INTR 8016 Project Research Report
Building a Hosting Service for Websites

David Monaghan
B.Sc.(Hons.) in IT Management
Department of Computing

Student ID: R00103763
Supervisor: Meabh O'Connor
Second Reader: Noreen Gubbins
Date: 10th January 2016

Table of Contents

1 - Executive Summary

The primary goal of this project is to create a service that will allow a customer to create, quickly and easily, a website without having to worry about any of the underlying technical issues involved in such an undertaking such as installing or configuring the servers required. The service will be designed with scalability and security in mind.

The proposed method for achieving this task is to create a virtual machine that will contain an Ubuntu server along with an Apache web server and Website Content Management System that is managed by a database. The customer will be able to launch this with the choice of a few options, then a click a button and the web-hosting service will do the rest.

The authors primary concern, while conducting research for the project, was to ensure that he gained an excellent understanding of the design principles required for a project of this nature. To that end, research focused on three specific areas:

1. **The industry consensus on the requirements for a web based application**

   There is much agreement to be found in the literature, but no universal consensus found. The author combined the opinions of many authors to produce a list of eight none-functional requirements: Availability, performance, reliability, manageability, cost, redundancy, scalability, and security.

2. **Best architecture design paradigm for a project of this nature**

   Two design strategies are explored, a vertical and horizontal scaling architecture. The pros and cons of each are reviewed. By consensus, in the literature, the view is that a horizontally scaled application is most robust and easier to scale in the long term.

3. **The best virtualisation solution for a project of this nature**

   A number of virtualisation solutions were reviewed, and they fall into two basic categories, Hypervisors with full virtualisation and Containers with partial virtualisation. By most metrics used in benchmarking, the two categories are nearly identical regarding performance except for one metric. Concerning virtual machine density per server, Containers are the clear winner with up to fourteen times the density as compared to a hypervisor.

With research completed we move to defining the requirements of the project: functional, none-functional and project constraints. The aim of the current project is not to build the entire system but instead to build a good solid foundation from which the project can continue in later development cycles. To that end, the requirements specify only the core of the system in this round of development. Other requirements are noted for later development. The coding system for a requirement is quite simple and easy to understand. Anything in Green is a requirement for this project. In total, there are 45 requirements for this project, divided into three categories: system, customer and administrator requirements. Approximately 24 of the requirements overlap categories.

The next chapter outlines possible technologies that could be used in the implementation stage of the project, and each section concludes with the authors recommendation. Initial designs are included for the website, database schema, network architecture and application architecture. Technologies chosen for the implementation phase are:

- **Main Website:** HTML/ CSS/ PHP
- **Database:** MySQL
- **Programming Languages:** Go / PHP
- **Virtualisation:** LXC Containers

The penultimate chapter outlines the initial project schedule and plan. A breakdown of how the application will be developed and when each component will be developed. The project has been budgeted with 14 Hours/ Week for 12 Weeks from 1st of February until the 23rd of April. A Gantt chart is provided as a visual breakdown of the work schedule.

The final chapter concludes with some lessons learned during the project. The author outlines what the project has been trying to achieve and gives a picture of what to expect when the project is demonstrated at the end of the implementation stage. Finally we look at possible at what future development will look like and how the project has sought to make such development as easy as possible.

2 - Introduction

2.1 - About The Project

Setting up, configuring and maintaining a server(s) is a tedious and painstaking task. A single mistake and one will inevitably have to start from scratch. The project's goal is to take away some of that tedium by automating many of these initial tasks and allowing the client to get to the important part of building a website, namely designing the site and creating its content.

The author set himself the task of creating a service that can:

1. Create an Ubuntu 14.04 virtual machine
2. Install and configure an Apache web server
3. Install and configure a Wordpress CMS
4. Install and configure a MySQL database for Wordpress site
5. A web interface to initiate and control the virtual machine
6. Finally, to complete these tasks as quickly as possible

This service should, once this project is completed, be easy to continue expanding if there is another development cycle, where ultimately the goal in mind is to create the service in full. The author aims to use best industry practices to ensure that another individual could take from where the project finishes.

The author's goals are twofold. The first objective is to investigate some new technologies and apply them to the projects solution. The second objective is to bring together many of the subjects and disciplines that the author has studied while in CIT and apply them to the new technologies investigated throughout the project.

2.2 - About The Author

The author is a mature student having returned to College after many years in the workforce and looking for a new challenge. While the author has enjoyed his time in CIT, he is very much looking forward to finishing studying and moving onto the next challenge.

2.3 - Declaration of Original Work

I hereby certify that this material which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent, that such work has been cited and acknowledged within the text of my work. I understand that my project documentation may be stored in the library at CIT, published on Blackboard and may be referenced by others in the future.

Date: _____ **Signature** _____

3 - Literature Review

3.1 - Overview

In conducting research for this project, the author explored a number of different areas that are relevant to the intended goal of creating web-hosting service. The research conducted was often broader than the intended scope of the project and for completeness has been included in this chapter to better give the reader an indication of the complexity involved in creating a website hosting service. The specific areas that the project will cover is laid out in detail in the following chapters. The sources consulted for this research where academic research papers published by the IEEE, white papers published by leaders in the industry and articles published by experts in the industry.

3.2 - Architecture of a Web-Based Application or Service

3.2.1 - Web-Based Application Architecture Considerations

When building a modern web-based application/ service, it is best to consider the applications architecture prior to the applications design. Chief among these considerations is how will the application deal with sudden and unforecasted changes in network traffic, whether that be a sudden increase or a sharp decrease. The ability to adapt to these changes, be they capacity or availability changes, determines the ultimate success of the application being developed.

Before planning the architecture of a web-based service, (1) (2) (3) many factors first need to be considered:

1. **Availability:** This is the uptime of a service. Consider a situation where a web-site would normally deal with traffic hits in the hundreds per hour but due to significant exposure on a social media site such as Reddit, it suddenly experiences a jump in traffic hits to hundreds of thousands per hour. Is this site designed in such a way as to handle the increased load? Is the underlying system architecture capable of increasing its resources to keep up with the growing demand, such as using AWS (Amazon Web Services) configured to scale up/ down as demand requires?
2. **Performance:** This is an end-user metric and is subjective to each user. It is widely considered essential that a good web-site should not have a user waiting overly-long for a service to complete. Performance is something that changes depending on the service e.g. the homepage and content pages of a web-site should load with minimal delay whereas a login server or payment server may require a few seconds or minutes to process the request. The performance of a web-service is directly related to the availability of that same service, a poorly performing service is an unavailable service in the mind of the customer.

3. **Reliability:** This is the accuracy of the data. Each time a computation is run on a set of data, it should return the same result unless that dataset has changed, in which case it should return an accurate value based on the new data. Consider, if an SQL query is run on the sales table, requesting the total sales for a given period, it should return the same value each time, unless new sales data for that period has been added to the dataset, in which case it should return the updated result. Data reliability becomes especially important when a web-based application is distributed over many servers and a wide geographic area. To ensure data reliability it is essential to ensure that data:
    - **Real Time Data-replication:** Ensures that all data stores contain the same data-sets. This is especially important to ensure consistency across geographic sites/ distributed servers and to prevent data inconsistency in the case where one server goes down, and the other servers are left without up to date data.
    - **Regular Automated Data-backup:** Ensures that the service can be recovered in the case of the failure of one or more elements providing the service. A backup policy should allow any service to be rolled back to an earlier state before an error occurred. Data backups should also be copied off-site to a long-term medium to ensure the failure of data storage on-site does not compromise the web service.
4. **Manageability:** Refers to how easy or difficult the service is to run. Issues to consider would be tool-sets used to manage the service e.g. configuration management tools like Puppet, automation of the service e.g. should scaling of service be done manually or by the system. An easily manageable service where everything is automated may come at the price full system control and calibration for the administrator. Manageability should also include items such as logging, reporting, disaster recoverability, ease of troubleshooting and the number of staff required to properly run the service.
5. **Cost:** TCO (Total Cost of Ownership). TCO refers to all the associated costs of developing, running and supporting a service these include but are not limited to hardware costs (networking/ data centre costs), software development (development, debugging, upgrading), operations (day to day operation of the service, public cloud usage costs) and staff costs.
6. **Redundancy:** This refers to the practice of designing an application or a service in such a way that if a single component where to fail, there would a second or third component that could take its place. In a scenario where a site uses a database to deliver dynamic content, the failure of that database would be the failure of the site as well. In this case it would be best to have a number of other databases, that have synchronised their data using real-time data replication, to take over from the database that has failed. This strategy would prevent the site from failing as well. A redundancy strategy can be used in concert with load-balancing strategy, one which aims to maximise performance across the network.

7. **Scalability:** This is the ability of a service to change its capacity to deal with current/ forecasted demand. Scalability can refer to many/ all elements of a system e.g. ti may refer to the ability of a data centre to create new web servers in response to increased network traffic or the ability of that data centre to increase data storage capability. Concerning scalability, there are two basic strategies:
    - **Vertical Scaling:** This is an architecture where one changes the capacity of the underlying system to deal with increased/ decreased loads e.g. upgrading a server's CPU, RAM or hard-drive to deal with bigger datasets and increased computation.
    - **Horizontal Scaling:** This architecture is one where the number instances of a service are increased/ decreased to match demand e.g. increasing the number of web servers available to a website to meet spikes in demand i.e. an e-commerce website may experience a spike at Christmas and during sales but may want to reduce capacity at other times of the year.
8. **Security:** It is essential that the security and privacy of the customers data are considered in all web application architectures for both practical and ethical reasons. First among these reasons is that a data breach can have devastating consequences for a business. In the EU, failure to comply with DP regulations can cost fines worth hundreds of thousands of Euro and from next year, possibly up to 4% of a companies revenue. Aside from regulatory sanctions, there is the loss of reputation and, perhaps, an even more devastating loss of business. From an ethical standpoint, it is essential that compromises of customers security are minimised to prevent such things as identity theft and IP (Intellectual Property) theft. To ensure customer and application security the following need to be considered:
    - **Firewalls:** Provides protection against unauthorised access to the application/ service and of the components that make up that service. Firewalls come in some categories:
        - **IDS:** (Intrusion Detection System) are designed to detect and alert when an unauthorised access has taken place
        - **IPS:** (Intrusion Prevention System) are designed to actively prevent an unauthorised access and often used in conjunction with an IDS
        - **Application Firewall:** Controls the inputs /outputs, and access, to an application.
    - **Encryption:** This is the process of encoding information in such way that it cannot be read unless it is decoded using the correct key(s). Concerning a web Application or service, there are 3 areas where encryption is likely to be used:
        - **HTTPS/ TLS:** Refers to encryption of traffic on the Transport Layer of the TCP/ IP model, specifically this would refer to client/ server communication both between the a customer & the web application/ service and between

components of the service such as the web server & the database server.

- **Stored Data Encryption:** Refers to data we store on behalf of our customers such as email, credit card details, and other customer details. This data would be encrypted on a file by file basis using a password or certificate-based encryption schemes.
- **Application/ Component/ File-system/ Full Disk Encryption:** As an extra security enhancement one could also encrypt an entire file-system or the hard-disk. The advantage comes where an attacker has compromised one component of the application/ system the damage would be limited due to them lacking encryption keys for the rest of the system.

- **User Authentication:** In this instance, refers to Human actors interfacing with the system and is categorised by 2 types of users:
  - **Application Administrators:** This user is one who has access, to one degree or another, to internal workings of the web application/ service and would access the application physically via the terminal or if accessing remotely, it is essential to ensure that a secure encrypted protocol such as SSH or a VPN is used.
  - **Application Customers:** This user will access the service via the Internet. Typically the customer will register for one or more accounts using by creating a user-name and password. The customer would then use these details to log in. To prevent data breaches, it is essential that this is provided using the correct encryption policies such as using HTTPS and ensuring passwords files are correctly encrypted and salted. Further to this, authentication cookies left on the customer's client should be sufficiently encrypted to prevent brute force cracking, in the event a customers cookie is stolen from their device.
- **Server Lock-down:** This refers to locking down who can access the server, how they access the server and what they can do once they do access the server. Many strategies can be taken in this regard such using public key encryption for user access or the use of firewalls. At the minimum every server should be configured in the following way:
  - **Close Unused Ports:** Computers communicate with each other using port numbers such port 22 for SSH servers. If a port on a server is open, then it means it will accept communication on the protocol associated with that port. It is considered best practice to close all unused ports on a server.

- **Permissions and Privileges:** For UNIX-like servers it is essential that the correct file permissions and privileges are used throughout the file-system. A UNIX system is a multi-user system that organises security by file ownership & permissions, and the each user will belong to one or more groups. Each member of a group has the owns the same files as all other members of that group and has the same permissions on those files.

  Not all users are necessarily people but are more often other Linux applications, programs or services. From the perspective of the web server where it is being accessed from another computer setting the correct ownership and permissions on the installed files and programs is essential to ensure that an attacker cannot get access to the wrong part of the system.

  File Ownership

  - **User:** Refers to the user-id of the user that is primarily in charge of that file
  - **Group:** Refers to the group that the User belongs to
  - **Other:** Refers any other User that is not a member of the group

  Permissions

  - **Read:** Users with Read permission can only view the file and cannot make any changes
  - **Write:** Users with Write permission can view the file and make changes to it as well
  - **Execute:** Users with Execute permissions can run programs

The issues outlined above are not an exhaustive list of all matters that require attention when developing a web-based application. Many of the issues described above are complementary, such as availability affecting the end-user perception of performance and others are possibly in contention with each other, such as a tightly secured web-based application may make manageability more difficult which in turn could affect the systems availability. Each of these elements will need to be considered concerning the other and trade-offs of one consideration versus another are inevitable. The key to good design is to account for as many of these issues in advance and to build a system that is easily updated as future issues, not initially considered, arise.

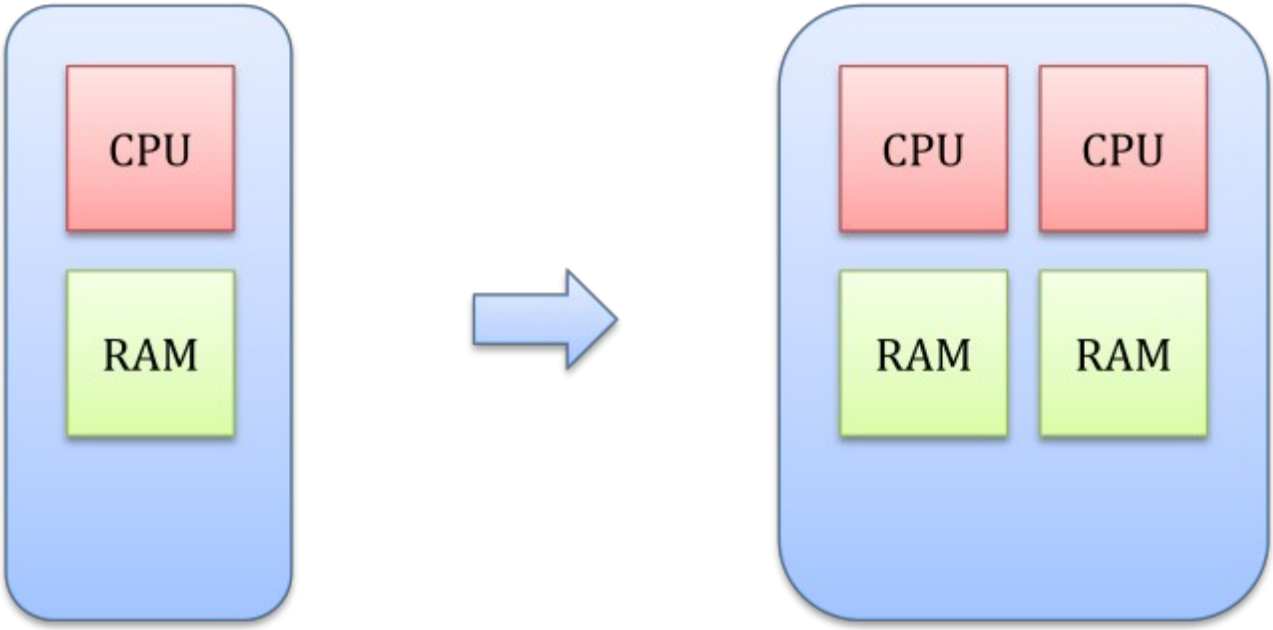
3.2.2 - Scalability: Vertical Scaling and Horizontal Scaling Strategies

Scalability is the ability of an application to grow its resources in response to increasing demand and conversely to reduce its resources in response to decreasing demand. Scalability is an important property of any web-based application as this ensures that resources, which may be scarce or expensive, are utilised to greatest efficiency. There are two basic strategies:

3.2.2.1 - Vertical Scaling

Entails adding more resources to a component of the application, allowing it to cope with increased demand e.g. a payment transaction database that experiences more database writes than database reads would need to increase its CPU, RAM and other components to deal with increased demand.

