of this project work, I believe I have given myself deep insight into an area that should serve me well in my future work. Docker has gained a lot of popularity in recent years, and I am sure that this will continue to be the case into the future, as more and more services come to use it. As a result of the experiences I have had with Docker through this project work, I have gained valuable insight into an area that I am sure will serve me well in years to come.

This sits in line with the objective set in my Terms of Reference (See appendix) whereby I aimed to learn how to 'implement and utilise' Docker containers.

10.1.3 Benchmarking Software

There are two areas within my objectives that relate to my development of skills in benchmarking tools, these being to determine a method for evaluating performance of the two systems, and to accurately measure performance of the systems. I have already laid out both successes and failings of my benchmarking and testing in section 9.2, but it is important to talk about the development of my skills in this area, irregardless of the actual test results.

Firstly, all of the toolsets and programs I have used to benchmark the system were new to me. I had no experience with iPerf3, Sysbench,

Namebench, JMeter, or Netdata before taking on this project, and as such, I had to perform a number of test runs and go through a lot of documentation when using these tools in order to learn how they worked so that I could collect the data needed for the my comparisons. This reflects the first objective mentioned in this subsection; "to determine a method for evaluating performance of the two systems". It also goes further though, as it I required my own effort to adapt and learn these new tools. This is something I feel I may have misjudged when designing my project plan. I made sufficient time for learning Docker, and even to learn about different benchmarks as part of the decision making process for which benchmarks I would use for my work, but I failed to account for the time it would take me to learn these various benchmarks. If I was to do the project again, I would like to spend more time looking at different benchmarks in more detail, so see if I could tune them to get more results. For example, both JMeter and Sysbench are extensive benchmarking suites; the tests I have performed with them in this research are just scratching the surface of what they can do, and I would love to work with them more in the future to see if I can generate some more interesting data.

Despite this, I am of the opinion that the data produced in this project speaks to my ability to pickup new tools quickly and sufficiently, and that this further reflects the second objective I mentioned in this section; "to accurately measure performance of the systems". Most of the tests I performed gave good and accurate results that allowed for detailed comparisons of the two systems. I also think that using the method of using a selection of tests was a benefit to the process of data collection within the project, as it allowed a greater spread of data, with various tests picking up the shortcomings of others in order to create a more well-rounded and concrete benchmark.

10.2 Personal Evaluation

The first part of the project report to be written was the analysis. During this stage, I struggled with the literature review, as it is something I am not particularly used to doing. I have done a lot of technical work, and as such written technical reports. Of course I use

references during these reports, but I rarely go into intricate detail around these studies, reports and papers, and so doing that for this research was something I struggled with. This had a knock-on affect towards the rest of my work in the early stages. My synthesis was pushed back as I failed to manage my time well around my Analysis. Time management is something I have always struggled with in the past however, so I do have my coping methods for when these things happen, as such I have managed to pull back time to sink into this project in a way that hasn't been detriment to the quality of my work. My technical abilities in computing which allowed me to pick up Docker, and the benchmarking suites quickly, were also key to my success here. Should I ever do this project or similar projects in the future, I would be more aware that I need to allocate more time to analysis, as it is something I am still not entirely confident in. If i hadn't been able to take on the project synthesis work in the way that I did, I feel the project may have suffered.

Another area worth mentioning is note taking. I think it would have been useful for me throughout my research, to have kept a sort of diary of thoughts around topics I would like to discuss. Often times my thoughts can be sporadic and messy, and I think a research diary would have helped collate some of these thoughts when writing my dissertation. Instead, there were times when I forgot about certain points that I would have liked to have made, or left sentences unfinished after losing my train of thought. This is something I have struggled with in the past, so I feel I should have made better efforts to maintain some degree of direction between writing sections. That being said, whilst my sporadic and sometimes irregular way of working can be a hindrance to my output, it's important to state that it is also this way of thinking that leads me into research topics and areas that I find interesting. So that is to say, that whilst my way of working could have been managed better, I don't think that it is inherently a bad way of working.

Another area that I overlooked that also set me back was a struggle I had around deciding the best way to create and manage the systems. In my terms of reference, I had initially planned to use a remote access virtual machine infrastructure that would be provided by the university. Once I started to plan for the research further, I realised

this was not the the right option. This was because the virtual machines were a key part of the testing, not just the medium for testing to take place. If I wanted to be able to measure performance of virtual machines and then compare that to Containers, it was clear that the hardware would need to remain the same between tests, and that using remote virtual machines, with no direct access to the host machine, was not going to work. This again links back to the points I have made surrounding issues with the hardware of the machine I was using. Whilst using my personal computer as the host machine wasn't a perfect solution, I do believe that this decision was the right one, as had I continued on the trajectory set in my Terms of Reference document, I feel my final results would not have been as reliable.

One advantage to my project plan was the fact I made time for troubleshooting during my synthesis as part of my learning time. This was useful as it meant I was under less pressure when things went wrong during the creation of machines, or during testing. This is something I would certainly take forward with me into future projects, as I understand that learning only takes place through failure, and that giving myself time to cope when things go wrong is important to the process of project of this scale, especially when using newly learned skills, or when self-teaching.

Despite the above mentioned flaws in my process I still believe that I have created an impressive piece of research, and I am extremely proud of the work I have created.

Part IV

Conclusion & Recommendations

Chapter 11

Conclusions

11.1 Main Test Result Conclusions

The primary aim of this research was to measure performance between Containerised and Virtualised network infrastructure, with the hope of generating recommendations to either those that already use virtual machine infrastructure, or to those that are creating new headless servers and wondering which option to use. With that in mind, I can confidently say that the total output from this work has been a success. A number of in-depth tests have been done to generate a solid set of results. These key and conclusive results are as follows:

- Docker, when being used to run headless servers, performs faster than the same servers perform on a VMware machine.
- Docker can provide in most cases, over double the performance output when compared to VMware for the same workload.
- Docker uses far less system resources when compared directly with VMware.
- Docker's performance is more stable and reliable than VMware under load.

11.2 Secondary Points of Interest

Other interesting points of discussion are also relevant, but less important than the main discoveries that are laid out above. One of

these discussion points sits with the way that Docker and VMware are both managed. These two different methods are not exclusive to containerisation or virtualisation respectively, but when making recommendations based on the work done in this research, these points are still important to mention.

- Docker is managed firstly from the command line. Dockerfiles can be written to quickly create new Docker Images, or, Images can be pulled directly from Dockerhub or a private repository. Images that are already created can be saved to .Tar files, or can be pushed to a repository. Multiple containers can use the same image, as IP addressing is done when launching a container. Networks which bridge to real interfaces must be created and defined by the user, using Docker's macvlan network driver. Macvlan is not the primary, or most supported network option provided by Docker.

- VMware is managed using VMware's workstation software. Virtual machines must either be created using the setup wizard using a real OS image file, cloned from another machine (If the original machine is deleted or lost, the clone will fail), or a direct copy of the VM can be created (by manually copying the Virtual Machine files). VMware comes ready with a 'bridged' network, along with virtual networks that can connect with other services via adapters. The bridged network option is well supported by VMware.

I believe these are important points to mention as they should be considered by anyone that is planning on using this research to form an opinion about which technology to use. Neither of these methods is the right or wrong way of managing containers and virtual machines, but as they are different, it may still be an important basis for that persons decision making. For example, Docker's command line interface could for some end users such as network administrators be a feature they prefer over VMware's GUI interface. On the other hand, those with less technical knowledge might find VMware more user friendly. Another example is the way that Containers and Virtual Machines are stored. Both allow ample opportunity for backups, and rapid redeployment of server solutions should the need

arise (though it could be argued that Docker is faster). These points are less quantitative, and more qualitative it is entirely up to the end user as to what matches their needs.

