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

Virtualisation vs. Containerisation for Hosting 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 ECHO is on.C:¿set PYTHONPATH=C:.Python.3.9˙qbz5n2kfra8p
C:.Python.3.9˙qbz5n2kfra8p039.exe -S default -f latex -P commandprefix=PYGECHO is on.C:¿set PYTHONPATH=C:.Python.3.9˙qbz5n2kfra8p039C:¿C:.Pytho
C:.Python.3.9˙qbz5n2kfra8p039.exe -S default -f latex -P commandprefix=PYGdefault`texc`

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

Abstract

A summary of the entire project. From background to conclusions. I recon on about half a page as the upper end of the summary.

This is an example structure for the Terms-of-Reference and the Dissertation. Along with some notes.

You can start by forking the repository on github `https://github.com/dr-alun-moon/cs-dissertation`. Then you have a working copy of this document as a starting point.

Contents

Chapter 1

Introduction

This document is split into several files. This makes the editing easier, if the file management a little harder.

The two principle means of including sub parts into a main document are the LaTeX commands `\input` and `\include`

1.1 Splitting the Input

The input command simply reads that file in, at that point in the main file. The result is as if the inputed file was pasted in at the point of the command.

The include command is a little more complex. The main point to note is that it forces a new page, then reads in the file. It is good foe content that needs to start a fresh page, such as chapters. As a rule of thumb I put each chapter in an included file.

1.1.1 Some tricks.

The Dissertation uses the book class, that gives the Chapter as the top sectional element, followed by section, subsection, paragraph. The Terms-of-reference uses the article class, which starts at the section level.

By writing the ToR as a driver document that handles all the setting up, and then inputing the tor content from a separate file. When it comes to including the tor as an appendix in the dissertation, you can start a chapter then just input the tor content file.

```
kappendix
kchaptern+nbTerms of referencen+nb
kinputn+nbtorn+nb
```

1.2 Magic comments

Since LaTeX needs to have a certain amount of setup (known as the preamble), this is the file that needs to be build. From a Makefile or command-line, this is simple enough.

Several editors allow you to trigger the build from within their environment. Here the file you are editing is a subpart and not the mater document that needs to be passed to LaTeX. The editor needs to know which `.tex` file is the master document. Some editors recognise a magic comment placed at the top of a sub-file.

```
c!TeX root=Dissertation.tex
```

This informs the editor that `Dissertation.tex` is the file to build to rebuild the document. I can save changes and just press the key combination set to trigger a LaTeX build, and the editor knows how to do the rest.

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 as a virtual machine. 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 task itself being controlled by a 'hypervisor' (This is explained in more detail in the next sub-section: 2.1.3).

This is an important definition to clarify, as often times containerisation (subsection **??**) can be viewed as a type of virtualisation, and other times not. 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.

2.1.3 Hypervisor

Hardware Virtualisation relies on an underlying software that runs on the already existing OS 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].

2.2 The workings of virtualisation

Virtualisation works

Chapter 3

Review of Containerisation

3.1 Terminology & Definitions

3.1.1 Containerisation

Containerisation has become the de-facto term to describe what could also be described as OS-level virtualisation. For the purposes of this report, I will be referring to this technology only as Containerisation, and each individual instance 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] when they have made their own distinction between Containers and Virtual Machines.

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 the base operating system.

Whilst a hypervisor works by running a full operating system on top of the existing one, a container engine works without that upper operating system by using the base operating system as the OS component for each container. Segregation of containers and resource allocation is simply managed by the container engine, so that each container is allocated resources. This can be done dynamically or be static, depending on the use-case.

3.2 Developments in containerisation technology

3.2.1 Docker

An ever more popular implementation of container technology is a software known as Docker. This software is primarily used by (and aimed at) developers, who can use the platform to quickly create and deploy applications for testing. The use of containers is supposed to make managing these applications much easier, and ensures compatibility along the whole development process. Datanyze (a technology market usage group) report that Docker is currently the second most used Containerisation platform, with 25.34% market share in Containerisation [Datanyze, n.d.].

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 daemon 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].