Figure 1: *Vertical scaling (4)*



3.2.2.1.1 - Advantages

- (3) Lower administration costs as a well designed vertically scaled web architecture will typically have fewer components than a horizontally scaled architecture e.g. a single web-server with sufficient resources like memory or network speed is easier to administer than multiple web-servers spread horizontally.
- Easier to implement.
- Lower cooling and power costs in the data centre due to fewer components spread across multiple physical devices.

3.2.2.1.2 - Disadvantages

- Requires that the web-application be correctly designed from the offset.
- Failure of a critical node can bring down entire system.
- Requires correctly calibrated resource benchmarks to ensure the system has enough resources to cope with demand. Correctly calibrating the system not be something that can be easily determined before building out the system. If it is not done properly, then one may end up with an underpowered system that is not capable of providing required availability.
- All the infrastructure costs are paid up front and may initially be excessive, compared to what is immediately required. These costs could be difficult to justify, especially where a service is new, and it is uncertain what its capacity and availability requirements will be.
- Each component of the web-application will have both a theoretical and practical upper-limit threshold at which it can operate, therefore, it is not possible to indefinitely increase capacity in a horizontal direction e.g. a website may be increasing in popularity to a point where a single data centre is unable to handle internet traffic due to local limits on network speed, in this case, one would need to consider creating a new data centre in another geographic location.
- Expansion of the capacity of vertically scaled system needs to be done manually and physically e.g. increasing hard-drive capacity, which will reduce the ability of the application to respond immediately to increased demand.

3.2.2.1.3 - Developing a Vertical Scaling Architecture

Another take on this strategy is to plan in advance the resource requirements for the web service and plan regular upgrades as the system needs to grow. This is the strategy implemented by Stackoverflow (5). Stackoverflow released figures detailing the load on one of their data centres. The figures below outline a 24-hour window on a normal weekday, 12th November 2013:

- 148,084,883 HTTP requests to our load balancer
- 36,095,312 of those were page loads
- 833,992,982,627 bytes (776 GB) of HTTP traffic sent
- 286,574,644,032 bytes (267 GB) total received
- 1,125,992,557,312 bytes (1,048 GB) total sent
- 334,572,103 SQL Queries (from HTTP requests alone)
- 412,865,051 Redis hits
- 3,603,418 Tag Engine requests
- 558,224,585 ms (155 hours) spent running SQL queries
- 99,346,916 ms (27 hours) spent on redis hits
- 132,384,059 ms (36 hours) spent on Tag Engine requests
- 2,728,177,045 ms (757 hours) spent processing in ASP.Net

To deal with this load StackOverflow, excluding extra equipment required for redundancy, use the following equipment in their data centre:

- 2 MS SQL Servers
  - 384 GB of memory with 1.8TB of SSD storage
- 2 IIS Web Servers
  - 32 GB and 2x quad core and 300 GB of SSD storage
  - 4x 1 Gb of network bandwidth
- 1 Redis Servers
  - 96 GB of RAM
- 1 Tag Engine servers
- 1 elasticsearch servers
  - 196 GB of RAM
- 1 Load balancers (HAProxy)
- 1 Network (a Nexus 5596 + Fabric Extenders)
  - 10 Gb of bandwidth on each port
- 1 Cisco 5525-X ASAs
- 1 Cisco 3945 Routers

The key to StackOverflows strategy is to ensure that the resources available greatly outweigh any possible demand while still allowing themselves room to grow in future. Another key element of this strategy is to make sure that the site was designed correctly in the first instance and to ensure excellent quality code that does not unnecessarily use up system resources.

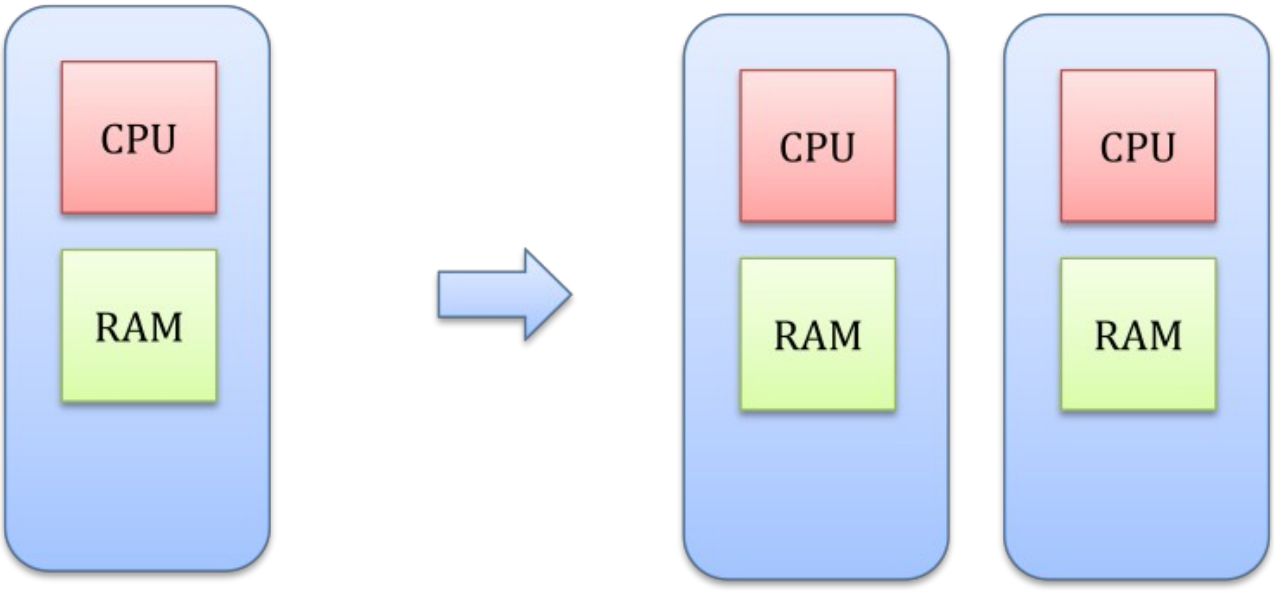
This strategy has a potential pitfall. To this point, StackOverflow has been able to predict and stay ahead of their Internet growth. In a situation where they miscalculate demand, or their growth exceeds even their most optimistic predictions, their architecture could soon be overwhelmed. This would leave them with many questions:

1. How long would it take them to upgrade their system?
2. How long can they afford to take the site down while they upgrade?
3. If they can't afford to take the site down, how long would it take to scale horizontally?

3.2.2.2 - Horizontal Scaling

Entails adding more component nodes and distributing the load across the increased volume of component nodes e.g. increasing the number web-servers for a particular site, when it is experiencing increased traffic.

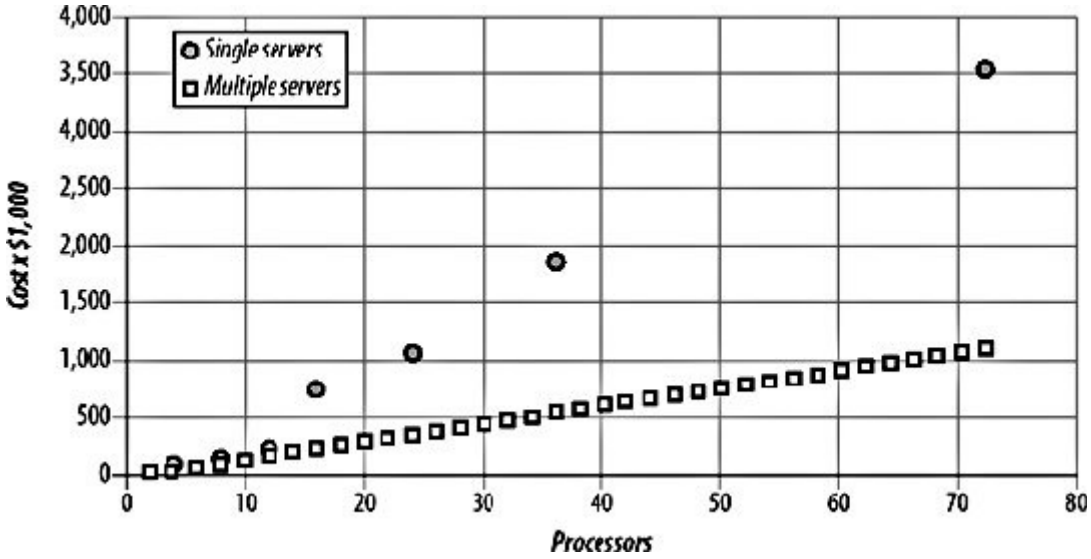
Figure 2: *Example of vertical scaling (4)*



3.2.2.2.1 - Advantages

- (3) (6) Increased redundancy as the load is spread across multiple nodes.
- No single point of failure.
- Easier to immediately respond to changes in availability demand.
- Reduced costs as compared to a vertical architecture with similar capacity as seen in figure 3

Figure 3: *Cost of Horizontal Scaling versus Vertical Scaling* ***(6)***



3.2.2.2.2 - 3.2.2.2.2 - Disadvantages

- More complicated to manage, it will require a comprehensive monitoring system to allow for both manual and automatic scaling.
- Developing distributed systems is more complex.
- Requires extensive planning and investment at the outset.
- Applications will need to be optimized to operate on a distributed architecture.
- Tasks will need to work in parallel automatically.
- Load balancing and network routing become more complex due to increased nodes performing the same function.
- Performance issues can be the result of increasing complexity in the data centre as it expands vertically. Increasingly complex distributed caching strategies will be required, such as using a CDN (Content Delivery Network) which is a worldwide network of servers used to deliver static content e.g. images, static HTML pages. CDN are intended to take the load off of the main application by delivering static content from shortest hop server.

3.2.2.2.3 - Developing a Horizontal Scaling Architecture

Horizontal scaling strategies have been put in place by many of the world's largest internet-based industries such as Google & Amazon and have been used in large-scale scientific computing environments for even longer (7). It is now easier to deploy a horizontally scaled architecture as companies like EMC/VMware now produce both the hardware and software to manage the distribution of component nodes such as servers and storage.

To develop a web application that scales vertically, the following items need to be considered: (3) (8) (6) (9) (10)

- **SOA (Service Oriented Architecture):** Refers to the practice of splitting an application into groups that deliver related services or a single service. In the e-commerce website scenario, instead of using a single server to provide content, login authentication and payment services, one could instead split each of these components into separate servers/ databases and software to provide each of these individually. The advantage of decoupling services from each other is that one can develop, maintain and scale each component without necessarily affecting the other services. SOA operates best when each service communicates with other services via a RESTful API, where each service passes only pre-defined methods along with relevant variables to each other. A further advantage of using SOA with a RESTful API is that allows one to better marry together different technology stacks.
- **Application Tiers:** Similar in concept to SOA, multi-tier or N-tier architecture is a client-server model of application development where different aspects of an application are kept physically and logically separate. There are many N-tier architectures used, but a simpler form of multi-tier architecture is the 3-tier architecture that divides an application into presentation, logic and data tiers:
    - **Presentation Tier:** This tier represents the interface through which a user would interact with the application such as web page, GUI or a CLI.
    - **Logic Tier:** This layer coordinates activity between the application layer and the data layer. This layer would also co-ordinate the application, processes commands, performs logical calculations, decisions and evaluations.
    - **Data-tier:** This tier would store and retrieve data from a database or a filesystem. The data would be passed to the logic tier for processing.
- **Stateless Application Architecture:** Refers to the policy of not storing stateful information in the presentation or logic tiers, this can include IP addresses for servers or copying files between servers. All stateful data needs to be accessed through the data layer so that each node can access this data. This practice will reduce the amount of storage space required for data, prevent the network being overloaded by files being synced between servers and will allow each node to access the same stateful data concurrently.

3.3 - Virtualisation

Virtualisation refers to using a software defined virtual environment that emulates another piece of hardware or software, which in turn allows one to run an application or service where one could not previously. A common use of virtualisation allows one to run several OS on a single piece of hardware, where previously each OS would have required its own piece of hardware. In the context of the project and use in data centres generally, virtualisation is used to gain the best efficiencies concerning resources used, reduced hardware expenditure and ease of scaling capacity horizontally.

Virtualisation comes in many categories (11):

1. **Full Virtualisation:** Simulates the required hardware entirely.
2. **Partial Virtualisation:** Provides some simulation of the required hardware. This means that a full OS cannot run in this environment but several individual applications would instead.
3. **Para-virtualisation:** Does not provide any hardware simulation instead provides a software layer that allows the guest OS to communicate with host hardware via an API. Guest applications have to be modified to run in this environment.
4. **Operating System Level Virtualization:** This method allows for multiple isolated user-space instances to share the same kernel, that allows multiple guest OS to share system resources without the need for hardware emulation.
5. **Application Level Virtualisation:** Provides encapsulation of an application or service from the OS. This encapsulation serves to isolate the application from its surrounding environment although it will operate as if interfacing directly with OS.

Each category of virtualisation uses a software layer between the host OS and the guest OS. For this report we look at two types, the first being a hypervisor (Full Virtualisation, Partial Virtualisation & Para-virtualisation) and the second being containers (Operating System Level Virtualisation & Application Level Virtualisation).