That being said, I do think that VMware's solution is more user friendly even if hardcore IT users may prefer a CLI over a more GUI. Someone with CLI knowledge may prefer that as an option, but they certainly wouldn't struggle to use VMware. The same could not be said the other way around. Docker is first and foremost a low resource solution, and as such should be used from the Command Line. Docker does have a 'Docker Desktop' version that works on Windows and Mac, but this version uses different techniques to simulate the containers (for example, on windows it uses Windows Subsystem for Linux [Docker, 2021b]), and as such we can't make recommendations for these versions of Docker, as there may be other constraints on container performance.

It is also important to talk here about the way that Networks are managed in both solutions. Docker's networks are generally configured by the user (a default virtual network is provided but not recommended). For containers to face a real network, the macvlan driver is used. This network mode isn't the most supported network option, and there can be problems regarding the handling of some IP traffic. This is evidenced by the issues around DHCP that we experienced in subsection 7.4.1. When looking at VMware, the network management is far more robust. The systems that VMware uses to bridge virtual machines to real networks are far better supported, and for all intents and purposes, function exactly like a real machine would.

11.3 Rounded Conclusion

To sum up, Docker is a far better option than VMware when looking solely at performance metrics and resource usage. VMware was far slower, and as a result of it's high resource utilisation, it suffered fluctuation making it less reliable under load. However, Docker isn't designed around this kind of use, and as a result can suffer from being less straight and operable. VMware provides a more robust

system for managing its instances. VMware's Bridged network mode is also more desirable than Docker's macvlan mode.

It is clear that more work needs to be done to provide a suitable base for containerised networking specifically. The performance benefits to using Containers over Virtual Machines is a one that should be utilised, but Docker, as it stands now, probably isn't the best way forward. Should Docker want to provide this as a part of their platform in the future, they need to work on better ways to implement containers with real networks.

I am sure that once these issues are ironed out, that container based networking will be the future. I would hypothesise that we will start to see less and less virtualised infrastructure in favour of containers simply due to being able to get the same performance out of lesser hardware (something that will save costs).

Chapter 12

Recommendations

12.1 For those considering moving to Containers

This section will comprise of information and recommendations to those that may want to use this research as part of their decision making process about moving to Containers to support their network infrastructure. Whether that be moving from virtual machines to container on an already existing machine, or whether it be part of a decision as to which technology a new system should be using.

The performance results in this report speak for themselves. Docker *is* the superior choice if performance and reliability is the only metric that matters to you or your organisation. If there are no other considerations for your use-case, then the recommendation here would be that containerisation is absolutely the correct choice. However, it would be very unlikely that these are the only considerations. Security, Cost, Ease of Use, and Management Options are all important aspects to consider when deciding how to host network infrastructure.

12.1.1 Security

Security in computing is often split into the CIA triad (Confidentiality, Integrity, Availability). Lets split both the technologies (Docker and VMware), along with the results we have collected, into the CIA triad, and see how they compare.

Docker

Confidentiality is something we didn't investigate during this research, but it has been suggested by other research that isolation is a problem with Docker. I would strongly recommend anyone planning on moving to Containers to read the paper by Watada et al. "Emerging Trends, Techniques and Open Issues of Containerization: A Review" (2019). This paper covers security and isolation concerns with containerisation which are important but outside of the scope of this research.

When we move to Integrity, we can comfortably say that this research tested this well, and we find that Docker's integrity stands up under scrutiny. The servers and services hosted via Docker were reliable and stable, and could hold up even when under heavy loads and stress testing.

Availability was also tested during this research. All the systems stayed up and running at reasonable levels throughout testing, with only minor hiccups that we would expect to see in any operational system anyway.

VMware

Again, we haven't touched on Confidentiality in our study as it was outside of our scope. However, VMware's isolation from the host machine, and between virtual machines is very well documented. For example; De Lucia discusses this in their "Survey of security isolation of virtualization, containers, and unikernels" (2017).

Where our research does interject is when looking at Integrity of data. The processing we saw in our tests, namely the database test (where important data would most likely be exchanged in a real system) showed that we had scattered results where we would observe very high latencies. As discussed in our evaluation, these were probably due to errors. The number of these errors we noticed was significantly more than that of Docker. So if you are planning on using VMware to host mission critical infrastructure, we would remind you that VMware may introduce these hitches that could be detrimental to the integrity of your data.

Availability is another area affected by VMware observed performance hits. The fluctuation caused by this lack of performance in our tests could result in end users sometimes not having access to systems during peak times, as sudden drops in performance happening quite often. This is not acceptable for any system that need to be used by a number of users.

Recommendation based on security

Our recommendation, then, would be that should Confidentiality be the main priority of your system, Docker is probably not the solution for you. If Integrity and Availability of your data is more important to you though, then we would recommend Docker over VMware.

12.1.2 Costing

Another interesting point to mention here is cost of applications. VMware Pro (which is required to use the network manager) is part of a paid software package, whereas everything we did in our tests was possible using Docker's free plan. It is more than possible that however that businesses using Docker may want private repositories or other features that are only provided in Docker's paid plans. It is also possible, that an organisation using VMware, may not require access to the network manager, as bridged connections shouldn't require much (if any at all) configuration. The cost difference's between the two technologies at this point then simply comes down to the preference and requirements of the organisation setting out to use either of the technologies.

12.1.3 Concluding recommendation

From what is discussed in the sections above we can see that the use of VMware or Docker isn't clean-cut. It entirely comes down to the individual needs and requirements of the entity that is requiring a network solution. So whilst our study does a lot to prove the efficiency and performance benefits of containers, it cannot be used as the sole indicator of whether you should used Containers over Virtual Machines.

This moves us onto our next, and final section.

12.2 Suggestions for further research and development

As discussed in the section above (Section 12.1), there are multiple reasons other than performance that come into the adoption of a new technology. Someone could develop the fastest database system in the world, but there would be very little uptake on the technology if it was also the least secure database system. End users need the reassurance that the systems they implement are right for them, and I don't yet feel fully confident in recommending container based networking, as I feel there are still issues that need to be addressed.

For full recommendation of Docker as a tool to host and run networks, it is clear there is work to be done. Firstly, Docker needs to do more work to offer flexibility in it's networking solutions. The macvlan adapter support is simply not good enough, or reliable enough to recommend as a long-term solution on a real network. Secondly, Docker should look to implement a more top-down management view of their containers, that allows end-users mass control over container networks in a similar way as is already possible on Virtualised networks. Furthermore, a deeper focus on the security implications would give end-users the confidence they need when hosting critical servers. Should these few sticking points be straightened I would be able to not only offer Docker as a better alternative in all ways to VMware, I would happily suggest that moving away from Virtualisation and into a new paradigm of Containerisation is inevitable. In-fact, despite the issues listed here, I am almost sure that containers are the future of single-machine networking.

Maybe the issue is that Docker was never designed to be the tool we are trying to turn it into. Docker was created firstly as a way for developers to easily host their applications for testing. It wasn't designed to be used as a permanent networking solution. With that in mind, my recommendation would be that a networking-focused container based solution need to be developed, from the ground up. I believe that a purpose-built containerised network solution, that ad-

dressed the issues discussed in this paper such as real world network cross-over, or security and isolation, would find themselves as a new leading industry standard. I also have no doubt that various Virtualisation and Containerisation solution providers are already working on these kinds of solutions. We are already starting to see uptake of containerisation in areas such as Platform as a Service (PaaS), whereby containers *are* being used in an online environment, in order to provide cost-effective application environments [Chang et al., 2010, Page 55].

Bibliography

Alexey Akopytov. sysbench, Feb 2020. URL https://github.com/akopytov/sysbench#features.

Ameer Alaasam, Gleb Radchenko, and Andrei Tchernykh. Comparative analysis of virtualization methods in big data processing. 6: 48–79, 05 2019. doi: 10.14529/js190107.

Alexa Internet. Alexa topsites, Apr 2021. URL https://www.alexa.com/topsites.

AMD. Amd epyc™ 7002 series processors leadership performance with containers, Aug 2019. URL https://www.amd.com/system/files/documents/EPYC-7002-Containers-Leadership-Performance.pdf.