3.2.2 How Docker works

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 liscense, 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 kernal 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 kernal feature known as namespaces [Docker, 2021, Section: Namespaces]. When new containers are created, a set of namespaces are created sepcifically for that container. This means that programs that might otherwise be considered by the kernal to be entirely sperate, 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 kernal together, and also process a number of actions as one whole unit, that otherwise would have been considered seperate tasks to the operating system. This is key to the function of containers, as it allows each container to process tasks as sperate entities, but make use of the same kernal and operating system across all containers.

Control Groups

Control groups is another feature of the linux kernal used by Docker. Control groups allow hardware resources to be segrated 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. This is similar to how a hypervisor would segregate resources, but instead this is managed entirely by the kernal.

Chapter 4

System design and definition

4.1 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 virtual machine 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 4.3 (Benchmarking).

4.2 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 topology for both the Docker system and the Virtual Machine system. LAMP is the acronym for Linux, Apache, MySQL and PHP. 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 Ubunutu Server as it is headless which 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 depolyment, 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.

4.3 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.

4.3.1 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).

4.3.2 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 packetz [Ramaswamy et al., 2004]. This is usually affected by the CPU performance, which is linked to the compute performance as discussed previously.
- Queing 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, as this delay is most usually 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 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 mutliple packets. This does tie in with Computing performance.
- 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, however, this would be down to the individual ways that each of the soltutions 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.

4.3.3 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 reccomendation 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 infrasturcuture. For example, in subsection 3.2.1 (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 pssible 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.

4.4 Requirements List: Infrastructure

To create a realistic web infrastructure, a number of services will need to be implemented across

4.4.1 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, as 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.

4.4.2 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 4.4.1. Secondary DNS servers will be configured to interact with clients, whilst the primary and authoritative server will be hidden from clients. 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).

BIND

As the machines running the network infrastructure will be using Linux, the best fit for DNS architecture will be the use of 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.

4.4.3 DHCP

The network will also require a DHCP server in order to allocate IP addresses to clients.

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.

4.4.4 MySQL

As discussed in 4.2, a MySQL 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 over to containers over virtual machines. MySQL has its own Ubuntu Server Package, making the server portion of the installation easy.

