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)
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.

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.

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 subsection: 2.1.3).

This is an important definition to clarify, as often times container-

isation (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, 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, 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, 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, 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, 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, Section: Docker registries].

3.2.2 How Docker works

Docker is written using the 'Go Programming language' [Docker, 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, 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 subsection 4.1.2 (Benchmarking).

4.1.1 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 sys-

tem. 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 I personally familiar with Ubuntu as an operating system. Using Ubuntu is also good as an example, as it is a widely available and utilised operating system among those that run linux servers. 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 I come to implement my own Docker setup, 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.

4.1.2 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 base compute performance, and of network performance.

Base Computing Performance

Base 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).

Network Performance

Further to the Base 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. Network delay can be split into a number of different types of delay:

- Processing Delay: The amount of time it takes for a node (router, server, client, etc) to process the header of a packet. This can be affected by
- Queing Delay
- Transmission Delay: The amount of time it takes for a node to 'push' packets (bit by bit) onto the line [che].
- Propogation Delay

4.1.3 DNS

4.1.4 Webserver

4.1.5 Intranet

Chapter 5

How will the system be measured

Part II

Synthesis

Chapter 6

Some TeXnical Details

6.1 Structure of the file set

The Terms of Reference and Dissertation are split into several files, to ease editing, and exploit some of LaTeX's capabilities. The structure is relatively *flat* with little hierarchy of directories, directories and sub-directories can be added to simplify some of the structure as the project grows.

The principle files are:

`Makefile` I use `make` still as my build driver. Any automated build system can work with LaTeX files, it just needs to know that PDFs are build from `.tex` files.

`TermsOfReference.tex` This is the main file for creating the *Terms of Reference*. It pulls in the `tor.tex` file, the Gantt chart from `gantt.tex`, and the ethics and risk assessment forms from scanned PDFs. The document is typeset as an `article`.

`Dissertation.tex` Is the main file for creating the dissertation. This pulls in a number of other files as required. It is typeset as a `book`. The sectioning starts as `\chapter` and has `\section` within. This way the terms of reference can be included as a chapter in the appendix. The chapters are each included using `\include`, this allows the chapters to be written as separate files. The difference between

`\include` and `\input` is that `\include` forces a new right-hand page to start (good for starting chapters).

6.1.1 Subsidiary files

`tor.tex` This file is the main *Terms-of-Reference* file. By using `\input` in the `TermsOfReference.tex` document, this gets included and typeset.

`gantt.tex` The Gantt chart is a little complex, so gets put into a separate file, see section 6.1.2.

6.1.2 The Gantt Chart

The Gantt chart in the Terms-of-Reference is drawn with another latex package. It uses an `input` command to pull it into the tor (and dissertation).

Most of the file `Gantt.tex` is the setup of the Gant chart, in order to make it fit in one page. The bottom section is where you can define the tasks. Each bar has a title, a start date, and an end date. Milestones have a title and a date. The `ms` option can be set to left or right to control which side of the milestone mark the text label.

Part III

Evaluation

Bibliography

The electrical engineering handbook. Elsevier Academic Press.

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`.

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

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.

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

Docker. Docker overview. 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.

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.

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

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.

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.

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

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`.

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`.

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.

Part IV

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

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

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