Radhwan Y Ameen and Asmaa Y. Hamo. Survey of server virtualization, 2013.

Apache Software Foundation. 18. component reference, Nov 2020a. URL https://jmeter.apache.org/usermanual/component_reference.html#View_Results_in_Table.

Apache Software Foundation. 3. elements of a test plan, Nov 2020b. URL https://jmeter.apache.org/usermanual/test_plan.html.

Apache Software Foundation. Apache jmeter™, Mar 2021. URL https://jmeter.apache.org/.

Anja Bog. *Benchmarking Transaction and Analytical Processing Systems: The Creation of a Mixed Workload Benchmark and its Application*. In-Memory Data Management Research. Springer Berlin Heidelberg, 2013. ISBN 9783642380709. URL https://books.google.co.uk/books?id=fV0NAAAAQBAJ.

David Booth, Hugo Hass, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard, Feb 2004. URL https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.

Contel Bradford. Battle of free virtualization tools: Vmware vs. virtualbox, Jul 2020. URL https://blog.storagecraft.com/battle-of-free-virtualization-tools-vmware-vs-virtualbox/.

BSD Certification Group. Bsd usage survey, Oct 2005. URL http://www.bsdcertification.org/downloads/pr-20051031-usage-survey-en-en.pdf.

Canonical. Canonical publishes lts docker image portfolio on docker hub, Nov 2020. URL https://ubuntu.com/blog/canonical-publishes-lts-docker-image-portfolio-on-docker-hub.

Canonical and tianon. Docker: Ubuntu, Mar 2021. URL https://hub.docker.com/_/ubuntu.

Emiliano Casalicchio. Autonomic orchestration of containers: Problem definition and research challenges. In *VALUETOOLS*, 2016.

W.Y. Chang, H. Abu-Amara, and J.F. Sanford. *Transforming Enterprise Cloud Services*. SpringerLink: Springer e-Books. Springer Netherlands, 2010. ISBN 9789048198467. URL https://books.google.co.uk/books?id=yyiPyIXgbxMC.

Wai-Kai Chen. *The electrical engineering handbook*. Elsevier Academic Press, 2005.

Tom Collins. Hyper-v vs. vmware: Which is best?, Dec 2019. URL https://www.atlantech.net/blog/hyper-v-vs.-vmware-which-is-best.

Jonathan Corbet. Process containers, May 2007. URL https://lwn.net/Articles/236038/.

R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, 1981. doi: 10.1147/rd.255.0483.

Michele Cyran. 3 application and system performance characterist-

ics, Dec 1999. URL `https://docs.oracle.com/cd/A87860_01/doc/server.817/a76992/ch3_eval.htm#2680`.

Datanyze. Containerization market share report: Competitor analysis, n.d. URL `https://www.datanyze.com/market-share/containerization--321`.

Michael J De Lucia. A survey on security isolation of virtualization, containers, and unikernels. Technical report, US Army Research Laboratory Aberdeen Proving Ground United States, 2017.

Peter J Denning. Thrashing: Its causes and prevention. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 915–922, 1968.

Docker. docker commit, Apr 2021a. URL `https://docs.docker.com/engine/reference/commandline/commit/`.

Docker. Docker editions, Apr 2021b. URL `https://hub.docker.com/search?q=&type=edition&offering=community`.

Docker. Use macvlan networks, Apr 2021c. URL `https://docs.docker.com/network/macvlan/`.

Docker. Docker overview, 2021. URL `https://docs.docker.com/get-started/overview/`.

Docker. Docker security, May 2021a. URL `https://docs.docker.com/engine/security/`.

Docker. Install docker desktop on windows, Apr 2021b. URL `https://docs.docker.com/docker-for-windows/install/`.

Docker. Docker desktop wsl 2 backend, Apr 2021c. URL `https://docs.docker.com/docker-for-windows/wsl/`.

Docker. Why docker?, May 2021d. URL `https://www.docker.com/why-docker`.

R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs containerization to support paas. In *2014 IEEE International Conference on Cloud Engineering*, pages 610–614, 2014. doi: 10.1109/IC2E.2014.41.

Jon Dugan, Seth Elliott, Bruce A Mah, Jeff Poskanzer, and Kaus-

tubh Prabhu. iperf - the ultimate speed test tool for tcp, udp and sctptest the limits of your network internet neutrality test, Mar 2021a. URL https://iperf.fr/.

Jon Dugan, Seth Elliott, Bruce A Mah, Jeff Poskanzer, and Kaustubh Prabhu. iperf 3 user documentation, Mar 2021b. URL https://iperf.fr/iperf-doc.php#3doc.

M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer. Performance evaluation of iot data management using mongodb versus mysql databases in different cloud environments. *IEEE Access*, 8: 110656–110668, 2020. doi: 10.1109/ACCESS.2020.3002164.

Fedora. E.2.15. /proc/loadavg, 2012. URL https://docs.fedoraproject.org/en-US/Fedora/17/html/System_Administrators_Guide/s2-proc-loadavg.html.

Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. Technical report, CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, 1987.

Philip J Fleming and John J Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986.

Jenny Fong. Are containers replacing virtual machines?, Aug 2018. URL https://www.docker.com/blog/containers-replacing-virtual-machines/.

Carlo Fragni, MD Moreira, DM Mattos, LHM Costa, and OCM Duarte. Evaluating xen, vmware, and openvz virtualization platforms for network virtualization. *Universidade Federal Rio de Janeiro*, 2010.

FreeBSD. Freebsd handbook. URL https://docs.freebsd.org/en/books/handbook/introduction/.

FreeBSD. Freebsd 4.0 release notes, Mar 2000. URL https://www.freebsd.org/releases/4.0R/notes/.

G2. Best server virtualization software, Apr 2021. URL https://www.g2.com/categories/server-virtualization.

GIGA-BYTE Technology Co. Specification: Motherboard: H81m-s2pv (rev. 1.0), 2013. URL `https://www.gigabyte.com/uk/Motherboard/GA-H81M-S2PV-rev-10/sp#sp`.

GitHub. Git commit, May 2021. URL `https://github.com/git-guides/git-commit`.

Suzanne Goldlust. Isc timeline, Feb 2021. URL `https://www.isc.org/blogs/isc-timeline/`.

Google. Google public dns, May 2021. URL `https://developers.google.com/speed/public-dns`.

Red Hat. What is a hypervisor?, 2021. URL `https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor`.

Scott Hogg. Software containers: Used more frequently than most realize, May 2014. URL `https://www.networkworld.com/article/2226996/software-containers--used-more-frequently-than-most-realize.html`.

IBM. Lamp stack explained, May 2019. URL `https://www.ibm.com/cloud/learn/lamp-stack-explained`.

Institute for telecommunication sciences. queuing delay, Aug 1996. URL `https://www.its.bldrdoc.gov/fs-1037/dir-029/_4318.htm`.

Intel. Intel® core™ i5-4670 processor (6m cache, up to 3.80 ghz) product specifications, 2017. URL `https://ark.intel.com/content/www/us/en/ark/products/75047/intel-core-i5-4670-processor-6m-cache-up-to-3-80-ghz.html`.

Bruce L Jacob. *Synchronous DRAM Architectures, Organizations, and Alternative Technologies*. 2004.

Grant A Jacoby et al. Critical business requirements model and metrics for intranet roi. *Journal of Electronic Commerce Research*, 6(1):1, 2005.

Hemant Jain. Lxc and lxd: Explaining linux contain-

ers, Jun 2016. URL https://www.sumologic.com/blog/lxc-lxd-linux-containers/.

A. M. Joy. Performance comparison between linux containers and virtual machines. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 342–346, 2015. doi: 10.1109/ICACEA.2015.7164727.

Krzysztof Ksiazek. How to benchmark performance of mysql & mariadb using sysbench, Jun 2018. URL https://severalnines.com/database-blog/how-benchmark-performance-mysql-mariadb-using-sysbench.