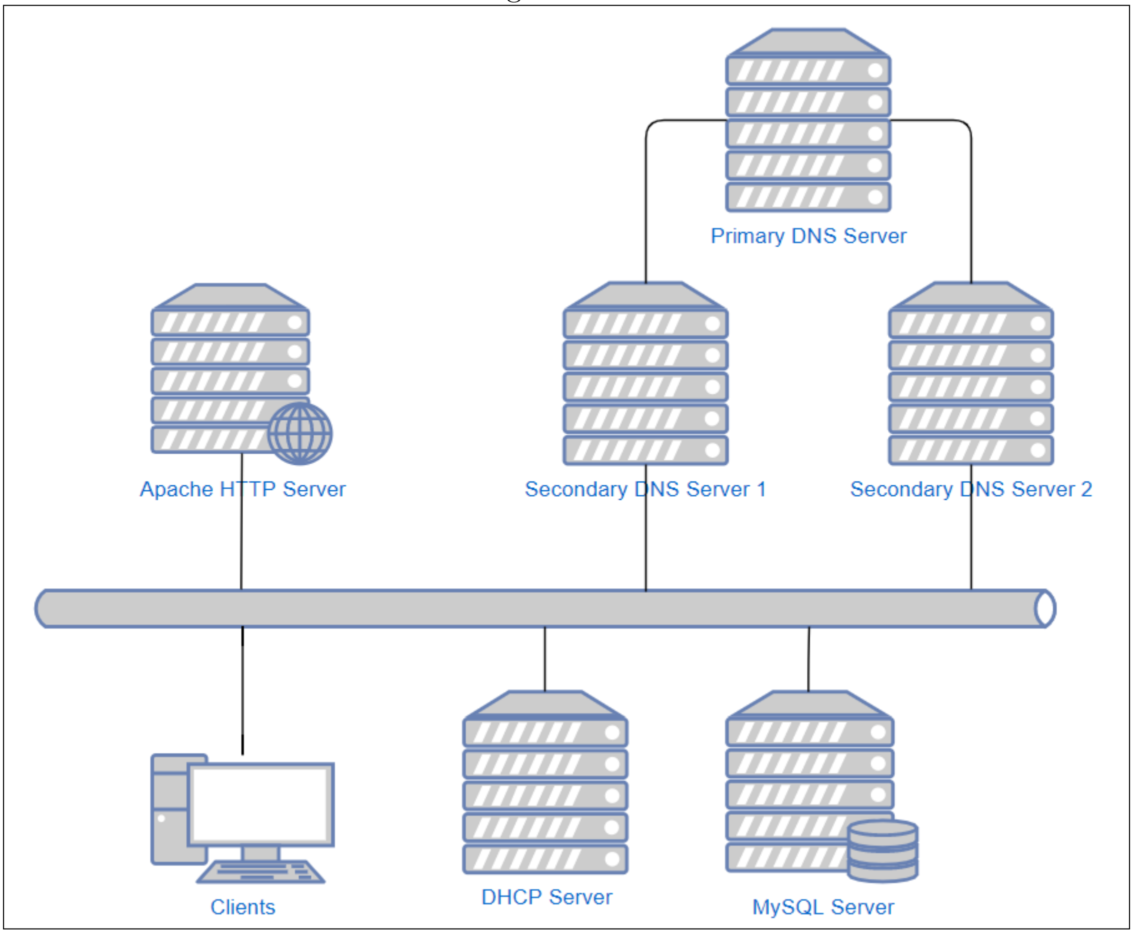
4.4.5 Clients

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 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 so that it can interact with the network setup for testing. The same system resources will be allocated to this client for both testing environments, so whilst there will be an impact on the performance of the machine doing the testing, it should be the same impact across both systems, and shouldn't change the legitimacy of the output.

4.4.6 Topology Diagram

Figure 4.1 is a mock-up of what the final topology should look like logically. This will be subject to change throughout the synthesis section of this report, as findings are made and areas are adapted.

Figure 4.1:



This topology should be replicated in the same way across both systems in order to ensure that the results are measuring exactly what is supposed to be measured.

The difference in setup here is that Docker containers are run from the command line on the host machine whereas VMware will be using the VMware workstation pro environment to virtualise a new operating system for each instance.

Chapter 5

How will the system be measured

5.1 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 elements of the system are measured, I will do four tests as follows.

5.1.1 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.

5.1.2 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 4.4.4, 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.

5.1.3 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].

5.1.4 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 4.4.1, 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 4.3.1), 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 6

System creation and setup

6.1 The host machine

6.1.1 Hardware

It was decided as a better test of Docker's ability to reduce strain on a system, that an older machine with limited resources would be used rather than a more modern and capable machine. This was to better represent the target group of this research; that being SMEs that may have aging hardware, attempting to get the most out of what they currently have available.

The test machine has

6.1.2 Operating System

6.2 Servers

6.2.1 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.

6.2.2 Software

The servers were created to use the software and topology laid out in the requirements within section 4.4.

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.

However, for VMware this was done once each server was installed and booted. On 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 **??** 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 correct folders.

6.3 Client

6.3.1 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. 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.

6.3.2 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 5.1. 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).

6.4 The Network

For both tests, VMware Workstation's NAT networking mode was used [VMware, 2021a]. 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, 2021b] to disable the built in DHCP server, so that it didn't conflict with the DHCP server created for the testing (see subsection 4.4.3).

Setting up VMware Virtual Machines to use the NAT (VMnet8) network is straight-forward. 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, 2021]. 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, 2021c], which is designed to allow the host machine to communicate with virtual machines over a virtualised IP interface that 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 6.3.1.

Table 6.1 shows the IP addressing and hostname scheme that was used in both tests.

Table 6.1:

	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'.

Chapter 7

Methods

Chapter 8

Testing & Benchmarks

Introductory explanations for each of these tests and benchmarks are included in the requirements in section 5.1.

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 specifies that is is in-fact running iPerf3 as a client, and tells the server which format to record throughput with. For this testing, the format used was 'M', meaning Megabytes [Dugan et al., 2021b].

The test runs for ten seconds, taking measurements of total throughput in a 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.