Figure 4: *A component-based comparison of three hypervisor strategies (12)*

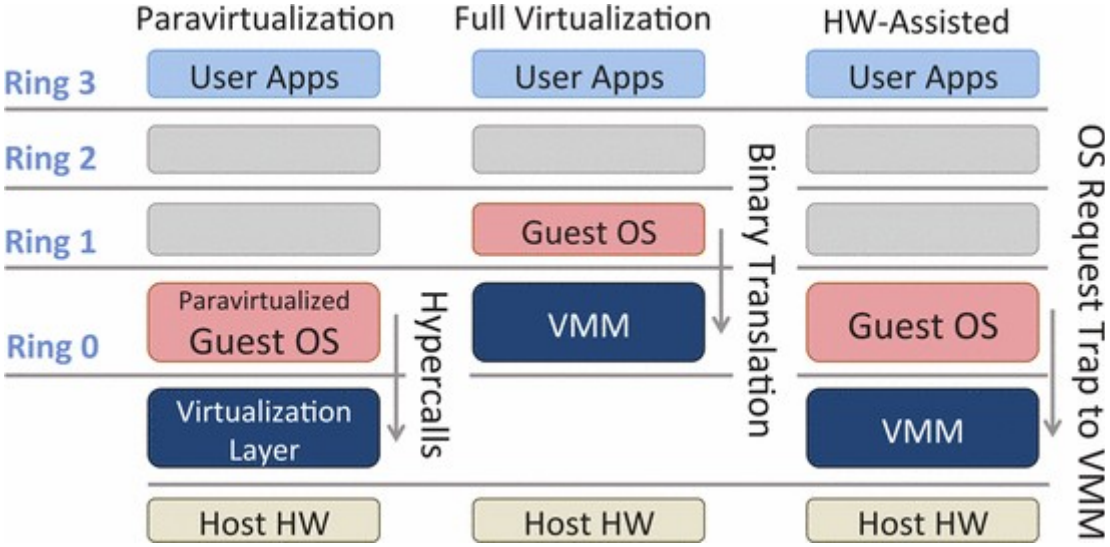
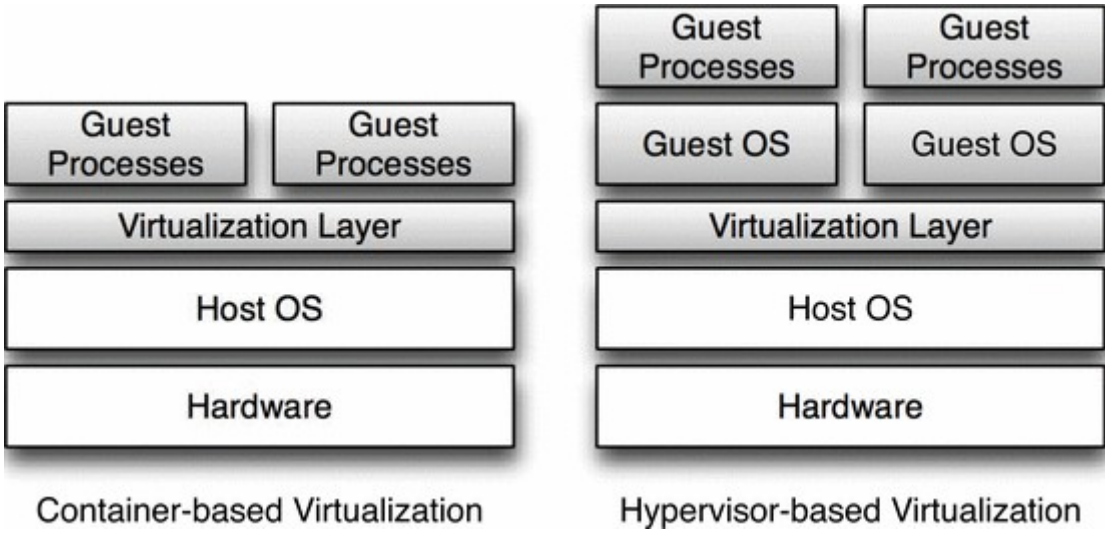


Figure 5: *High level outline of the difference between full OS virtualisation and a Conatiner (13)*



3.3.1 - Hypervisors

A hypervisor is a software layer that allows for the creation, running and management of virtual machines. These virtual machines will run full an OS. Virtualisation platforms providing full OS virtualisation for use in a data centre are provided by Hyper-V, KVM, vSphere and Xen. Many questions need to be asked when comparing these platforms

Table A: ZDNET published a comparison of some of these figures in 2012 (14)

Feature	XenServer 6.0	MS Hyper-V 2.0	MS Hyper-V 3.0	VMware Vsphere 5.0	RHEV 3.0 (KVM)
Cores/ Host*	160	64	160	160	160
RAM/ Host*	1TB	1TB	2TB	2TB	2TB
CPUs/ VM*	16	4	32	32	64
RAM/ VM*	128GB	512GB	1TB	1TB	64GB
VM Disk*	2TB	2TB	64TB	2TB	2TB
VM Live Migration	Yes	Yes	Yes	Yes	Yes

* Maximum

Table B: The following table is update on the above table for 2015: (15) (16) (17) (18)

Feature	XenServer 6.5	MS Hyper-V Server 2008 R2	MS Hyper-V Server 2012 R2	VMware Vsphere 6.0	RHEV 3.5 (KVM)
Cores/ Host*	160	320	320	480	160
RAM/ Host*	1 TB	4TB	4TB	12TB	4TB
CPUs/ VM*	32 (Linux) 16 (Windows)	64	64	128	160
RAM/ VM*	192GB	64GB	1TB	4TB	4TB
VM Disk*	2TB	64TB	64TB	62TB	Not Available
VM Live Migration	Yes	Yes	Yes	Yes	Yes
VM Density*	650 (Linux) 500 (Windows)	1024	1024	1024	Unlimited
Cost	Yes	Yes	Yes	Yes	Yes

* Maximum

The tables above show that while there are differences in maximum capacity between Hypervisor platforms they are largely on par with each other, the obvious exception being the gap between XenServer and the other platforms in 2015.

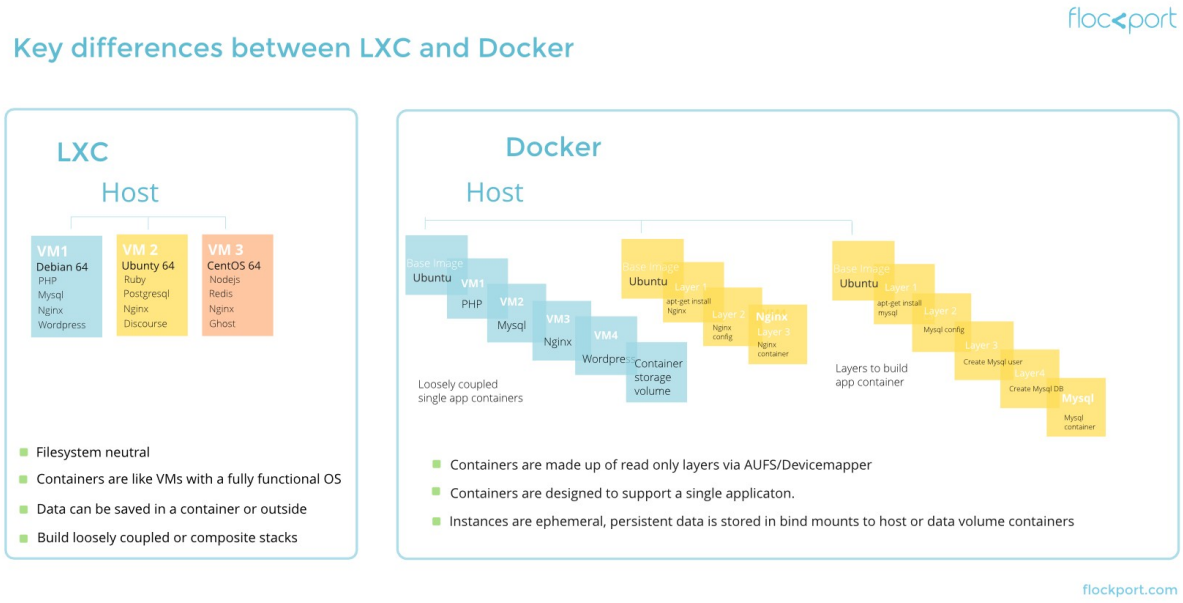
3.3.2 - Containers

A container is an operating system or application level virtualization technology, and it allows multiple guest OS to share the host OS kernel. It can do this by creating multiple user-space instances on a single OS host. Isolation of each container is provided by the use of cgroups, which manage resource allocation to processes, and AppArmour, which is a Linux security module that restricts the capabilities of each container. Two modern examples of container technology are LXC provided and Docker, which are both available as free and open-source software.

Docker and LXC are both OS level virtualisation platforms. The key difference between the two is that Docker is intended to be used as a single container for each application service and then link them together to create the application, in essence, Docker strips LXC of everything but what is exactly required for that application service. LXC, on the other hand, is intended to emulate a VM with full OS capabilities but without any hardware emulation. (19) (20)

The use case between the two is obvious. Docker is used where one has a single decoupled service that does not rely on any other service or dependency, whereas LXC would be used when one has two or more coupled services that need to be packaged together.

Figure 6: *Key differences between LXC and Docker (20)*



Runtime constraints on containers are not determined by a Hypervisor as detailed in the previous section but are instead directly tied to the host system. This means that limits on Cores/ Host, RAM/ Host and VM disk size are determined by the host OS/ host hardware limits and not by the hypervisor. Runtime constraints on containers are set with a default value but can both be changed via CLI. (21) (22) (4) This means depending on the capabilities of the host OS and/ or hardware that containers will able to match or exceed the capabilities of Hypervisor VMs in terms maximum specifications for host/ guest VMs.

VM density per host is another important question to consider. Ubuntu recently published a benchmark comparing LXC VM density as compared to KVM (Kernel Virtual Machine), which is full virtualisation technology. The key benchmarks where: (23) (24)

- LXC achieves 14.5 times greater density than KVM
- LXC launches instances 94% faster than KVM
- LXC provides 57% less latency than KVM

Figure 7: *Image Ubuntu insights figure 1 (23)*

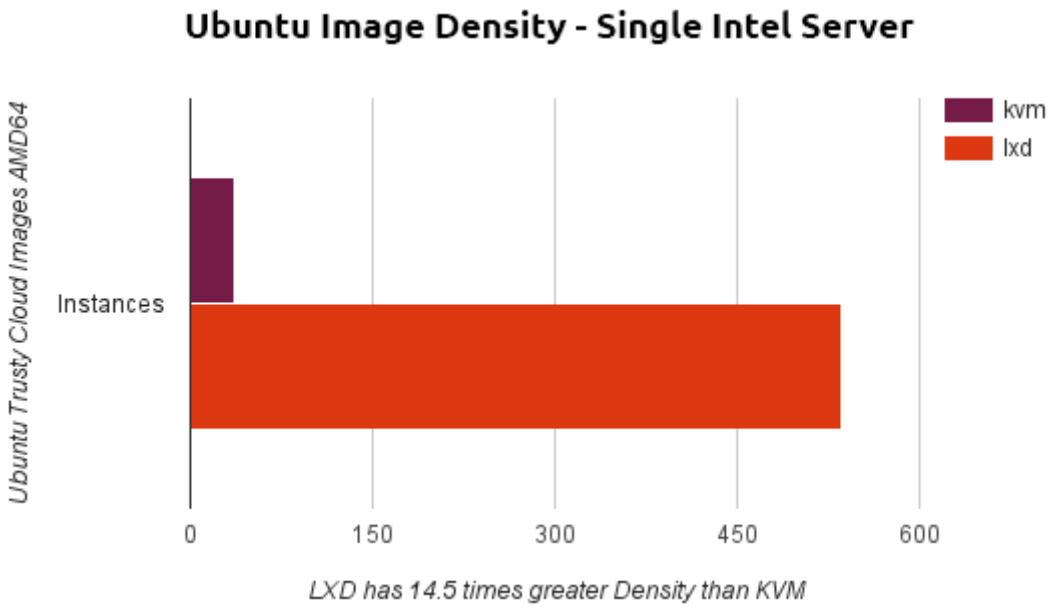


Figure 8: *Image Ubuntu insights figure 2 (23)*

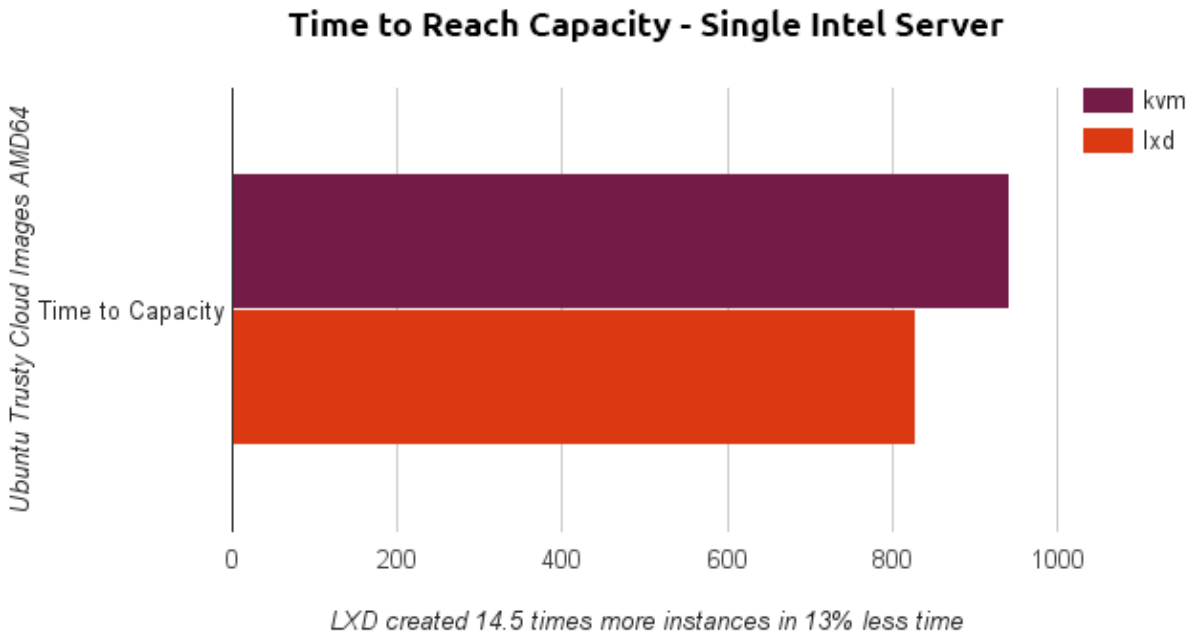
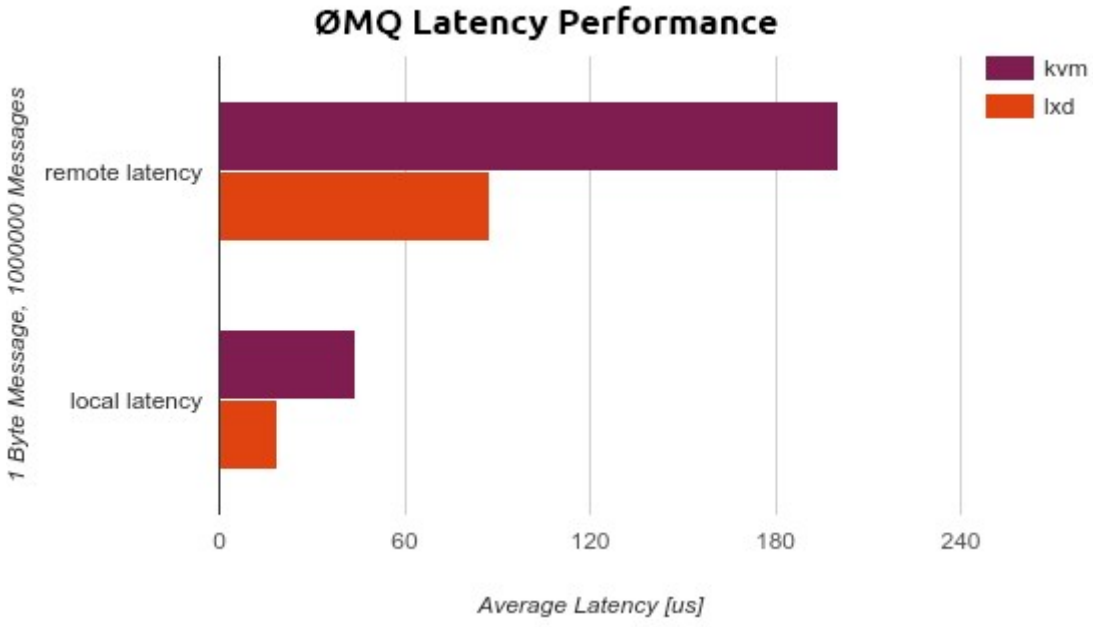


Figure 9: *Image Ubuntu insights figure 3 (23)*



One would expect that there will be variances between KVM and other Hypervisor solutions. Considering a fully virtualised server comes in sizes varying from 100's of Mbps to multiple Gbps and a single container can be the size of few Mbps it is easy to see that VMs/ host are going to be much greater for container-based solutions.

3.3.3 - Comparing the Performance of Hypervisors and LXC

Several benchmarking experiments have been run comparing LXC and other virtualisation techniques including traditional hypervisor solutions. Studies by Hwang et al. (12), Beserra et al. (25) and Lakew et al. (26) have consistently shown that LXC is on a par other virtualisation technologies, in some case better and in others worse but overall on par.