Brandon Lee. Virtual switches in hyper-v: Short guide, Feb 2017. URL https://www.nakivo.com/blog/hyper-v-networking-virtual-switches/.

Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, and Jianping Wu. Measuring query latency of top level dns servers. In *International Conference on Passive and Active Network Measurement*, pages 145–154. Springer, 2013.

A. B. Lindquist, R. R. Seeber, and L. W. Comeau. A time-sharing system using an associative memory. *Proceedings of the IEEE*, 54 (12):1774–1779, 1966. doi: 10.1109/PROC.1966.5261.

Linux Containers. Lxc introduction, May 2021a. URL https://linuxcontainers.org/lxc/introduction/.

Linux Containers. What's lxd?, May 2021b. URL https://linuxcontainers.org/lxd/.

Linux Containers. Network configuration, May 2021c. URL https://linuxcontainers.org/lxd/docs/master/networks.

Craig Loewen. Announcing wsl 2, May 2019. URL https://devblogs.microsoft.com/commandline/announcing-wsl-2/.

Craig Loewen. Comparing wsl 1 and wsl 2, Sep 2020. URL https://docs.microsoft.com/en-us/windows/wsl/compare-versions#whats-new-in-wsl-2.

James Messer. [3.11] what is propagation delay? (ethernet physical layer), Feb 2021. URL https:

//stason.org/TULARC/networking/lans-ethernet/3-11-What-is-propagation-delay-Ethernet-Physical-Layer.html.

Microsoft. Supported windows guests, May 2016. URL https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/supported-guest-os.

Microsoft. Introduction to hyper-v on windows 10, Jun 2018. URL https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.

Microsoft. What is windows subsystem for linux, Jul 2020. URL https://docs.microsoft.com/en-us/windows/wsl/about.

Paul Mockapetris. Domain names - implementation and specification. RFC 1035, November 1987. URL https://tools.ietf.org/html/rfc1035.

Terry Myerson. Hello world: Windows 10 available on july 29, Jun 2015. URL https://blogs.windows.com/windowsexperience/2015/06/01/hello-world-windows-10-available-on-july-29/.

Netdata. Netdata overview, Mar 2021. URL https://www.netdata.cloud/product/.

netdata. Step 7. netdatas dashboard in depth, Mar 2021. URL https://learn.netdata.cloud/guides/step-by-step/step-07.

Oracle. Virtualbox user manual (online), Apr 2021a. URL https://www.virtualbox.org/manual/.

Oracle. Oracle vm virtualbox, May 2021b. URL https://www.oracle.com/virtualization/virtualbox/.

Oracle. Chapter 7 examples of common queries, Apr 2021c. URL https://dev.mysql.com/doc/mysql-tutorial-excerpt/8.0/en/examples.html.

Oracle. 13.3.1 start transaction, commit, and rollback statements, Apr 2021d. URL https://dev.mysql.com/doc/refman/8.0/en/commit.html.

Rani Osnat. A brief history of containers: From the 1970s till now, Jan 2020. URL https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

Milos Pavlik, Roman Mihal, Lukas Lacinak, and IVETA ZOLOTOVA. Supervisory control and data acquisition systems in virtual architecture built via vmware vsphare platform. In *The 16th WSEAS International Conference on Circuits. Kos Island: WSEAS*, pages 389–393, 2012.

Emerson Pugh. *Building IBM : shaping an industry and its technology*. MIT Press, Cambridge, Mass, 1995. ISBN 9780262161473.

Q-Success. Usage statistics of web servers, Feb 2021a. URL https://w3techs.com/technologies/overview/web_server.

Q-Success. Usage statistics of linux for websites, Feb 2021b. URL https://w3techs.com/technologies/details/os-linux.

Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. 2004.

451 Research. Application containers will be a $2.7bn market by 2020, 2017. URL https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf.

A.G. Rodrigues, B. Demion, and P. Mouawad. *Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter*. Packt Publishing, 2019. ISBN 9781839218200. URL https://books.google.co.uk/books?id=D_amDwAAQBAJ.

Spiceworks. The 2020 state of virtualization technology, 2020. URL https://www.spiceworks.com/marketing/reports/state-of-virtualization/.

Thomas Stromberg. Namebench, May 2010. URL https://code.google.com/archive/p/namebench/.

The Go Authors. Frequently asked questions, what are go's ancestors?, Jan 2021a. URL https://golang.org/doc/faq#ancestors.

The Go Authors. Frequently asked questions, what are the guiding

principles in the design?, Jan 2021b. URL https://golang.org/doc/faq#principles.

The Linux Documentation Project. 1.3. /bin, May 2021a. URL https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/bin.html.

The Linux Documentation Project. 1.9. /lib, May 2021b. URL https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/lib.html.

G. K. Thiruvathukal, K. Hinsen, K. Läufer, and J. Kaylor. Virtualization for computational scientists. *Computing in Science Engineering*, 12(4):52–61, 2010. doi: 10.1109/MCSE.2010.92.

Ubuntu and Docker. Ubuntu's docker profile, Feb 2021. URL https://hub.docker.com/u/ubuntu.

Ubuntu Documentation Team. Ubuntu server guide, Mar 2021.

UpGuard Team. Lxc vs docker: Why docker is better, Apr 2021. URL https://www.upguard.com/blog/docker-vs-lxc.

VMware. Explore vmware cloud solutions, May 2021a. URL https://www.vmware.com/uk/cloud-solutions.html.

VMware. Vmware guest os compatibility guide, May 2021b. URL https://www.vmware.com/resources/compatibility/pdf/VMware_GOS_Compatibility_Guide.pdf.

VMware. Network address translation (nat), Apr 2021c. URL https://www.vmware.com/support/ws5/doc/ws_net_configurations_nat.html.

VMware. Changing the networking configuration, Apr 2021d. URL https://www.vmware.com/support/ws45/doc/network_configure_ws.html.

VMware. Adding and modifying virtual network adapters, Apr 2021e. URL https://www.vmware.com/support/ws5/doc/ws_net_configurations_changing_vadapters.html.

VMware. Workstation for linux : Linux virtualization for everyone, May 2021f. URL https://www.vmware.com/

content/vmware/vmware-published-sites/au/products/workstation-for-linux.html.

VMware. Vmware product guide, Apr 2021g. URL https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/product/vmware-product-guide.pdf.

VMware. Vmware workstation pro evaluation, Mar 2021h. URL https://www.vmware.com/uk/products/workstation-pro/workstation-pro-evaluation.html.

VMware. What is esxi?: Bare metal hypervisor: Esx, May 2021i. URL https://www.vmware.com/uk/products/esxi-and-esx.html.

VMware. Hypervisor, n.d. URL https://www.vmware.com/topics/glossary/content/hypervisor.

J. Watada, A. Roy, R. Kadikar, H. Pham, and B. Xu. Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, 7:152443–152472, 2019. doi: 10.1109/ACCESS.2019.2945930.

Chris Wolf. Let's get virtual: A look at today's server virtualization architectures. *Burton Group Data center Strategies*, pages 1–42, 2007.

Łukasz Zemczak and Canonical. Release notes: Ubuntu 20.04 lts (focal fossa), Feb 2021. URL https://wiki.ubuntu.com/FocalFossa/ReleaseNotes.

Part V

Appendices

Appendix A

Terms of Reference

A.1 Background

Virtualisation has been utilised by a number of industries for a long time now, with first iterations of virtual machines dating back to IBM in the 1960s [Pugh, 1995]. System Virtual machines allow an operating system to emulate the function of a full operating system layered on top of a base operating system. Functionally, this allows multiple different logical 'computers' with varying operating systems to run on one physical computer. In most modern implementations, virtualisation requires software (known as a hypervisor) to manage and create the virtual machines.