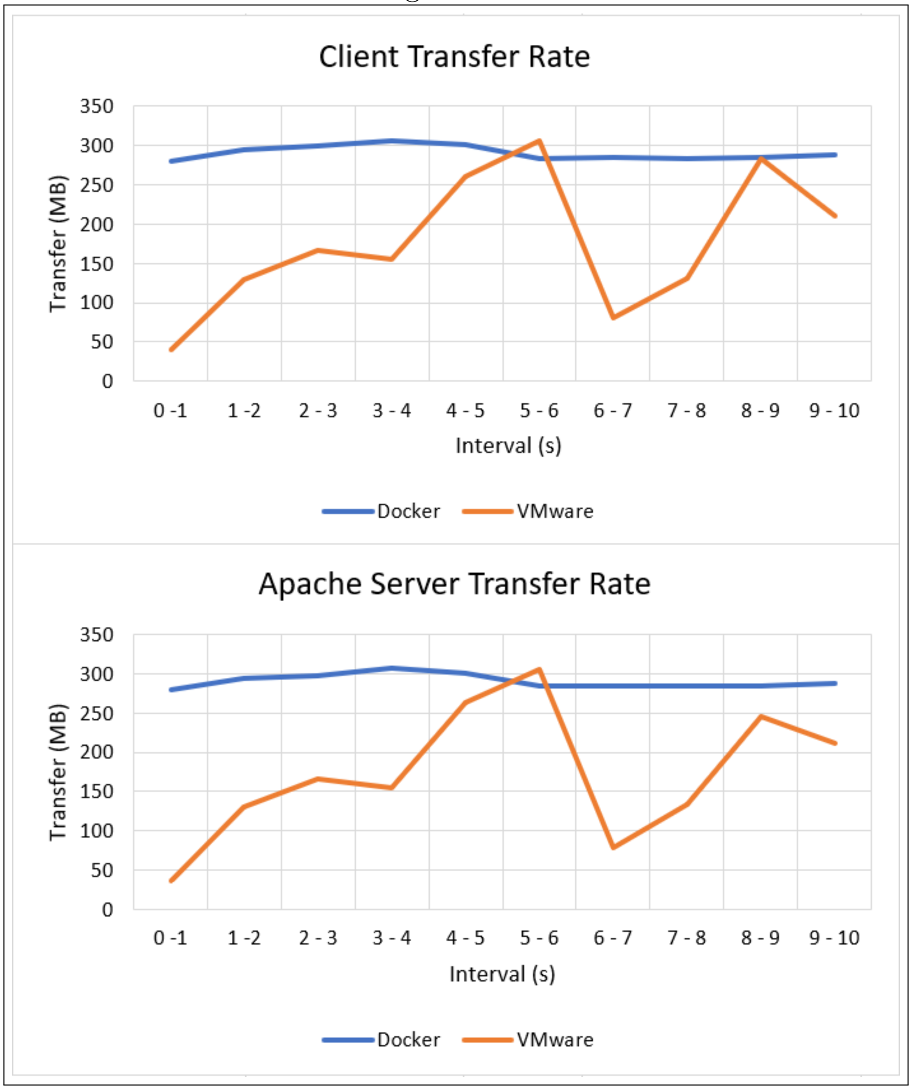
Table 8.1: Table showing the average transfer rate in a 10 second period

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:



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 section 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 of 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 ant 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 500000.
- 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[**?**], 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 different users or programs at the same time.
- '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:

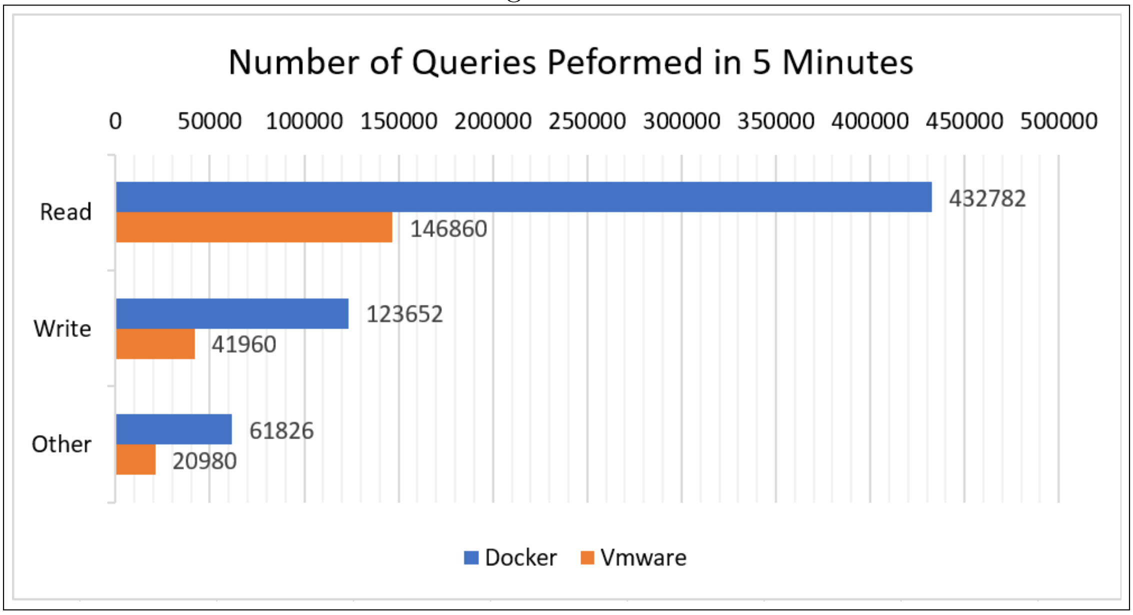
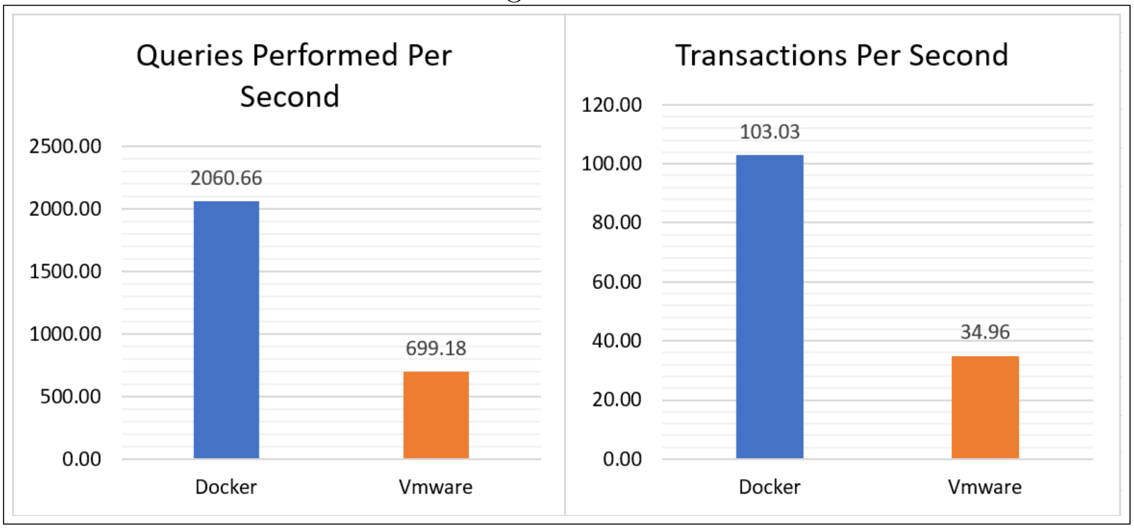


Figure 8.3 shows both the total queries per second, and the total transactions per second for both systems. A query is defined as being a single statement such as SELECT, INSERT or UPDATE[Oracle,

2021a]; whereas a transaction is a combination of multiple of these statements within one command[Oracle, 2021b], typically in order to manipulate multiple rows of data.