In an experiment run by Walters et al. (27), we have the closest approximation to an extremely large load on a data centre of the four experiments reviewed. In this experiment, the researchers measured GPU passthrough for four virtualisation technologies: KVM, Xen, VMWare ESXi, and LXC. The experiments were performed using CUDA (28) a parallel computing platform created by NVIDIA and OpenCL, a framework for creating parallel computing applications (29). Once their systems where ready, they performed a series of benchmarking tests. The results of the tests are displayed in the figures below and they show that LXC performs as well as other virtualisation platforms or better. Annotations for the images are quoted directly from the paper. (27)

Figure 10: *SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.*

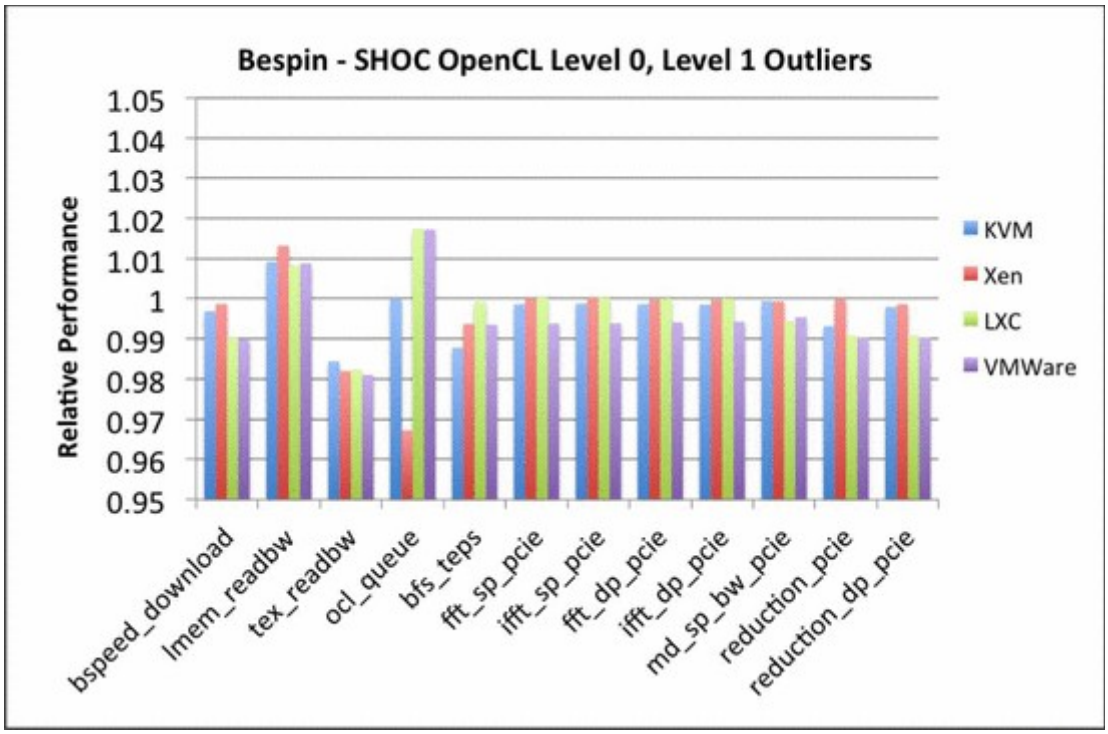


Figure 11: SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

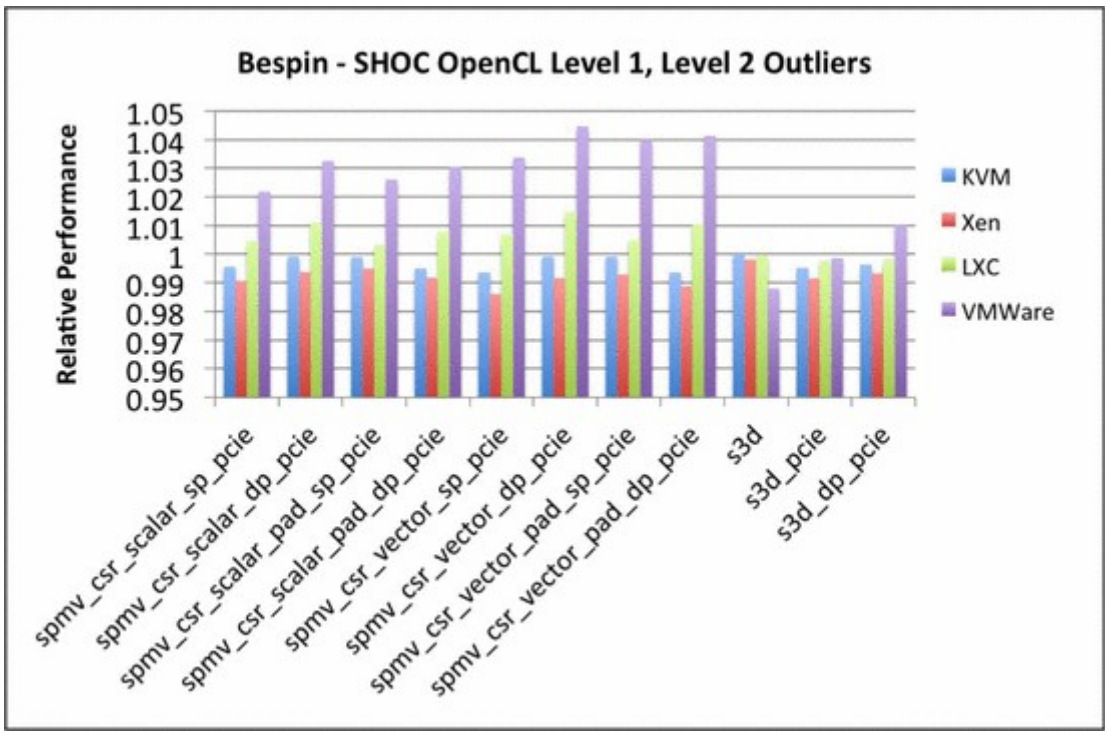


Figure 12: *SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.*

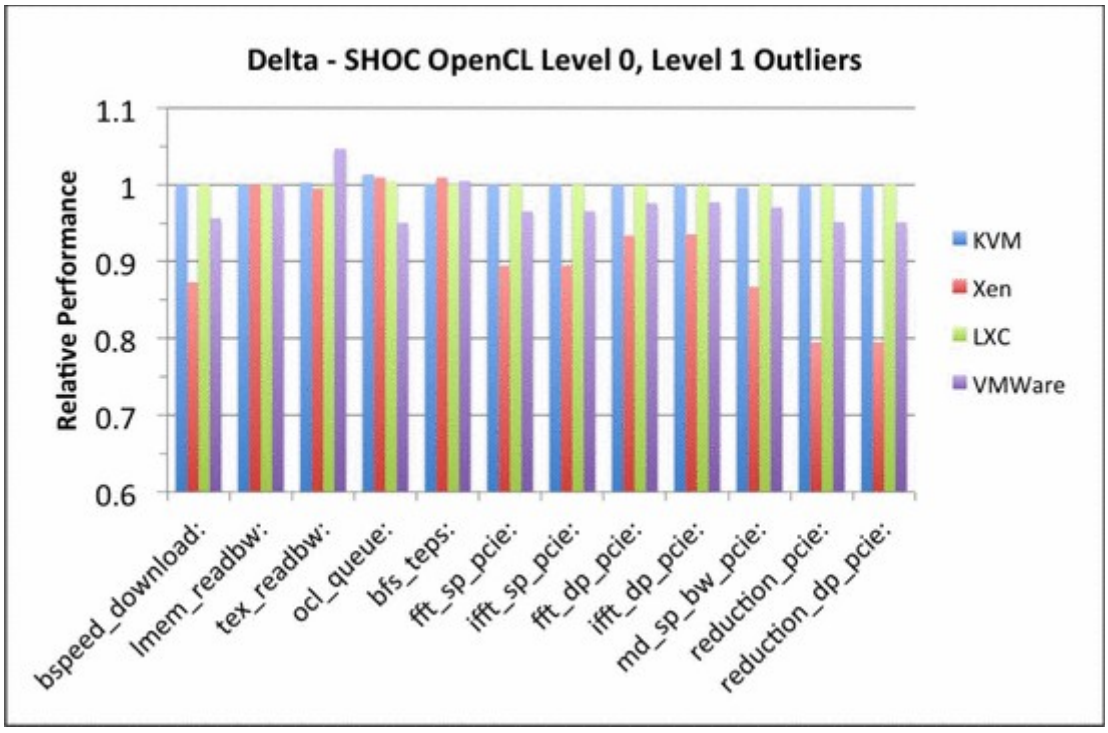


Figure 13: *SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.*

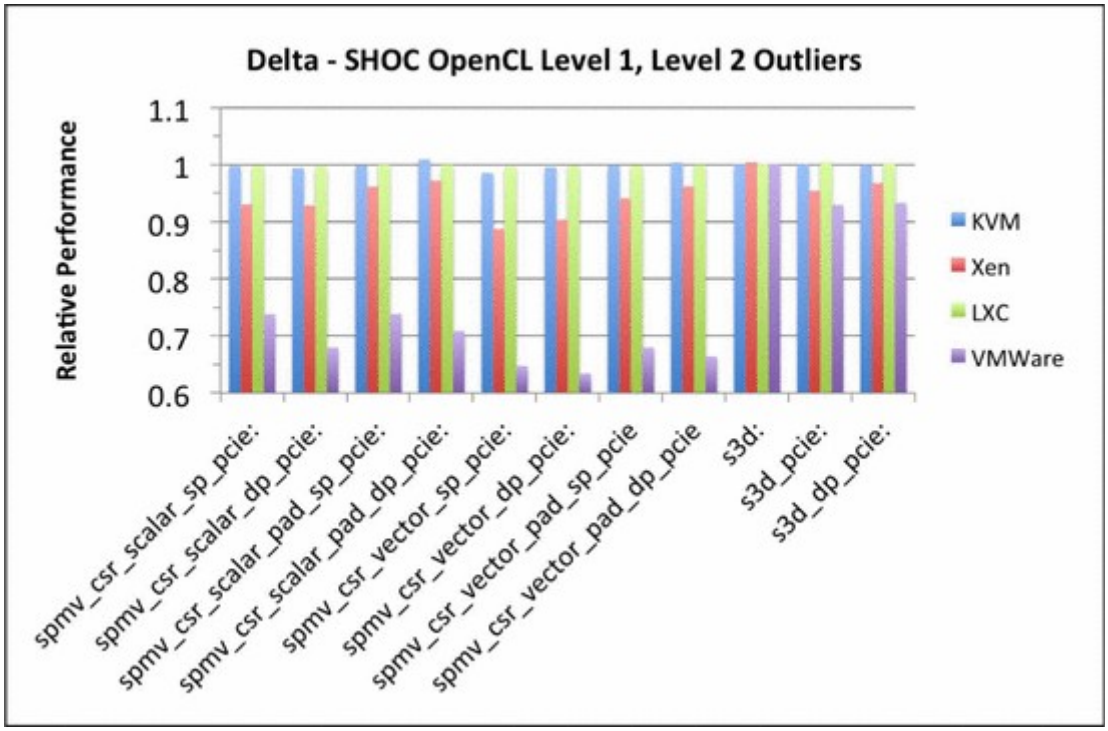


Figure 14: *GPU-LIBSVM relative performance on Bespin system. Higher is better.*

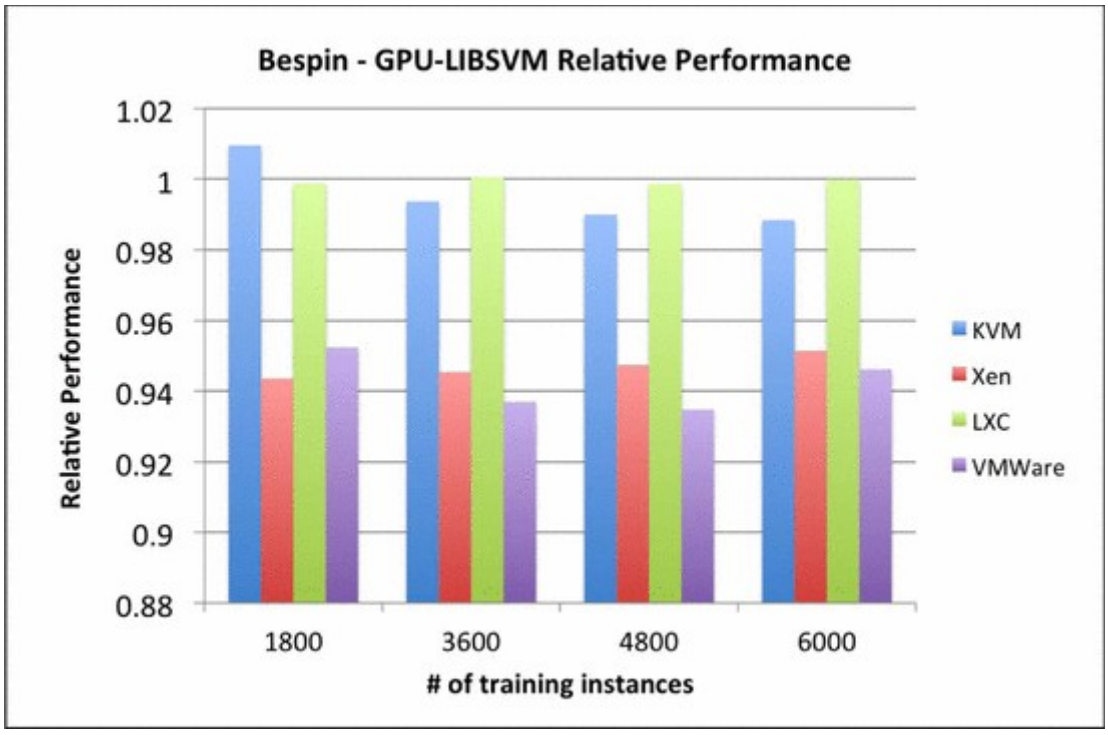


Figure 15: *GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.*

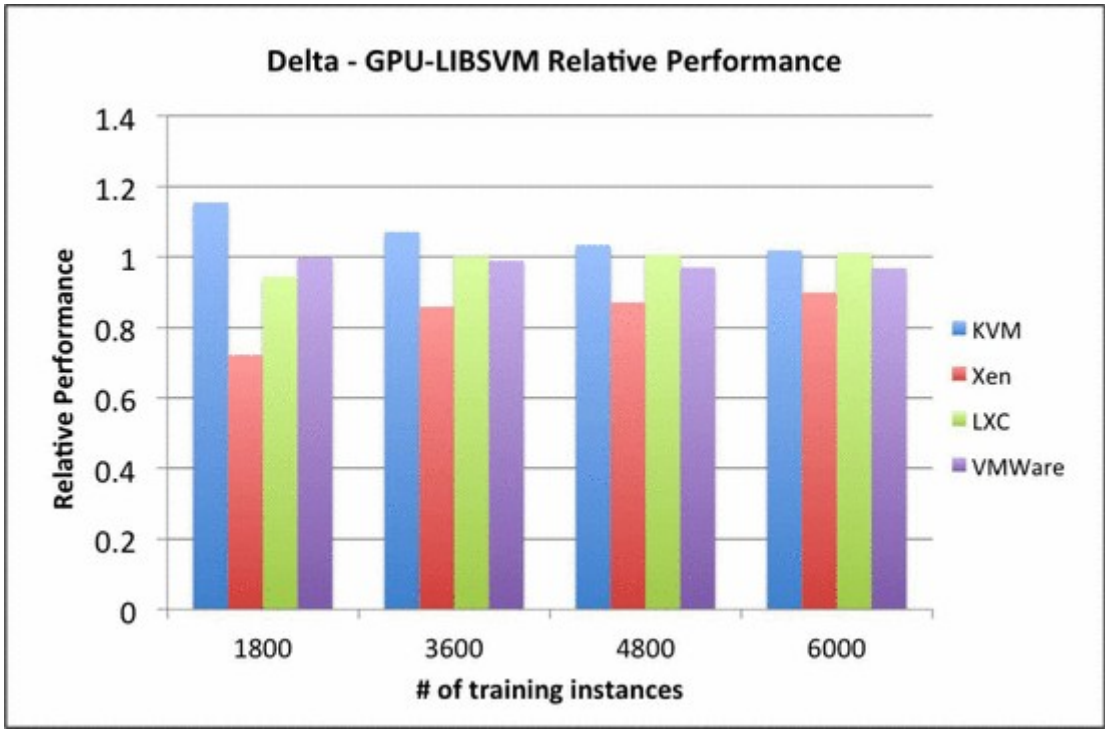


Figure 16: *LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.*

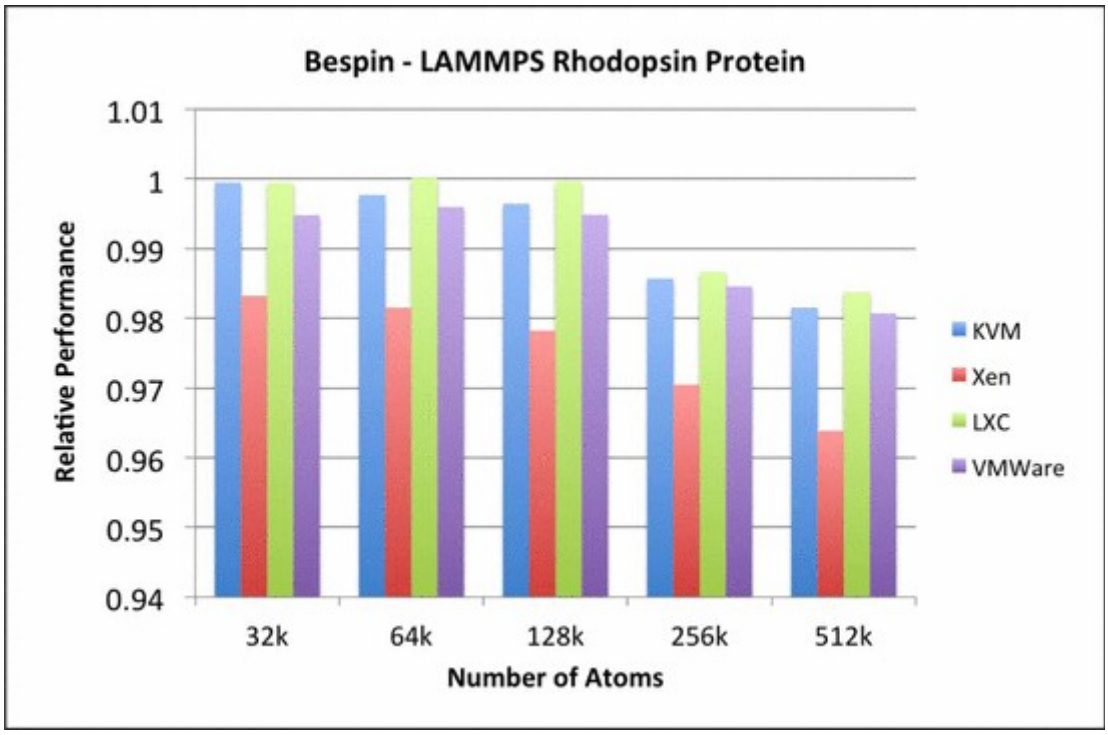


Figure 17: *LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.*

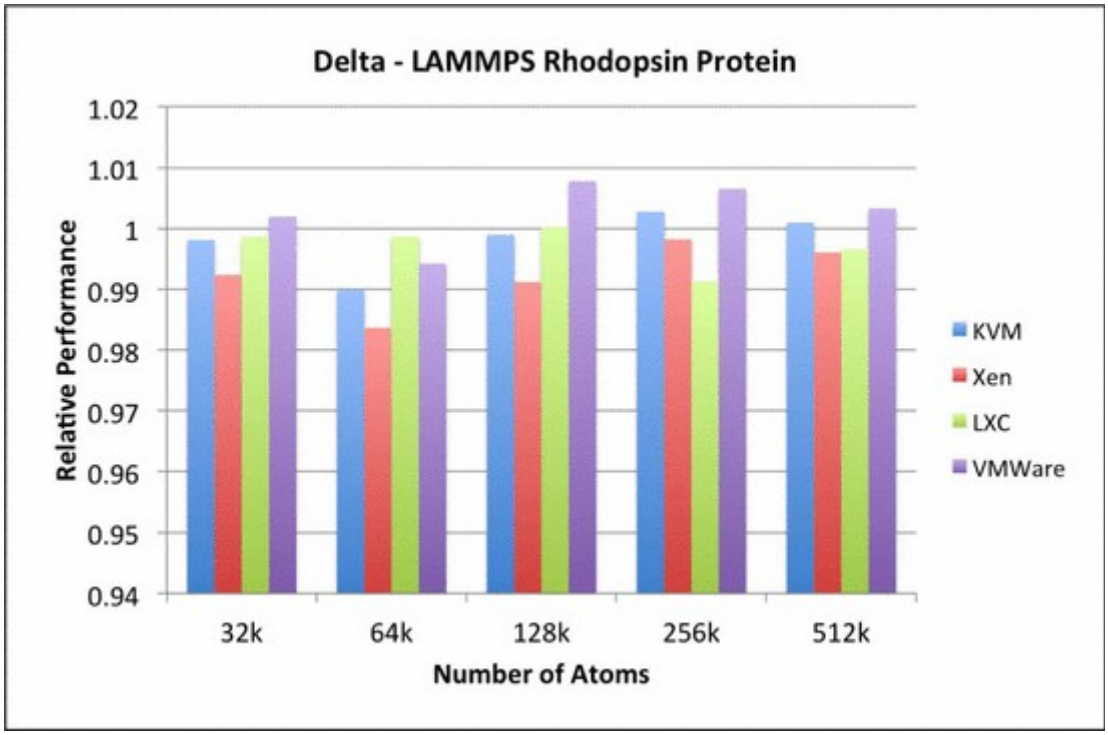
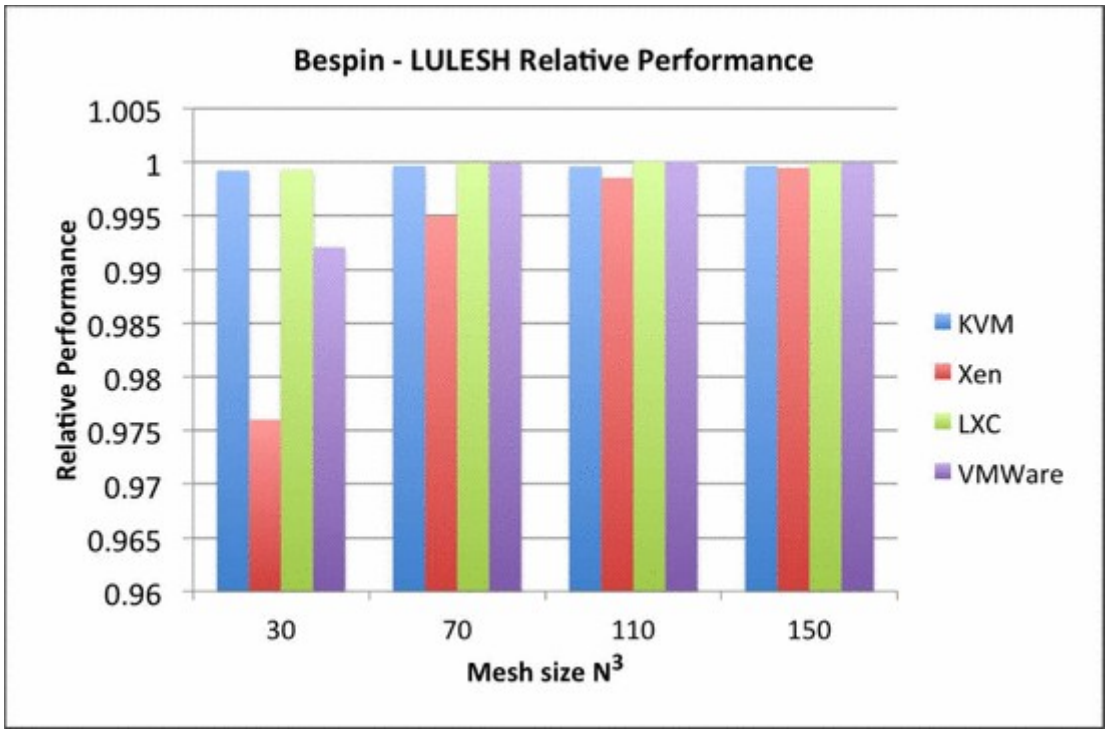


Figure 18: *LULESH relative performance on Bespin. Higher is better.*



3.4 - Conclusion

The research conducted for the project focused on two key areas: scalable architectures for web applications and on promising virtualisation technologies for use in the application.

Concerning scalable architecture, both vertical and horizontal architecture proved to be viable candidates. In the end though considering the type of application being developed for the project, a hosting service for websites, the horizontal architecture proved to be most suitable. Regarding cost, where both types have similar capacity, horizontal architecture proved to be much cheaper. With regard to long-term scalability, it is impossible to predict when a service of the type being developed for the project would hit maximum capabilities of an application with a vertical architecture. Add these points to the fact that a web hosting service needs to maintain isolation between clients virtual machines and a horizontal architecture is the only way achieve that goal.

Concerning virtualisation, again all the technologies examined would have been feasible for use as virtualisation platforms within the project. Some would have been disqualified by virtue of their cost, and because they are closed proprietary platforms, others though do have open-source counterparts. With that said, it is clear that LXC is the best choice. Regarding performance, it is equal to or better than other virtualisation platforms. On those terms, this would make LXC a right choice for the project. Where LXC excels though is in VM density per server at 14.5 times greater density, launches instances 94% faster and has 57% less latency. Given the authors limited hardware capacity this makes LXC the clear winner.

4 - Project Requirements

4.1 - Overview

Project requirements are a set of measurable metrics that allow one to determine if the project has been successful or not. The goal of this chapter is to distil the research into a set of these metrics and to use these requirements to define the features that this project's final product will contain. The author has always approached this project from the position that this would be the first of many projects, all with the ultimate goal of creating a website-hosting service that includes a data centre, all management tools and all features in place. Considering the time constraints for the project it is, of course, impossible to produce an entirely working solution; therefore, the following sub-chapters will specifically outline what features the author will and will not produce during the project implementation phase.

4.1.1 - Scoring of Requirements

The author will use a 5 point scale to indicate the importance of the feature or use case.

Table C: Requirements priority scale

P	Description	Priority
1	Requirement	High
2	Requirement	Medium
3	Requirement	Low
4	Nice to have	None
5	Future Development Cycle	None

Features or use-cases given a score one, two or three will be produced as a feature of the project implementation, the success or failure to produce these features will be the success or failure of the project.

Those features/ use cases given a score of four will only be attempted if all higher priority requirements have been completed, tested and documented. Requirements given a score of four should be considered ones that will produced as high priority requirements in the next project development phase.

Requirements given a score of five are features or use cases that would be required if the system where going into production at the end of this development phase. A requirement with a score of five will not be attempted.

4.1.2 - Naming Conventions and Definitions

Throughout the requirements chapter the following terms are defined as follows:

Front-end: This refers to all system components providing the website such as the site itself and the server hosting the site. Conceptually front-end components will be internet facing components.

Back-end: This refers to components that provide supporting services to the website. Conceptually back-end components will not be internet facing components.

Customer: Refers to any party that only interfaces with systems front-end.

Administrator: Refers to any party that interfaces with both the systems front-end and back-end.

4.1.3 - Project Scope

The scope of this project is limited to creating a basic web-browser interface that will allow a customer to create a website using a web-server and a CMS (Content Management System).

The system will then host the client's website and provide backups and minimal scalability.