Whilst I was on a year-long placement provided as part of a sandwich course at my university, I had the chance to work in an IT risk department at a reputable enterprise in Newcastle Upon Tyne. whilst working there I witnessed first hand, infrastructure and operations departments using virtualisation for much of the internally and externally facing server infrastructure, in what was an unwieldy and cumbrous use of scripts to update and install patches and dependencies across a multitude of systems. This resulted in a number of incidents where systems had to be pulled down in order to update the Operating Systems of individual virtual machines. These methods often caused unnecessary down time for systems, resulting in substantial risk for the business. Furthermore, this method often put a substantial stress on the hardware of these systems, with hardware boxes often running at full resource potential under heavy

load, and whilst these boxes were designed to withstand these kinds of loads, it still affected the longevity of the hardware.

In recent years there has been research into the use of containers instead of virtual machines for various operations across computing industries [Watada et al., 2019]. Containers are different from virtual machines in that they run on the base OS, and don't require a secondary emulated operating system to be managed by a hypervisor. This is a benefit as it reduces the resources [Joy, 2015] used by each instance. Overall, containers are a more lightweight and efficient system, that are easier to keep up to date. Whilst research has claim in displaying the benefits of containerisation, much of the server infrastructure of enterprise remains reliant on Virtualisation, not Containerisation.

I have also had the opportunity to partake in gatherings held by CoTech, a network of tech co-operatives that meet semi-regularly to discuss various tech related topics. Whilst I was there, it was discussed that a large portion of these small enterprises used Docker, a system for packaging and deploying containers, to develop applications for their clients.

There is historical research that compares virtualisation and containerisation for other technical applications, for example [Dua et al., 2014] compares the two methods inside the context of PaaS (Platform as a service), which is similar to co-techs use of Docker. It seems that the application for containerisation has been realised as early as 2014 when it is being applied to the development and hosting of applications, but not as much when talking about the running of wider server infrastructure. This is further supported by research from '451 Research' who in 2017 estimated a compound annual growth rate of 40% for containers in 2020 [Research, 2017], suggesting a coming paradigm shift from virtualisation to containerisation.

A.2 Proposed Work

I plan to do similar work to the studies I have already mentioned [Joy, 2015], [Dua et al., 2014], but instead focus this work on my own context and my own interest in Operating Systems and Server

Infrastructure. I have experience with running headless servers on virtual machines already, through the Advanced Operating Systems module I did in my second year of study at Northumbria University.

It is important to set a baseline system here, and to best reflect a realistic topology, I will use LAMP (Linux + Apache + MySQL + PHP). I have experience with Ubuntu server, so I will use this version of Linux as the base operating system for my virtual machines, and plan to host a full working topology of servers, including a DNS architecture, a web-server architecture that includes HTTP (Apache) servers, NFS servers and MySQL servers. The reasoning for doing this is to create as realistic a topology as possible, and to ensure a somewhat realistic level of network traffic so that meaningful data can be captured.

For the virtual machines, I will use the virtual machine infrastructure available to me through the university, and for the container infrastructure, I plan on using Docker. This is because Docker is a popular choice among app developers, and is supposed to make container deployment easy. Though, as I have never worked with containers before, if Docker fails to be a good way of providing containers for web-servers, then there are a number of other ways to deploy containers, such as LXC, that I could turn to as a contingency.

I will also need to set a standard for measurement to ensure that my data is meaningful and uses the scientific method. I plant to use tools such as iPerf3 (for network performance) and Sysbench (for hardware performance) across a number of servers. It is important to measure both network and hardware performance (CPU, Memory, Disk, etc) to ensure that described benefits are achieved for the system as a whole. iPerf is an industry standard tool for measuring network performance on Linux systems, whilst Sysbench is a full benchmarking suite for linux that will let me benchmark the CPU, file IO, and importantly, MySQL performance (allowing direct measurement of database performance on the machine that will be hosting the previously mentioned MySQL database. Whilst I have mentioned these benchmarking tools, it is possible that they could have problems with integrating with certain applications or environ-

ments, in this case, there is a number of other standard benchmark utilities (Phoronix Test Suite, KDiskMark, UnixBench, etc) available that should give me the flexibility to create measurements. Whilst these utilities all have differing overheads within them (meaning they will use up varying resources to run the benchmark), as long as I use the same benchmark across the same machines I am comparing, this overhead shouldn't effect the measurable performance difference between these machines.

Once data is collected, I will do a comparative analysis of the data to determine if there is a clear improvement as previous research suggests, and if this difference is enough to justify a change in the current best-practice use of virtualisation.

A.3 Aims and Objectives

A.3.1 Aims

> To compare and contrast the performance difference and hardware impact between virtualisation and containerisation when running headless servers.

A.3.2 Objectives

Your objective list is a series of measurable objectives, can you tick each one off as *done*? I usually expect between 8 and 12 objectives

1. **Explain the problem domain that encompasses current practices in virtualisation.**
2. **Explanation of the two different methods in practice.**
3. **Determine the software and hardware to be used.**
4. **Design the network infrastructure to be built.**
5. **Learn how to implement and utilise containers using Docker.**
6. **Build the network topology on the virtual machines.**
7. **Build the network topology on the container system.**

8. **Determine a method to scientifically evaluate and determine performance of both systems.**
9. **Accurately measure performance of the two systems.**
10. **Compare and contrast between the findings for both systems to determine improvements or failings.**
11. **Create a recommendation regarding the real-world implementation of containerisation in this use case.**

A.4 Skills

This is where you can cover the skills you have relevant to the project and the new skills you are going to acquire during the project.

1. Advanced Operating Systems 1, see module KF5004.
2. Computer Networking Experience.
3. Computer Technology, see module KF4004.
4. Virtual machine configuration.
5. LAMP topology experience.
6. Docker configuration.

A.5 Resources

I will require access to Virtual Machine infrastructure in order to do build my topology and compare virtual machine performance. I will also require access to a machine that has the same hardware and system resources as those shared by the virtual machines, as I will need to perform my containerisation tests on the same hardware in order to effectively control that variable.

A.5.1 Hardware

I shouldn't require any hardware as long as I can get remote access to the virtual machine infrastructure in a way that allows me to use the same resources for containerisation (as mentioned above).

As contingency, if I cant access this infrastructure, I will still need some way of creating and managing virtualisation. This might be possible on my own hardware, though I'd rather use university infrastructure as this should be more reliable and accessible.

A.5.2 Software

I will need to install Docker, and will need access to benchmarking suites, but these are free and/or open-source.

A.6 Structure and Contents of the Report

Below I will set out the chapters that I will most likely have in my final report.

A.6.1 Report Structure

Abstract & Introduction Here I will set out the background for my project and the reasoning behind the work I will be doing. The Abstract will also summarise the work that has been done along with my findings, subsequent recommendations and conclusions.

What is virtualisation & containerisation An introduction to virtualisation and its current context and usage in today's world of computing. An introduction to containerisation, what it is, how it works, and how it differs to virtualisation. This will form part of my literature review.

Current uses of virtualisation An explanation of modern uses of virtual machines. I will emphasis virtual servers here, and the reasons they are used.

Current uses of containerisation. An explanation of modern usage of containers. I will emphasise application development and other, non-main-sever applications here.

Application of containerisation to server infrastructure Introduce the idea of applying containers to the area I study, with the use-case of server infrastructure. Explain what the implications of this could be. Set out a detailed definition of the problem I want to solve/test. Brief explanation of how this could be designed, with reference to other similar studies.

Network Infrastructure Design Here I will lay out the infrastructure that I want to implement for my testing and my reasoning for this, along with references (see proposed work).

How to test and measure the system Justify the creation or use of a particular set of benchmarks that I will use against the system for testing. I will also explain here how I will measure and evaluate the two systems.

Building the Virtual System Here I will build the virtual system to work with the topology laid out in the infrastructure chapter. I will make sure to explain how to do it. I will also provide evidence of the system working.

Building the Container System Similar to the previous chapter, I will build the container based system following the topology laid out in the infrastructure design section. I will explain how this is done. (ie, using Docker). (see proposed work). As per the previous chapter, I will provide evidence of this working in practice.

Testing and gathering data Here I will create a test plan for running benchmarks across the parallel servers in order to compare the like for like running of both systems.