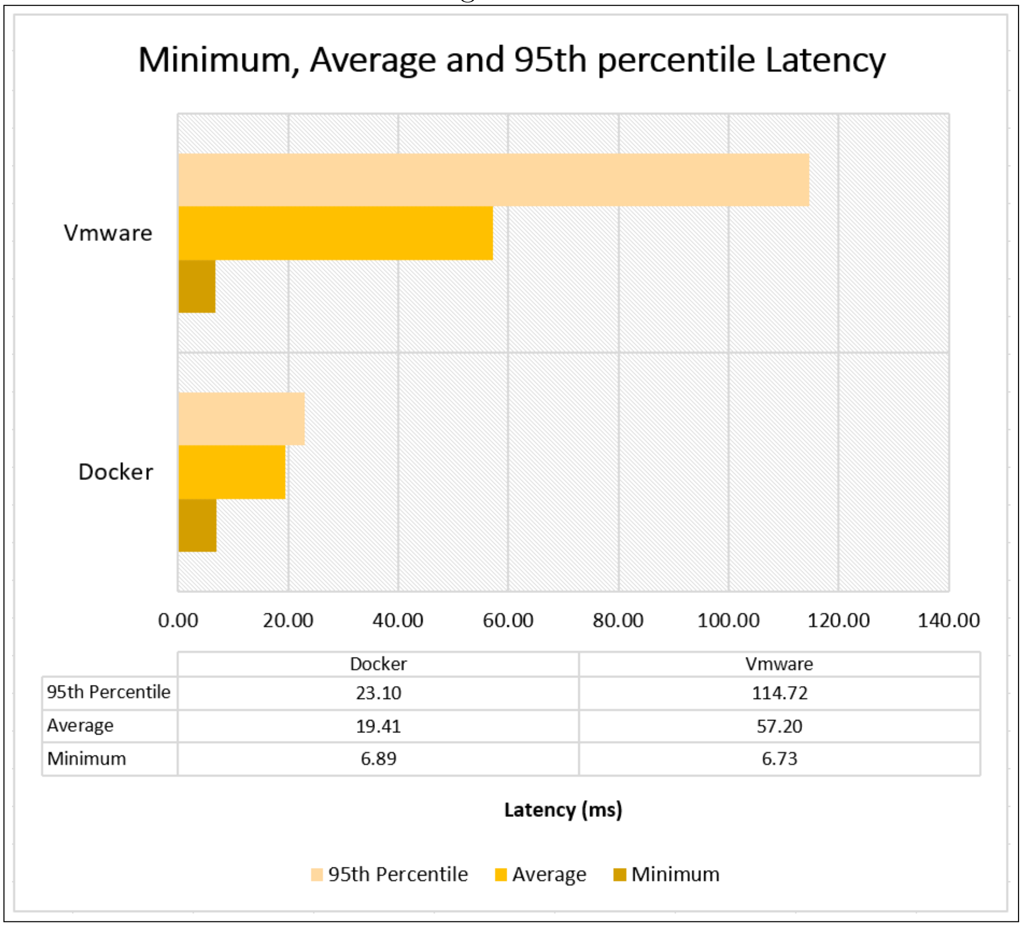
This further shows just how much faster the Docker system was at data manipulation.

Figure 8.3:



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 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 [**?**]).

Figure 8.4:



	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.

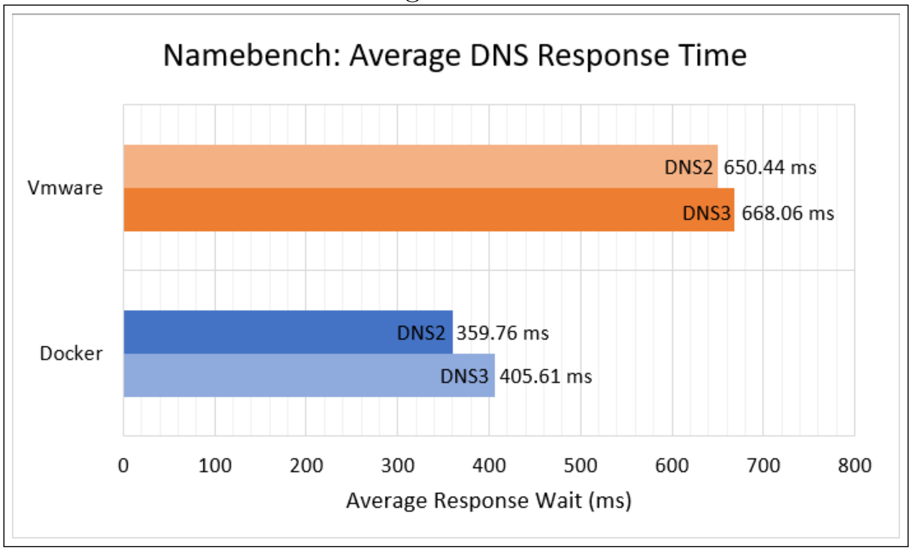
Namebench works by going through a list (specified by the user) of websites and recording the latency for the DNS server to perform the query[Stromberg, 2010]. Before the test commenced, the caches on all of the DNS servers in our infrastructure were wiped to ensure a fair test between the two systems.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