4.1.4 - Project Stakeholders

Table D: Project Stakeholders

Project Research	David Monaghan
Project Implementation	David Monaghan
Project Supervisor	Meabh O'Connor
Second Reader	Noreen Gubbins

4.1.5 - Project Constraints, Resources and Risks

4.1.5.1 - Time-frame Constraints

- 12 weeks
- 14 hours per week
- 168 hours total
- The project must be completed within the allocated time frame (12 weeks).
- Each individual component of the project will have to be completed to meet the weekly deadlines
- Prototype Cycles will determine any change in the requirements of the project until the design is finalised

4.1.5.2 - Budget and Resources

- There is no financial budget for this project
- Use of free and open-source software

4.1.5.3 - Miscellaneous Risks

- Human error
- System failure

4.2 - Requirements

4.2.1 - Application/ Service Requirements

Table E: Application/ Service Requirements

#	The system shall...	Type	Fit Criterion	p
1	Not depend on non-free and/ or proprietary products	Constraint	No features or components to depend on any proprietary products or to have any licensing costs	1
2	Support Ubuntu 14.04 +	Constraint	Any products used in the implementation stage must support Ubuntu14.04 or newer	1
3	Services is built securely	Constraint	Each application component configured to best security practices	1
4	Application tier meets modern design standards	Constraint	Responsive to screen size and other design best practices	4
5	Application tier available as native mobile applications	Constraint	IOS, Android, Microsoft etc.	5
5	Service meets all legal requirements	Constraint	Refers to data-protection, consumer protection	5
6	Application responds automatically to changes in demand	Functional	Responds to changes in traffic to customer(s) websites	5
7	Application built with full redundancy/ fail-over	Functional	Application designed to avoid failure one node crashing entire application	4
8	The application and services are easy to manage	None-Functional	The application tier should have both GUI and CLI	4
9	All data should be recoverable	Functional	Includes real-time replication and back-ups	5
10	All services respond efficiently and quickly	None-Functional	Service should match or exceed customer expectations and/ or comparative services from competitors	5
11	All services have 100% availability	None-Functional		5

4.2.2 - Customer Requirements

Table F: Customer Requirements

#	The system shall...	Type	Fit Criterion	P
1	Web browser interface to manage customers web-site	Functional		1
2	Web-site is HTML 5 compliant	Constraint	This gives best out of box cross-platform support	1
3	Web-site responsive to form factor	Constraint	This gives best out of box cross-platform support	1
4	Ability to create web-server via web browser interface	Functional		1
5	Ability to configure web-server via web browser interface	Functional		5
6	Ability to create CMS via web browser interface	Functional		1
7	Ability to configure CMS via web browser interface	Functional		1
8	Ability to stop customer web-site via web browser interface	Functional		2
9	Ability to start customer web-site via web browser interface	Functional		2
10	Ability to restart customer web-site via web browser interface	Functional		2
11	Ability to create a back-up of customer web-site via web browser interface	Functional		3
12	Ability to restore customer web-site from back-up via web browser interface	Functional		3
13	Ability to create a snap-shot of customer web-site via web browser interface	Functional		3
14	Ability to restore customer web-site from snap-shot via web browser interface	Functional		3
15	Ability to delete customer web-site from snap-shot via web browser interface	Functional		2
16	Ability to create a customer account	Functional		5
17	Ability to update customer account	Functional		5
18	Ability to delete customer account	Functional		5
19	Ability to log an issue/ complaint/ question	Functional		5
20	Access to FAQ/ Help section	Functional		5

#	The system shall...	Type	Fit Criterion	P
21	Ability to comment on a FAQ/ Help section	Functional		5
22	Ability to post question on customer forum	Functional		5
23	Ability to answer question on customer forum	Functional		5
24	Facility to set-up recurring/ automatic payment	Functional		5
25	Facility to manage recurring/ automatic payment	Functional		5
26	Localisation, i.e. language and currency	Functional		5
27	Ability to back up customer website to local-host	Functional	Local-host being the remote machine the customer is work upon.	5
28	Ability to pick Fully Qualified Domain Name	Functional		5
29	System automatically updates relevant DNS	Functional		5
30	Customised SEO services	Functional		5
31	Email service integrated to CMS	Functional		5
32	Integration of CMS with application dashboard	Functional		5
33	Live communication with support staff via VOIP/ IRC/ Other media	Functional		5

4.2.3 - Administrator Requirements

Table G: Customer Requirements

#	The system shall...	Type	Fit Criterion	P
1	Web browser interface to manage customers web-site	Functional		1
2	Ability to create web-server via web browser interface	Functional		1
3	Ability to configure web-server via web browser interface	Functional		5
4	Ability to create CMS via web browser interface	Functional		1
5	Ability to configure CMS via web browser interface	Functional		1
6	Ability to stop customer web-site via web browser interface	Functional		2
7	Ability to start customer web-site via web browser interface	Functional		2
8	Ability to restart customer web-site via web browser interface	Functional		2
9	Ability to create a back-up of customer web-site via web browser interface	Functional		3
10	Ability to restore customer web-site from back-up via web browser interface	Functional		3
11	Ability to create a snap-shot of customer web-site via web browser interface	Functional		3
12	Ability to restore customer web-site from snap-shot via web browser interface	Functional		3
13	Ability to delete customer web-site from snap-shot via web browser interface	Functional		2
14	Command line interface to manage customer web-sites	Functional		1
15	Ability to create web-server via command line interface	Functional		1
16	Ability to configure web-server via command line interface	Functional		1
17	Ability to create CMS via command line interface	Functional		1
18	Ability to configure CMS via command line interface	Functional		1