Analysis & Comparison I will briefly explain how the data shown was required as an introduction to what the data shows. This is where I will explain in detail the actual results, and what they mean, I won't infer here.

Evaluation of the system A technical evaluation of the system I created. I will highlight and strengths, and failings in my ability to meet the requirements of my implementation. I will also ensure to evidence this accordingly.

Evaluation of the project process Here I will evaluate my own learning, and explain what I would do differently about my approach, should I go on to do the same, or a similar, project in the future. My objectives should be assessed as per the marking scheme laid out in the project handbook.

Conclusion and Recommendations I will first create a balanced conclusion, considering the main points mentioned in my evaluation sections. This should cover the direction the work went in, presenting key findings as a way of getting my opinion on the project across. I will present my opinions on the usage of containers in practical setting (ie, whether I think they should be used instead of virtualisation; where they would/could be used) Furthermore, as my work should have practical implications on enterprise and organisations that use virtualisation, I will aim to create recommendations of further work in order to push this area further in the future.

A.6.2 List of Appendices

My appendices will first include my TOR, Ethics form and Risk Assessment.

Some of the network design documentation may be included if too large for the main body, along with the configurations from the various servers.

I should also include the full and direct results of my benchmarks, as these will almost certainly be too large to display in my main text. (I will instead pull the direct numbers from the benchmarks and reference to the related Appendix).

A.7 Marking Scheme

The marking scheme sets out what criteria we are going to use for the project.

Project Type General Computing.

Project Report *Analysis Chapters*

- What is virtualisation & containerisation.
- Current uses of virtualisation.
- Current uses of containerisation.
- Application of containerisation to server infrastructure.

Synthesis Chapters

- Network Infrastructure Design.
- How to test and measure the system.
- Building the Virtual System.
- Testing and gathering data.
- Analysis & Comparison.

Evaluation Chapters

- Evaluation of the system.
- Evaluation of the project process.
- Conclusion and Recommendations.

Product The 'product' includes the Network Infrastructure design, the configurations across both systems, the virtual system and the container system themselves.

Fitness for Purpose

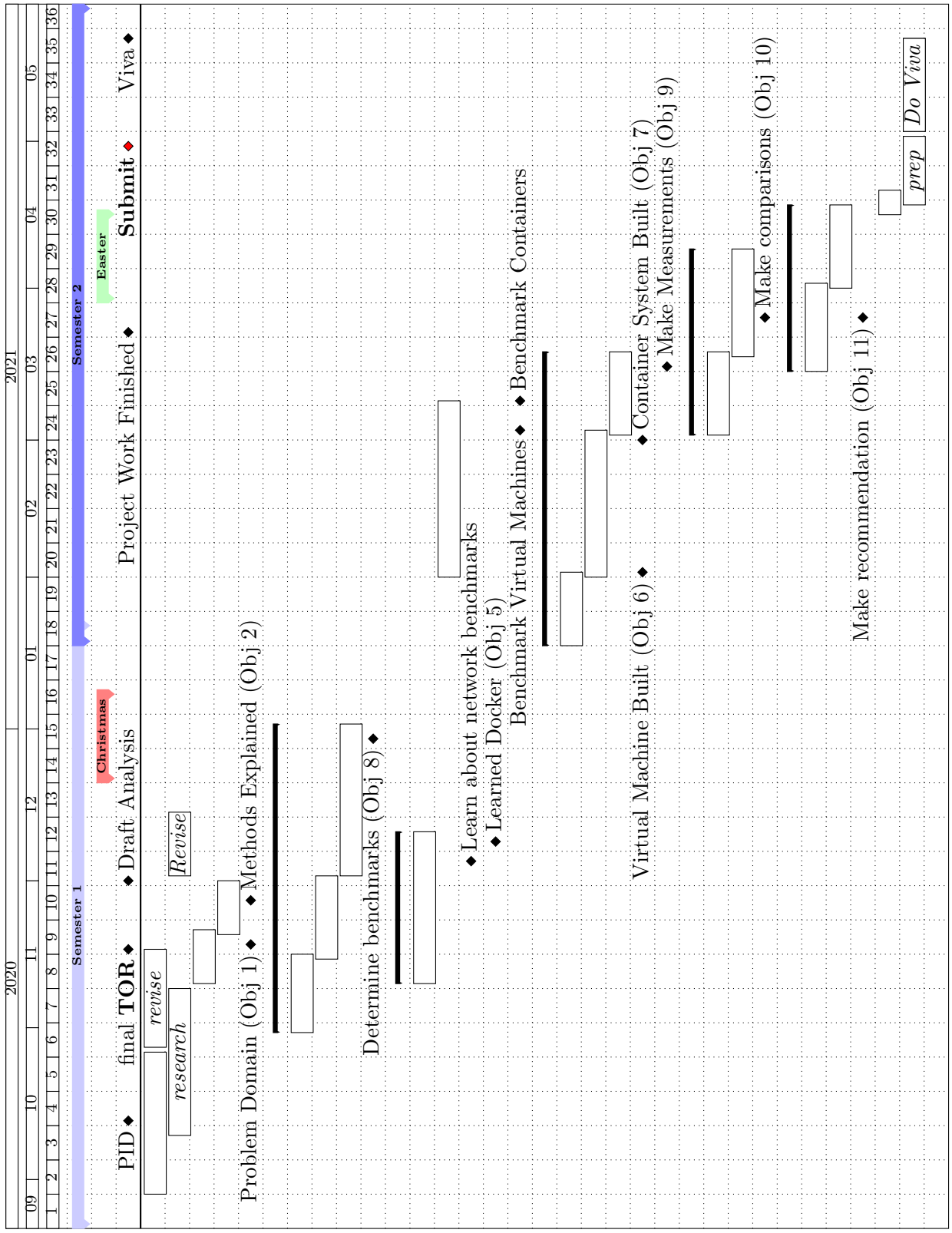
- Meet requirements identified.
- Application to real world infrastructure.

- Whether the headless servers can communicate effectively on both systems.
- Whether both systems achieved the same goals and outcomes. They should have the same logical topology.

Build Quality

- Requirements specification and analysis.
- Quality of the infrastructure design.
- Configuration quality.
- Benchmarking plan quality.

A.8 Project Plan



Appendix B

Test Data

B.1 Test One - iPerf3

B.1.1 VMware Client

```
student@ubuntu:~/Desktop$ iperf3 -c www.intranet.co.uk -f M
Connecting to host www.intranet.co.uk, port 5201
[  5] local 192.168.72.53 port 39864 connected to 192.168.72.150 port 5201

[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  39.7 MBytes  39.7 MBytes/sec    1   1.74 MBytes
[  5]   1.00-2.00   sec   130 MBytes   130 MBytes/sec  757   1.22 MBytes
[  5]   2.00-3.00   sec   166 MBytes   166 MBytes/sec    1   1.29 MBytes
[  5]   3.00-4.00   sec   155 MBytes   155 MBytes/sec    0   1.35 MBytes
[  5]   4.00-5.00   sec   261 MBytes   261 MBytes/sec    0   1.44 MBytes
[  5]   5.00-6.00   sec   306 MBytes   306 MBytes/sec   18   1.17 MBytes
[  5]   6.00-7.00   sec  80.0 MBytes  80.1 MBytes/sec    0   1.21 MBytes
[  5]   7.00-8.00   sec   131 MBytes   131 MBytes/sec    0   1.30 MBytes
[  5]   8.00-9.00   sec   248 MBytes   248 MBytes/sec    0   1.39 MBytes
[  5]   9.00-10.00  sec   211 MBytes   211 MBytes/sec    0   1.47 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  1.69 GBytes   173 MBytes/sec  777             sender
[  5]   0.00-10.01  sec  1.69 GBytes   173 MBytes/sec                  receiver

iperf Done.
```