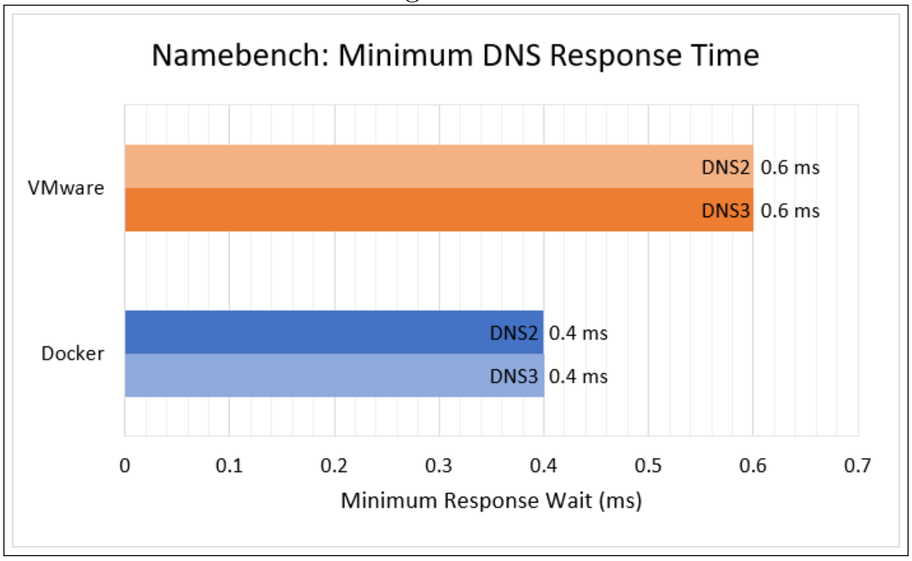
Figure 8.5 shows a graph of the average DNS response times. We can see from this graph that Docker has smaller average DNS response when compared to VMware. An interesting, but unrelated behaviour here is that DNS2 in both tests out performed DNS3; this is unexpected given that both servers share exactly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

Figure 8.5:



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:



No maximum latency is given in a graph as in both tests, certain requests hit their 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. (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 agreements.

8.4 Test 4 - Apache JMeter and Performance measurements using 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[Apache Software Foundation, 2020b], in which a number of different parameters can be set to create a test workload 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, 3.1 Thread Group]. The time between starting one thread and starting the next is equal to $\frac{Time(s)}{ThreadCount}$ [Apache Software Foundation, 2020b, 3.1 Thread Group]. 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, 3.2.1 Samplers] 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, 3.6 Configuration Elements]. 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, 3.3 Listeners]. As we wanted to be able to compile the results ourselves, we used the 'view results in table' listener[Apache Software Foundation, 2020a, 18.3 Listeners, View Results in Table], which allows us to export the results to a CSV file for easy data manipulation later.

8.4.2 Results

JMeter

Netdata

Whilst CPU and RAM usage are self explanatory, 'load' may seem more vague. Load, as measured and defined by Netdata is

Part III

Evaluation

Part IV

Conclusion & Recommendations

Bibliography

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

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

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/.

A. 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.

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.

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

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 characteristics, 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.

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

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

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 Kaustubh 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.

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.

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

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

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

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

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`.

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.

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.

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

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. Chapter 7 examples of common queries, Apr 2021a. 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 2021b. URL https://dev.mysql.com/doc/refman/8.0/en/commit.html.

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.

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`.

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

Ubuntu Documentation Team. Ubuntu server guide, Mar 2021.

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

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

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

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 environments, 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

A.9 Ethics Form

If you scan the Ethics form on one of the multifunction printers, you can get a pdf copy. This can then be included with the LaTeX command

```
kincludegraphicsn+nbethics.pdfn+nb
```

Assuming of course you have saved the form as `ethics.pdf`

A.10 Risk Assessment Form

Likewise you can scan and include the Risk Assessment Form

```
kincludegraphicsn+nbriskassesment.pdfn+nb
```