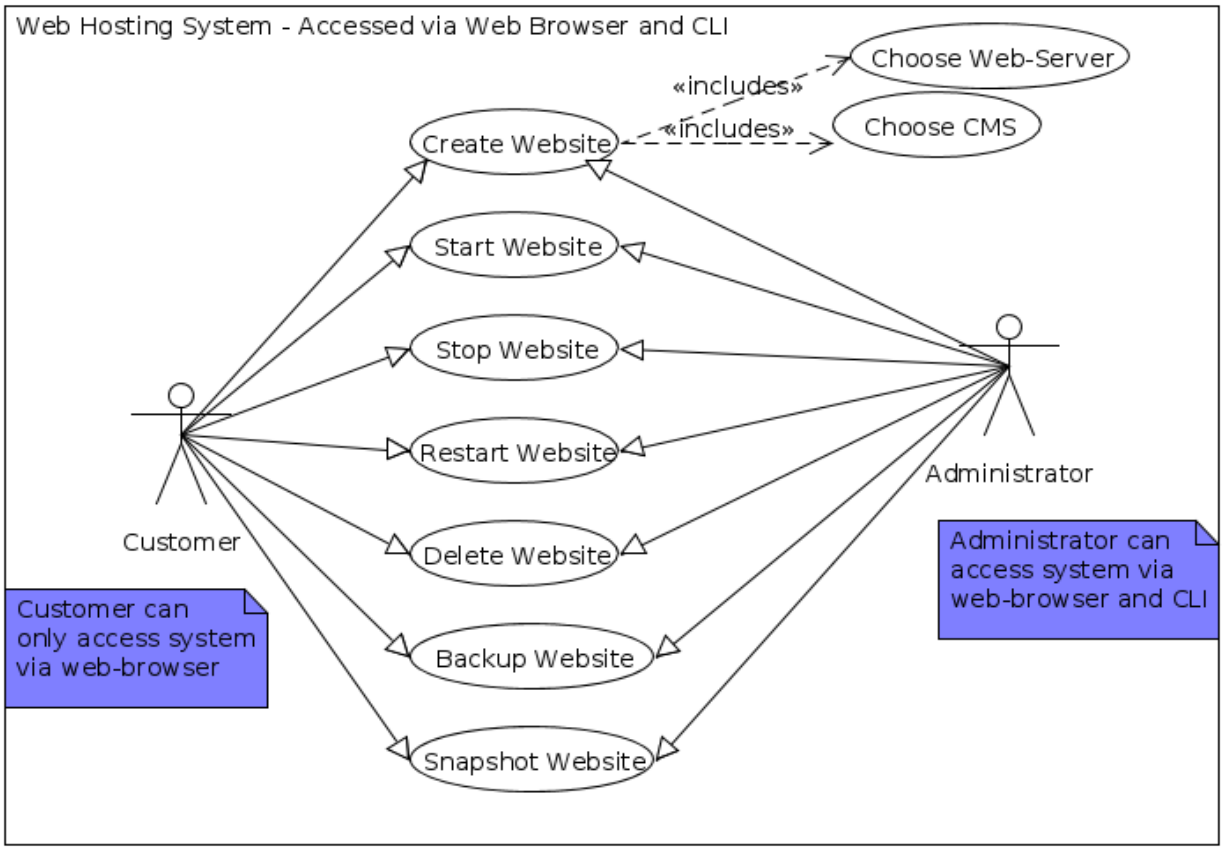
#	The system shall...	Type	Fit Criterion	P
19	Ability to stop customer web-site via command line interface	Functional		1
20	Ability to start customer web-site via command line interface	Functional		1
21	Ability to restart customer web-site via command line interface	Functional		1
22	Ability to create a back-up of customer web-site via command line	Functional		1
23	Ability to restore customer web-site from back-up via command line	Functional		1
24	Ability to create a snap-shot of customer web-site via command line	Functional		1
25	Ability to restore customer web-site from snap-shot via command line	Functional		1
26	Ability to delete customer web-site from snap-shot via command line interface	Functional		1
27	Ability to add new web-server options via web browser interface	Functional		5
28	Ability to add new web-server options via command line interface	Functional		1
29	Ability to add new CMS options via web browser interface	Functional		5
30	Ability to add new CMS options via command line interface	Functional		1
31	Ability to create a customer account	Functional		5
32	Ability to update customer account	Functional		5
33	Ability to delete customer account	Functional		5
34	Ability to log an issue/ complaint/ question	Functional		5
35	Access to FAQ/ Help section	Non-Functional		5
36	Ability to add new items to FAQ/ Help section	Functional		5
37	Ability to comment on a FAQ/ Help section	Functional		5
38	Ability to post question on customer forum	Functional		5

#	The system shall...	Type	Fit Criterion	Priority
39	Ability to answer question on customer forum	Functional		5
40	Facility to set-up recurring/ automatic payment accounts	Functional		5
41	Facility to manage recurring/ automatic payment accounts	Functional		5
42	Ability to back up customer website to local-host	Functional	Local-host being the remote machine the customer is work upon.	5
43	Ability to assign Fully Qualified Domain Name to customer website	Functional		5
44	System automatically updates relevant DNS	Functional		5
45	Customised SEO services	Functional		5
46	Email service integrated to CMS	Functional		5
47	Integration of CMS with application dashboard	Functional		5
48	Live communication with support staff via VOIP/ IRC/ Other media	Functional		5

4.3 - Conclusion

We can summarize the previous requirements that are to be produced during the implementation phase with the following use-case diagram:

Figure 19: *Requirements use-case diagram*



The use-case outline above is subject to the following constraints outlined in Table H:

Table H: Constraint Requirements

#	The system shall...	Type	Fit Criterion	p
1	Not depend on non-free and/ or proprietary products	Constraint	No features or components to depend on any proprietary products or to have any licensing costs	1
2	Support Ubuntu 14.04 +	Constraint	Any products used in the implementation stage must support Ubuntu14.04 or newer	1
3	Services is built securely	Constraint	Each application component configured to best security practices	1

5 - Project Design

5.1 - Overview

This chapter will outline the initial design for the project implementation. In keeping with research conducted for this project, the author will evaluate the feasibility of the two design paradigms investigated. Once a design paradigm has been chosen, the author will examine available technologies to implement the project using that design paradigm. With both the design paradigm and technologies selected, it will then be possible to create the initial network and application architecture.

During the requirements stage of the project, it was determined that any solution for the implementation phase of the project would require compatibility with Ubuntu 14.04 + and not be dependent upon any technology stack that is proprietary and/ or subject to licensing costs. These requirements are placed upon the author by the necessity of keeping costs to a minimum and to avoid the possibility of vendor lock-in or to become dependent upon into a single technology stack.

In the research phase of the project, it became apparent to the author that using LXC as the virtualisation technology is clearly the best path forward. This section will not go into further detail regarding LXC, as this topic has been covered in depth previously, the exception being the compatibility of technology stacks, investigated in this chapter, with LXC.

While the author has taken every care to ensure that each aspect of the design has been given full consideration, the author concedes that it is impossible to fully design a software solution in advance of development and that some or many aspects of the design may require a redesign during the project implementation.

5.2 - Application Architecture Design

During the research phase of the project, two design paradigms were investigated, a horizontal architecture design and a vertical design architecture. Both design paradigms have their advantages and disadvantages, and each has their individual use-cases. This decision is the most important decision of the design phase, as this is one that will most greatly affect the ability of the project to meet the project's none-functional requirements such as scalability, and not to exceed its constraints such as being built securely.

A vertical design architecture is one where the capacity of each component is increased to meet or exceed expected demand. Typically applications constructed in this way have components that are tightly coupled together. Advantages include lower administration costs, easier implementation, easier to manage and troubleshoot. Disadvantages include entire application/service needs to designed from the very beginning, failure of a single node can bring down application/ service, all costs need to be paid up front and the practical upper limits of software/ hardware capabilities.

With a horizontal design architecture, one increases/ decreases component nodes to meet changes in demand. Applications built using this design paradigm will follow a Service Oriented Architecture and using an N-Tier architecture to provide stateless communication via an API. Advantages of this architecture include not being limited to a single technology stack due to the use of APIs, no single point of failure and reduced costs as compared to a vertical solution with similar capacity. Disadvantages of this architecture include greater complication in design and management due to the distributed nature of the architecture.

5.2.1 - Recommendation

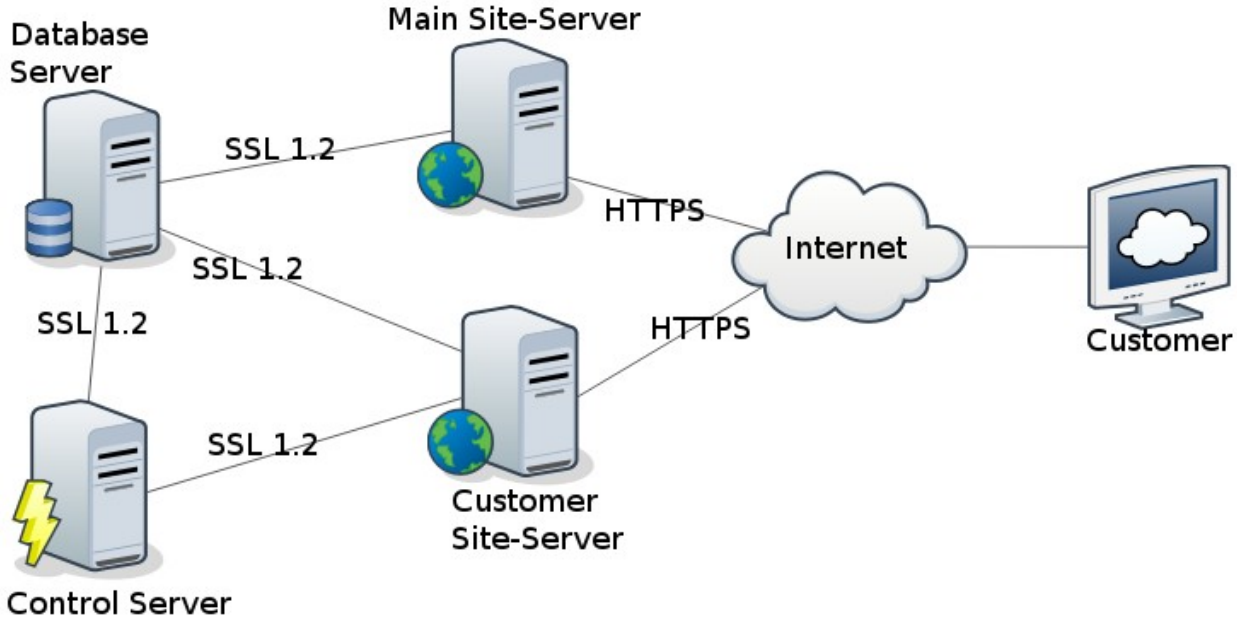
The author considered recommendation would be to use a horizontal scaling architecture in the design of the application.

1. The purpose of the project is to create a service that will host the websites of third parties. The very nature of this service requires some isolation of each site from the other. The easiest way to achieve this isolation is to host each site on its own server which immediately means the application will need to be able to expand horizontally.
2. Decoupling each component of the application makes it easier to secure each of those components.
3. Decoupling each component of the application makes it easier to meet other non-functional requirements of the application that will be addressed in future development cycles such as availability, performance, redundancy and scalability.
4. Decoupling each component of the application makes it easier to upgrade each component using new/ improved technology stacks without disrupting the services provided by the other components.

5.2.2 - Network and Application Architecture Design

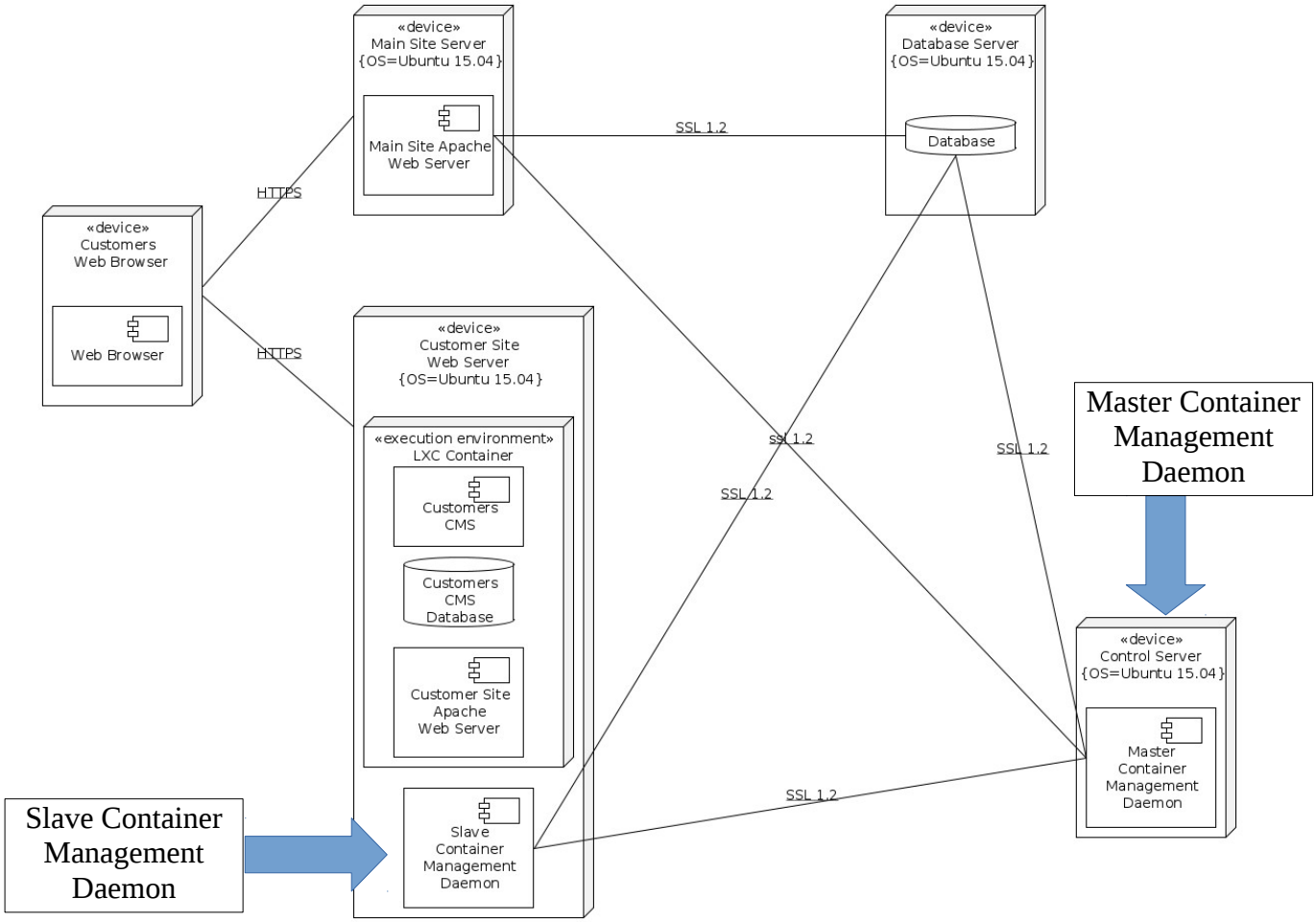
The system architecture follows a client-server model, each of the application tiers will be separated from each other by its own layer. In fig 20 below we can see a high-level view of the network architecture. These are the main components needed to deliver on the requirements outlined earlier. The image shows that the project will be implementing HTTPS for any internet traffic and also implementing SSL 1.2 on the ports required for communication between the servers. In line with the proposed horizontal architecture, each of the components will be logically separate from each other.

Figure 20: *High view network diagram*



In figure 21 below we can see a more detailed model of the planned architecture.