B.1.2 VMware Apache Server

```
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.72.53, port 39862
[  5] local 192.168.72.150 port 5201 connected to 192.168.72.53 port 39864
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  36.0 MBytes  35.9 MBytes/sec
[  5]   1.00-2.00   sec   131 MBytes   131 MBytes/sec
[  5]   2.00-3.00   sec   166 MBytes   166 MBytes/sec
[  5]   3.00-4.00   sec   155 MBytes   154 MBytes/sec
[  5]   4.00-5.00   sec   263 MBytes   263 MBytes/sec
[  5]   5.00-6.00   sec   306 MBytes   306 MBytes/sec
[  5]   6.00-7.44   sec  78.1 MBytes  54.2 MBytes/sec
[  5]   7.44-8.00   sec   133 MBytes   237 MBytes/sec
[  5]   8.00-9.00   sec   246 MBytes   246 MBytes/sec
[  5]   9.00-10.00  sec   212 MBytes   212 MBytes/sec
[  5]  10.00-10.01  sec  1.68 MBytes   202 MBytes/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.01  sec  1.69 GBytes   173 MBytes/sec                  receiver
```

B.1.3 Docker Client

```
student@ubuntu:~/Desktop$ iperf3 -c www.intranet.co.uk -f M
Connecting to host www.intranet.co.uk, port 5201
[  5] local 192.168.72.50 port 40570 connected to 192.168.72.150 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   280 MBytes   280 MBytes/sec    0   3.04 MBytes
[  5]   1.00-2.00   sec   295 MBytes   294 MBytes/sec    0   3.04 MBytes
[  5]   2.00-3.00   sec   299 MBytes   298 MBytes/sec    0   3.04 MBytes
[  5]   3.00-4.00   sec   306 MBytes   307 MBytes/sec    0   3.04 MBytes
[  5]   4.00-5.00   sec   301 MBytes   301 MBytes/sec    0   3.04 MBytes
[  5]   5.00-6.00   sec   284 MBytes   284 MBytes/sec    0   3.04 MBytes
[  5]   6.00-7.00   sec   285 MBytes   284 MBytes/sec    0   3.04 MBytes
[  5]   7.00-8.00   sec   284 MBytes   285 MBytes/sec    0   3.04 MBytes
[  5]   8.00-9.00   sec   285 MBytes   284 MBytes/sec    0   3.04 MBytes
[  5]   9.00-10.00  sec   288 MBytes   287 MBytes/sec    0   3.04 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  2.84 GBytes   291 MBytes/sec    0             sender
[  5]   0.00-10.00  sec  2.84 GBytes   291 MBytes/sec                  receiver
```

B.1.4 Docker Apache Server

```
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.72.50, port 40568
[  5] local 192.168.72.150 port 5201 connected to 192.168.72.50 port 40570
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   280 MBytes   280 MBytes/sec
[  5]   1.00-2.00   sec   294 MBytes   294 MBytes/sec
[  5]   2.00-3.00   sec   298 MBytes   298 MBytes/sec
[  5]   3.00-4.00   sec   307 MBytes   307 MBytes/sec
[  5]   4.00-5.00   sec   301 MBytes   301 MBytes/sec
[  5]   5.00-6.00   sec   284 MBytes   284 MBytes/sec
[  5]   6.00-7.00   sec   284 MBytes   284 MBytes/sec
[  5]   7.00-8.00   sec   285 MBytes   285 MBytes/sec
[  5]   8.00-9.00   sec   284 MBytes   284 MBytes/sec
[  5]   9.00-10.00  sec   287 MBytes   287 MBytes/sec
[  5]  10.00-10.00  sec  1007 KBytes   285 MBytes/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.00  sec  2.84 GBytes   291 MBytes/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

B.2 Test Two - Sysbench

B.2.1 Test Command

```
student@ubuntu:~/Desktop$ sysbench --db-driver=mysql --mysql-user=sbtest_user --mysql-password=password --mysql-db=sbtest --mysql-host=192.168.72.151 --mysql-port=3306 --tables=2 --table-size=500000 --threads=2 --time=300 --events=0 --report-interval=5 /usr/share/sysbench/oltp_read_write.lua run
```

B.2.2 VMware results

```
SQL statistics:
    queries performed:
        read:                            146860
        write:                           41960
        other:                           20980
        total:                           209800
    transactions:                        10490  (34.96 per sec.)
    queries:                             209800 (699.18 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          300.0632s
    total number of events:              10490

Latency (ms):
         min:                                    6.73
         avg:                                   57.20
         max:                                10959.01
         95th percentile:                      114.72
         sum:                                600016.68

Threads fairness:
    events (avg/stddev):           5245.0000/10.00
    execution time (avg/stddev):   300.0083/0.03
```

B.2.3 Docker results

```
SQL statistics:
    queries performed:
        read:                            432782
        write:                           123652
        other:                           61826
        total:                           618260
    transactions:                        30913  (103.03 per sec.)
    queries:                             618260 (2060.66 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          300.0286s
    total number of events:              30913

Latency (ms):
         min:                                    6.89
         avg:                                   19.41
         max:                                2901.77
         95th percentile:                      23.10
         sum:                               599930.89