Figure 21: *Application architecture diagram*

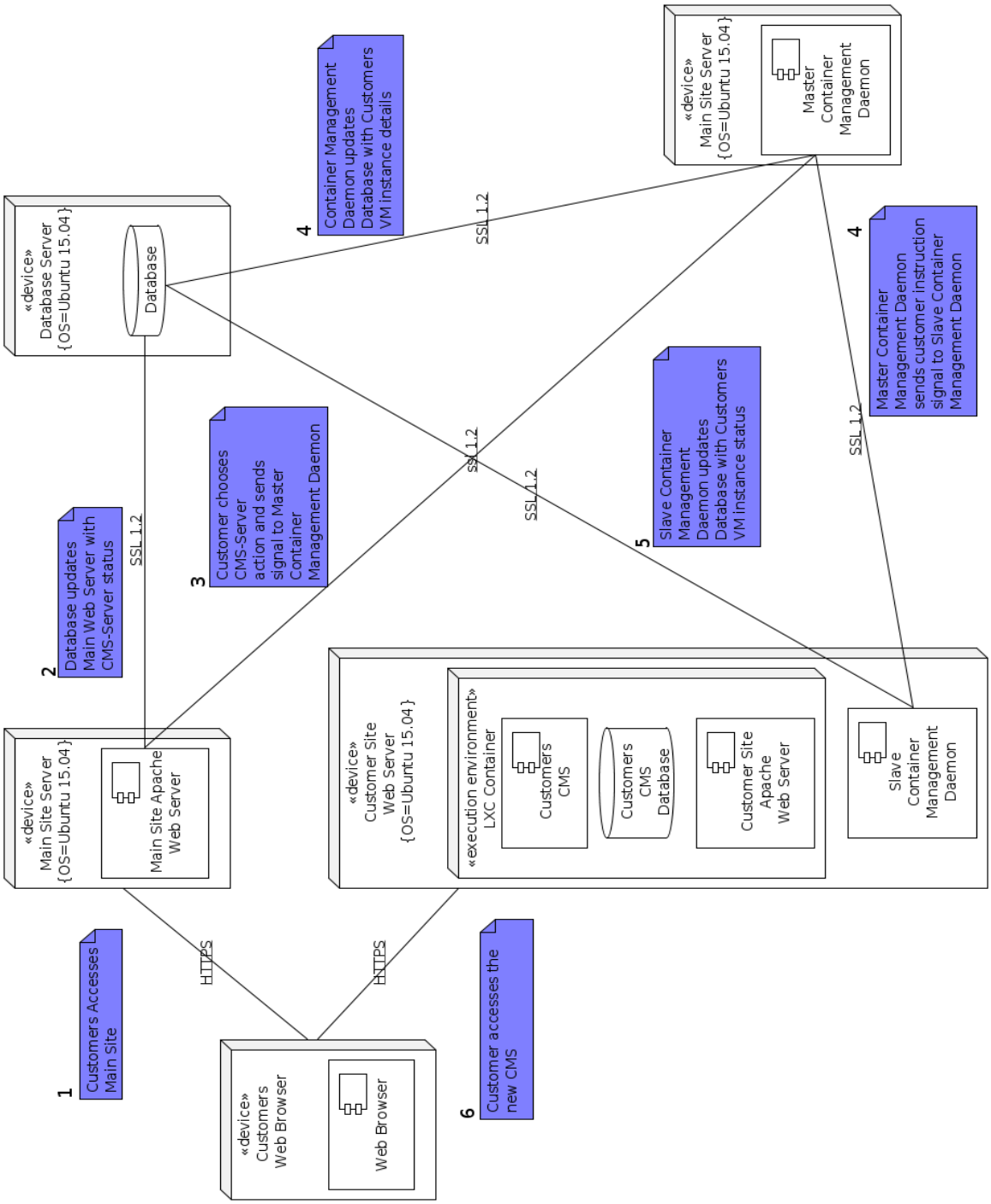


There are a few things to note about figure 21:

1. There are two control Daemons, a master and slave Daemon.
2. The master Daemon will co-ordinate activity between the main web-server, database and the customer site web-server.
3. The slave Daemon will perform any actions required of any containers on its server, once signalled to do so by the master Daemon, and then update the database with the containers status.

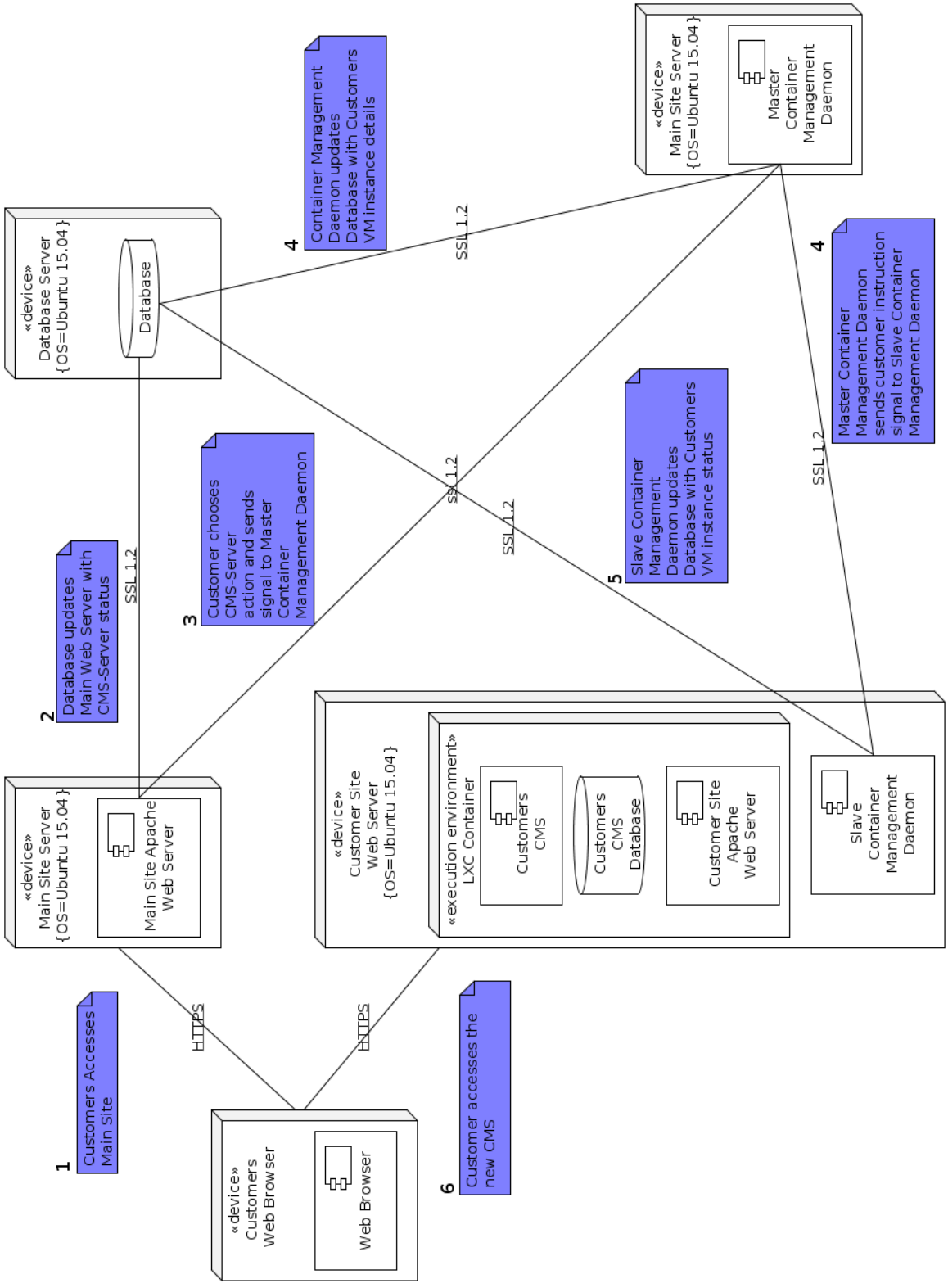
In figure 22 we can see the sequence of events that will take place when a customer or administrator requests an action be taken on a container.

Figure 22: *Process diagram for changing container VM state*



In figure 23 we can see the sequence of events that will take place when a customer or administrator requests an action be taken on a container.

Figure 23: *Process diagram for creating a container VM*



5.3 - Presentation Tier

An administrator will be able to access the application via a CLI on an individual server and via an internet browser. The customer will only be able to access the application from a browser. For this section, the presentation tier will refer only to internet access of the system and not the CLI. As with all browsers the primary protocol for accessing content is HTTP delivered using a mixture of HTML/ CSS/ JavaScript/ Server side scripting. There are many ways to go about creating the HTML/ CSS/ Javascript for a website from Content Management Systems to web application frameworks and coding the pages manually using PHP/ HTML/ CSS.

CMSs (Content Management Systems) are a set of pre-programmed web-page templates that have been integrated with a database such as Wordpress and Drupal. Customisation is achieved by selecting options from a database and refreshing the site. CMSs are designed to allow the developer to avoid as much coding and technical work as possible, instead enabling them to concentrate on creating a professional looking website and creating content for the website. CMSs are ideal for people who lack the coding or technical experience required to build a site from scratch. They are also ideal when one needs to build a web-site quickly and one that does not require much customisation. A CMS can be extended with the use of themes and extensions to increase functionality although a lot of these require a subscription.

A web application framework is a software framework used to create web pages, web applications, web resources and web APIs. Web application frameworks are similar in concept to a CMS, where they are trying to avoid as much of the technical difficulty as possible. The difference being that web application frameworks require the use of programming e.g. Python for the Django framework. Web application frameworks hide the complexity involved in dealing with HTTP requests and provide libraries for automatically responding to these requests with pre-generated HTML/ CSS.

The final method is to use server-side scripting, such as using PHP to generate the HTML/ CSS. PHP is used to create dynamic web pages that respond to the users interaction with the site. PHP can be used in conjunction with web application frameworks, and it is portable to most modern systems. PHP is processed by a PHP interpreter that is a CGI (Common Gateway Interface) module on a web server. PHP can also be run on a CLI of a server for testing purposes.

5.3.1 - Recommendation

The author recommends using server-side scripting to generate HTML/ CSS for the site as the most feasible option.

1. There will some business logic processing needed to be performed by the main site server. PHP is an excellent candidate for this task as it can work well with databases, APIs and with other programming languages via TLS ports.
2. Due to the time constraints on the project, the author will not have the time available to learn a new framework.

5.3.2 - Main Website Prototype

Figure 24 is the prototype for the homepage of the main site

Figure 24: *Homepage mockup*

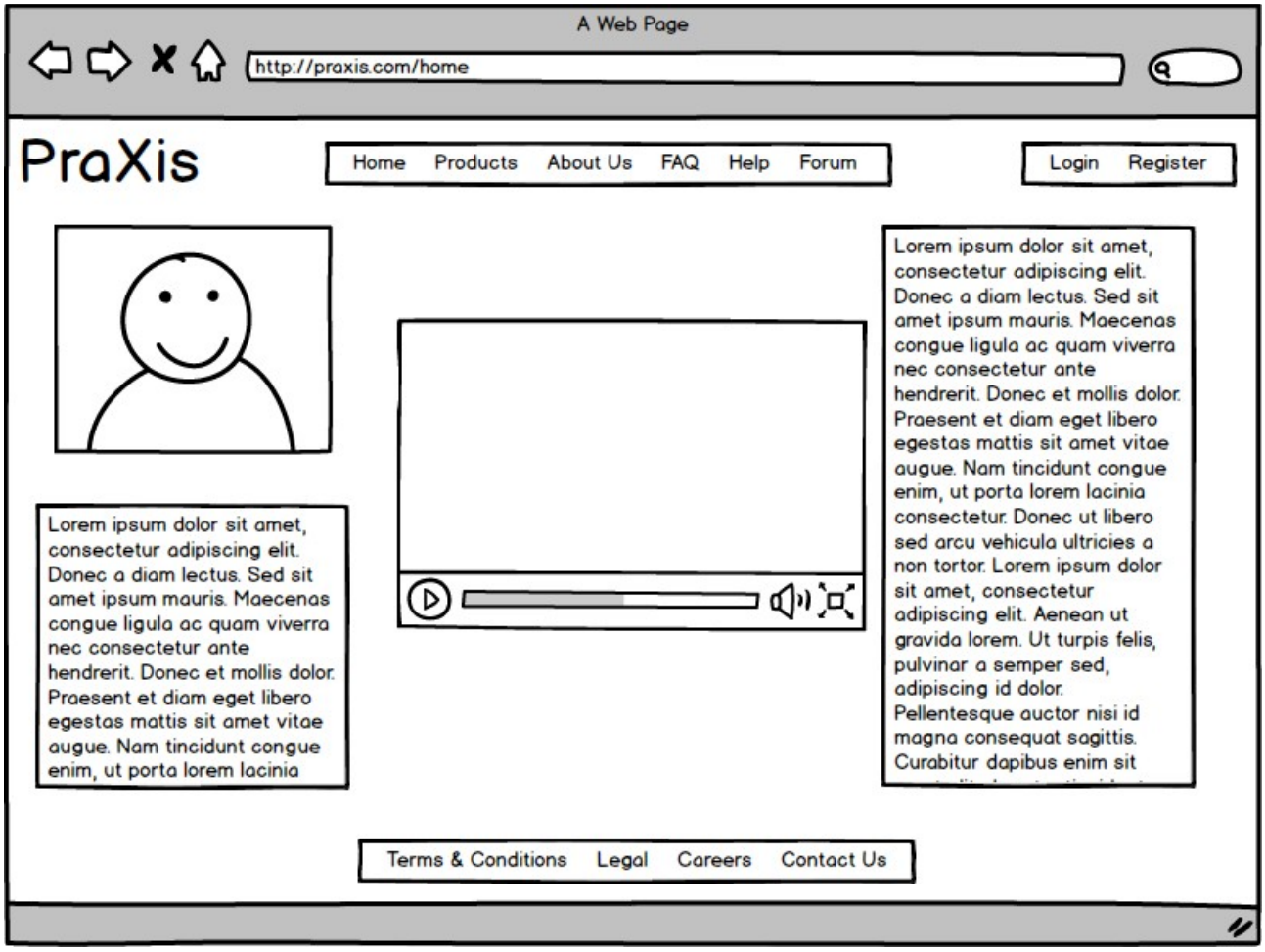


Figure 25 is the prototype for the page where a customer chooses their container and server options.