Threads fairness:
    events (avg/stddev):           15456.5000/1.50
    execution time (avg/stddev):   299.9654/0.01
```

B.3 Test Three - Namebench

B.3.1 VMware Configuration

▶ **Configuration**

Name	Value
benchmark_thread_count	2
enable_censorship_checks	0
health_thread_count	40
health_timeout	3.75
hide_results	0
input_source	alexa
num_servers	11
ping_timeout	0.5
query_count	250
run_count	1
select_mode	automatic
template	html
timeout	3.5
upload_results	0
version	1.3.1

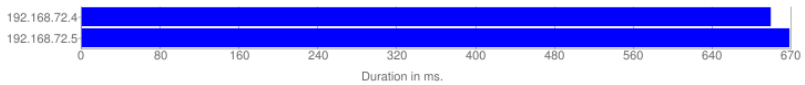
B.3.2 VMware Responses

▶ **Tested DNS Servers**

IP	Descr.	Hostname	Avg (ms)	Diff	Min	Max	TO	NX	Notes
192.168.72.4	192.168.72.4	192.168.72.4	650.44		0.6	3500.0	23	12	• twitter.com appears incorrect: 104.244.42.193, 104.244.42.1 • www.google.com is hijacked: 216.58.213.100 • google.com appears incorrect: 142.250.178.14
192.168.72.5	192.168.72.5	192.168.72.5	668.06	-2.6%	0.6	3500.0	23	10	• www.google.com is hijacked: 172.217.20.132 • google.com appears incorrect: 172.217.169.46 • twitter.com appears incorrect: 104.244.42.1, 104.244.42.65

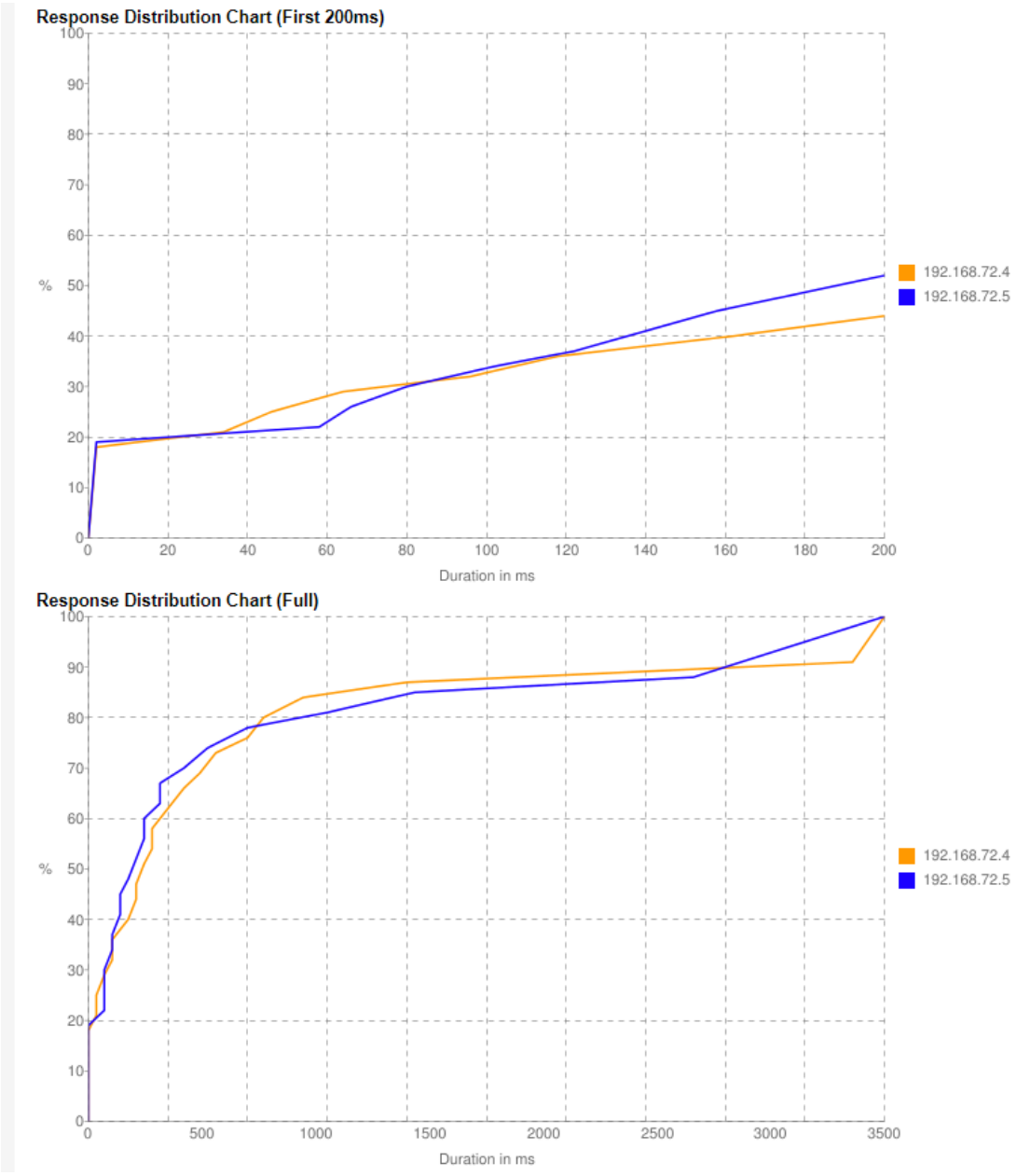
▶ **Graphs**

▶ **Mean Response Duration**



Fastest Individual Response Duration

B.3.3 VMware Distribution Chart



B.3.4 Docker Configuration

▶ **Configuration**

Name	Value
benchmark_thread_count	2
enable_censorship_checks	0
health_thread_count	40
health_timeout	3.75
hide_results	0
input_source	alexa
num_servers	11
ping_timeout	0.5
query_count	250
run_count	1
select_mode	automatic
template	html
timeout	3.5
upload_results	0
version	1.3.1

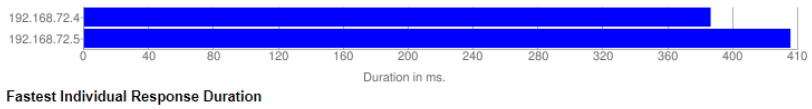
B.3.5 Docker Responses

▶ **Tested DNS Servers**

IP	Descr.	Hostname	Avg (ms)	Diff	Min	Max	TO	NX	Notes
192.168.72.4	192.168.72.4	192.168.72.4	359.76		0.4	3500.0	3	6	• twitter.com appears incorrect: 104.244.42.129, 104.244.42.193
192.168.72.5	192.168.72.5	192.168.72.5	405.61	-11.3%	0.4	3510.7	2	6	• twitter.com appears incorrect: 104.244.42.1

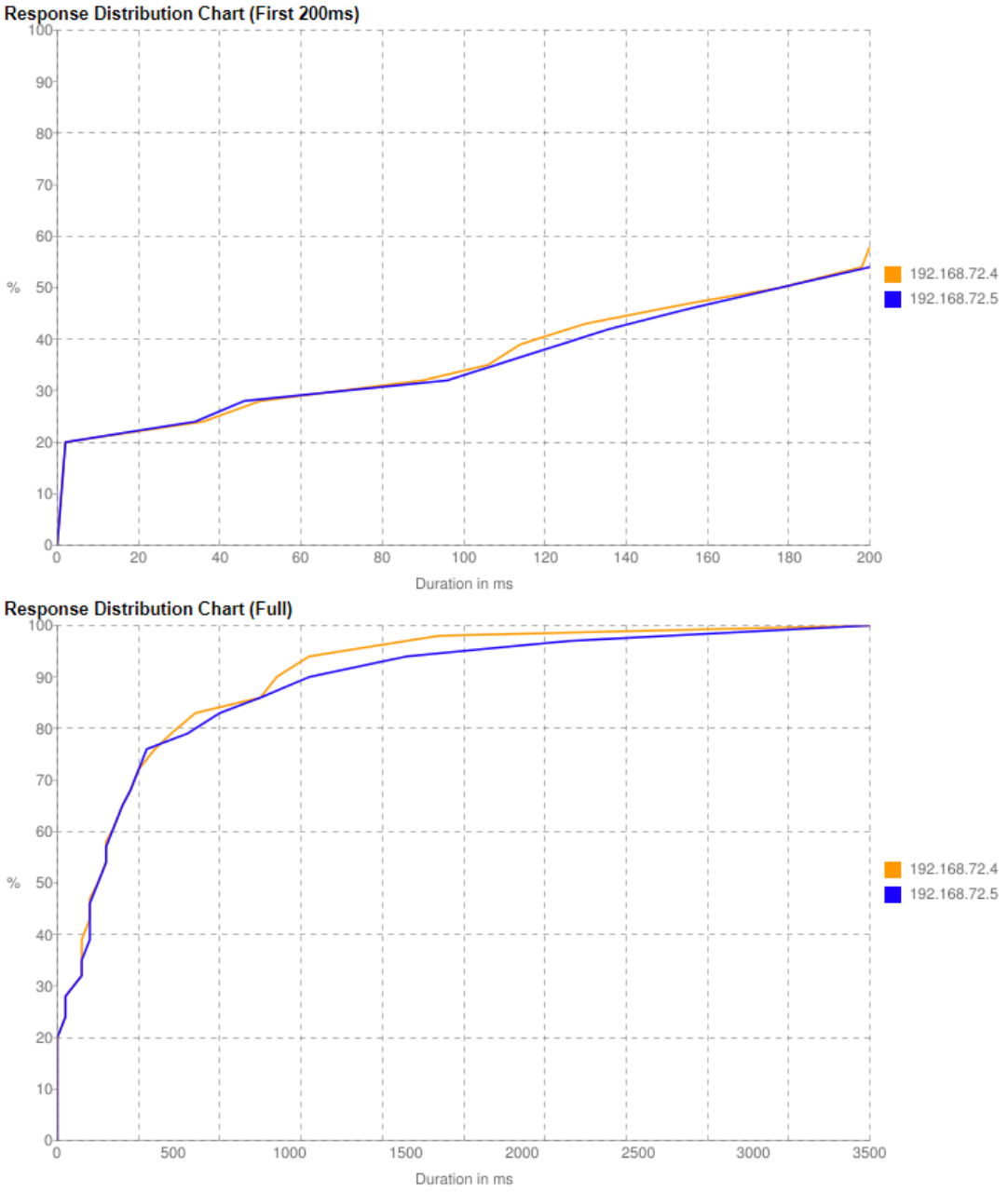
▶ **Graphs**

▶ **Mean Response Duration**



Fastest Individual Response Duration

B.3.6 Docker Distribution Chart



B.4 Test Four

B.4.1 Results

The testing produced 1000 results for each both VMware and Docker. Important results are already referenced in the synthesis part of this report. Raw data .XLS files were submitted as part of the deliverables for this project.