Figure 25: *Container-server-CMS choice web-page*

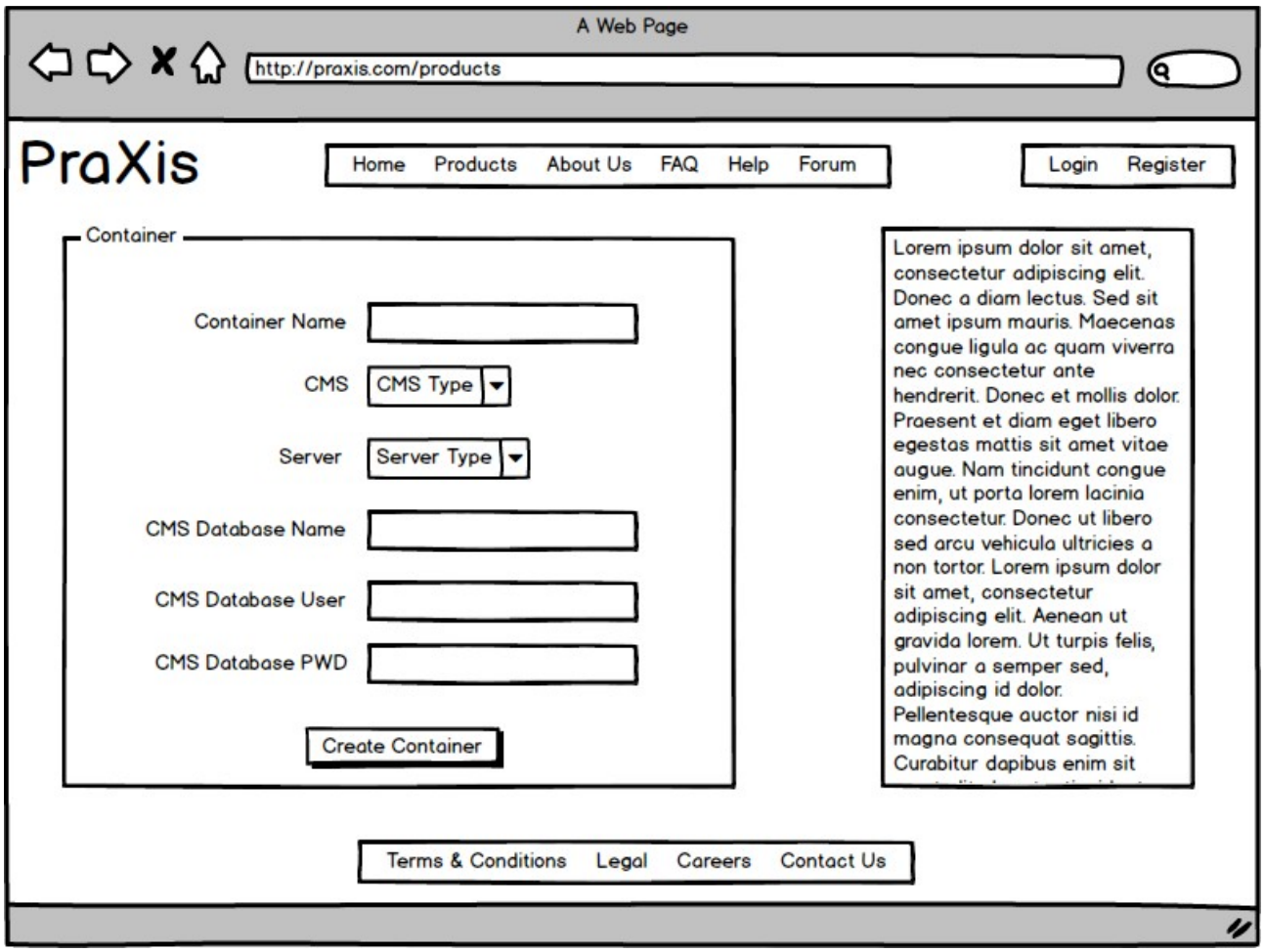
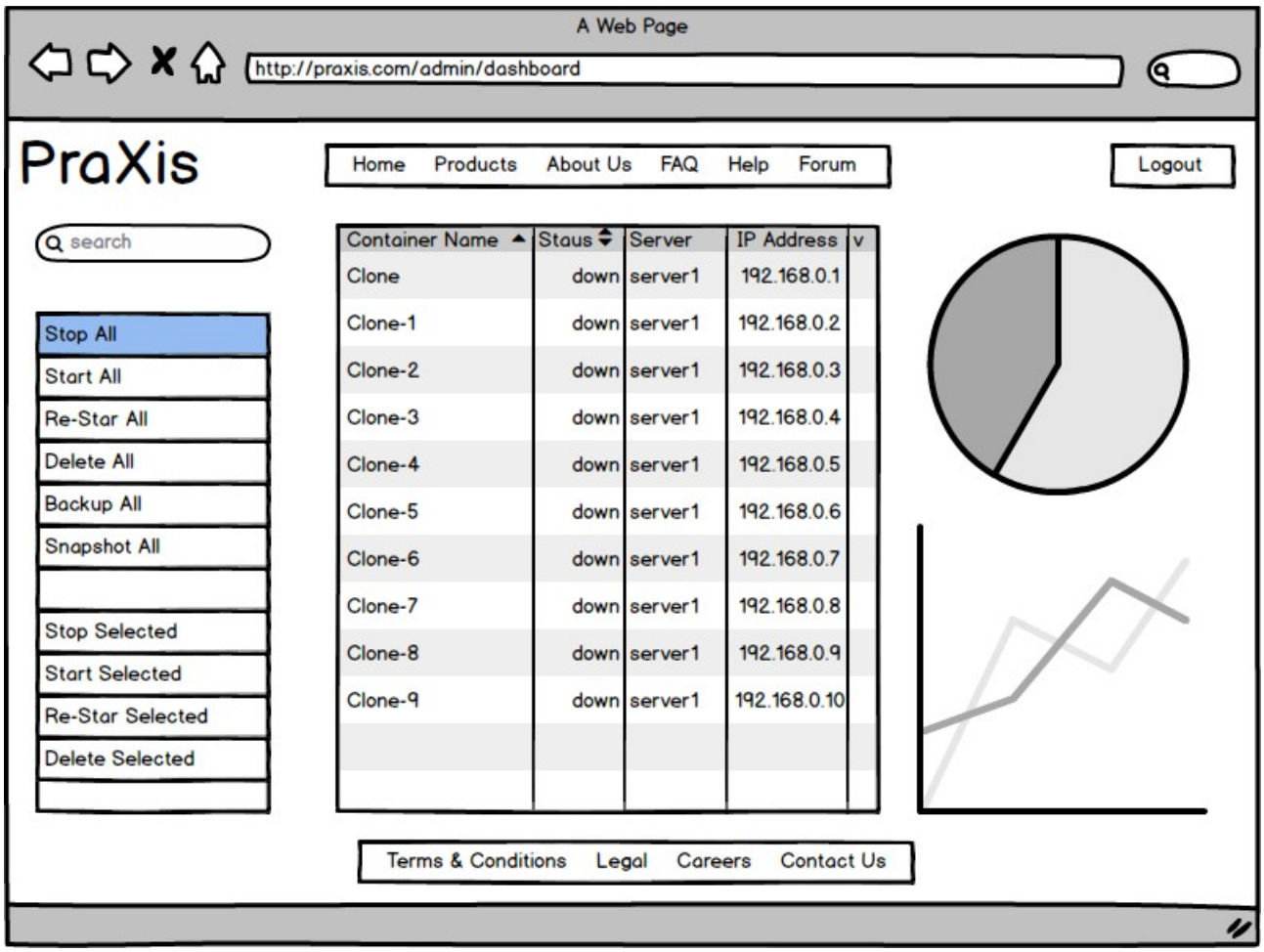


Figure 26 is the prototype for the page where a customer/ administrator can manage containers

Figure 26: *Customer/ admin dashboard web-page*



5.4 - Logic Tier

The logic tier encodes application rules that determine how data is used, created, stored or changed. The logic tier coordinates activities between the different tiers of the application, the presentation tier, and the data tier. The logic tier also processes commands, performs logical calculations, makes decisions and evaluations.

Considering this part of the application takes place on the back-end of the application it will require a number of capabilities:

1. Compatible with Ubuntu 14.04 +
2. Able to interface with MySQL, OS, PHP , LXC
3. Able to communicate over SSL
4. Speed
5. Concurrency: this is the ability of the language to handle computations that are executing during overlapping time-frames. This is important in a network programming environment as a server may receive multiple requests for the same service at the same time and may have to wait a differing amount of time to respond, while it waits for responses from another part of the application.

A lack of official support for Ubuntu 14.04 + and LXC will be grounds for disqualification.

Four programming languages will be evaluated under the previous criteria

1. Go
2. Java
3. Perl
4. Python

Table I: Programming Languages Requirements Scores

	Ubuntu 14.04 +*	MySQL*	OS*	PHP**	LXC*	SSL*	Speed#	Concurrency*	Total
Go	2	2	2	0.5	2	2	4	1	15.5
Java	2	2	2	0.5	1	2	3	1	13.5
Perl	2	2	2	1	1	2	N/A	1	11
Python	2	2	2	1	2	2	2	1	14

***** Official Support = 2, Unofficial Support = 1, Not Supported = 0
****** Official Support = 1, Unofficial Support = 0.5, Not Supported = 0
# Fastest = 4, Slowest = 1 source (30)
Sources: Go-> (31) (32) (33) (34) (35) (36) Java->(37) (38) (39) (40) (41) (37)
Perl-> (43) (44) (43) (45) (46) (42) Python->(48) (49) (50) (51) (52) (53)

5.4.1 - Recommendation

From the table I we can see that Go and Python are the clear winners. Even had Java or Perl gotten winning scores they would have had to be disqualified due to lack of LXC support. Go edges past Python due to the speed of its run-time, which, in a study by (30) Nanz S, Furia for the 37th IEEE International Conference on Software Engineering, was found to perform significantly faster than other scripting languages. If we take Go's runtime speed along with its native support for LXC and given its ease of learning, and it is a clear winner.

PHP has previously been chosen for use in the presentation tier, where appropriate PHP will also be used as part of the logic tier when a business decision needs to make.

5.5 - Data Tier

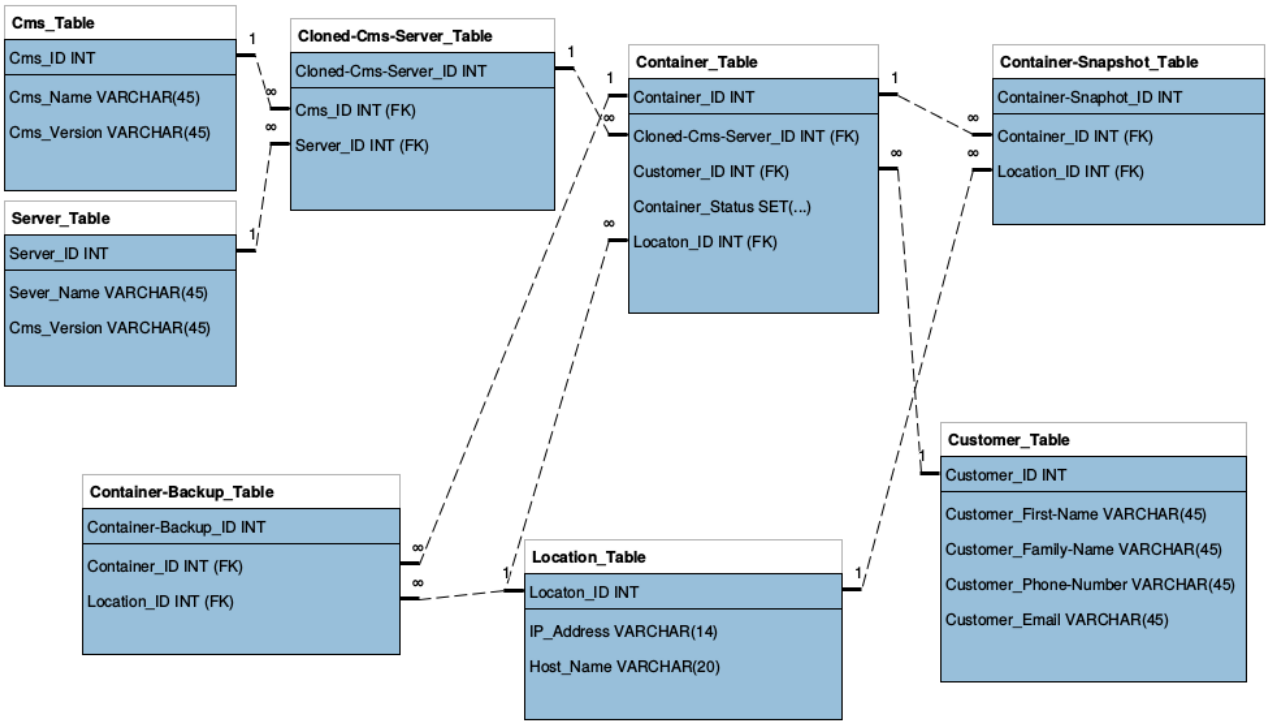
The data tier is where the application would store and retrieve data from either a file system or a database. Due to the previous decision to keep the application data stateless, this will preclude using a file-system from our analysis, due to the concurrent nature of data requests by the application. This is an application requirement.

The author did not explore alternatives to the original goal of using MySQL as the database server and client. There some reasons for this decisions:

1. Project time-frame does not allow for learning a new database system and its SQL syntax.
2. The stated goal of the project is to host a virtual machine instance that includes a web-server, itself hosting a Wordpress installation. A Wordpress installation requires MySQL database for operation. Due to this fact, the author felt it would be a boon to the project to avoid any confusion that may be caused by using two DBMS (Database Management System)

5.5.1 - Database Design Schema

Figure 27: *Prototype database schema*



5.6 - Conclusion

After exploring a number of technologies and their appropriateness for achieving the project's goals, which includes an initial design template for each aspect of a component of the application, the author believes that tools are now available for a successful project implementation. Experience with previous projects has shown that early confidence can sometimes be misplaced, and the author is fully prepared to abandon either a technology or design if it clearly is not functioning as expected. Any changes to the design or chosen technology will be outlined in the final implementation report.

A summary of the technology chosen will conclude this chapter in Table J.

Table J: Summary of Selected Technology

Design Architecture	Horizontal Scaling Architecture
Presentation Tier	HTML/ CSS PHP
Logic Tier	Go PHP
Data Tier	MySQL

6 - Project Plan

6.1 - Overview

The project will be conducted using the Agile Project Management System. Agile projects are carried out in small incremental stages where the product is developed iteratively and in cooperation with the stakeholders. This allows the development of a system that is as closely tailored to the stated requirements as is possible. The process involves a review at the beginning and end of each stage to ascertain the current status of all project elements. In consultation with the stakeholders, it is easier identify changing, mislabelled or previously unknown requirements and adapt the project to these changes quickly. Multiple short face-to-face meetings characterise the process. As each stage of the project is quite short, the stakeholder can quickly get an understanding of how the project is progressing and the direction it is heading.

6.2 - Goals and Objectives

1. Preparation of Development Platform
2. Development of Website
3. Development of Database
4. Integration of Website and Database
5. Development of Master Container Management Daemon
6. Integration of Master Container Management Daemon with Website and Database
7. Development of Slave Container Management Daemon
8. Integration of Slave Container Management Daemon with Master Container Management Daemon
9. Integration of Slave and Master Container Management Daemon with Website and Database
10. Testing
11. Documentation
12. Demonstration and Handover

6.3 - Project Plan

1. **Preparation of Development Platform**

   The first stage of the project will involve the installation and configuration of the required servers, Ubuntu and Apache. The configuration of the servers will involve locking down and securing servers and setting up passwordless private/ public key authentication. Once the servers have been locked down, each server will have the relevant software installed, configured and then be locked down. The final part of Stage one will involve the creation of the demonstration/ test LXC base clone VM.

2. **Development of Website**

   The second phase will involve the creation of the websites three pages. This will entail writing the HTML/ CSS for the web-pages. There will be some testing to ensure a minimum level of responsiveness to a change in web-browser form factors. While the author will accept purely functional pages for the website, there will be some thought into the aesthetic feel of the pages.

3. **Development of Database**

   An initial database schema has already been created. This phase will mostly involve stress testing the schema to ensure it will perform the functions required of the database. Any changes required to the database will be made at this stage.

4. **Integration of Website and Database**

   Stage 4 will involve combining the website with the database. PHP will be used to accomplish this task. Testing of the website and database will continue at this stage.

5. **Development of Master Container Management Daemon**

   This phase will involve the creation of the master management Daemon. The primary responsibility of this Daemon is to co-ordinate requests made by the customer or the administrator and ensure they are completed. The master management Daemon is responsible for managing daily tasks.

6. **Integration of Master Container Management Daemon with Website and Database**

   Stage 6 will see the master Daemon connected with the website and the database. Testing of this phase will be done to ensure that the Daemon is responding correctly to requests made by the administration or customer via the website and ensuring the database is updated appropriately.

7. **Development of Slave Container Management Daemon**

   The slave container daemon will be responsible for performing requests sent to it by the master management Daemon and updating both master management Daemon and the database with the current status of the customers instances.

8. **Integration of Slave Container Management Daemon with Master Container Management Daemon**

   This stage will be responsible for testing and debugging the interoperation of the two management daemons. That will involve ensuring that both Daemons are responding as expected and updating the database correctly.

9. **Integration of Slave and Master Container Management Daemon with Website and Database**

   The final stage of development will be to integrate all components of the application and perform testing of the integration and perform any required debugging.

10. **Testing**

    This will involve extensive testing of all components to ensure that they are both individually and collectively working predictably and as expected. To this point, testing will have been ongoing, the goal of this stage is to validate the final product and ensure it ready for handover.

11. **Documentation**

    Any documentation due for handover will be finalised and completed at the end of this stage.

12. **Demonstration and Handover**

    The final stage will involve a practical demonstration of the project deliverables and presentation of the projects findings and lessons learned.

6.4 - Expected Project Deliverables

The expected deliverables for this project are outlined below:

1. 3 page database-driven website
2. Website-hosting system with a horizontal architecture
    - Create new instance
    - Start, stop & restart an instance
    - Delete an instance
    - Backup and take snapshot of an instance
3. Demonstration of above
4. Final documentation

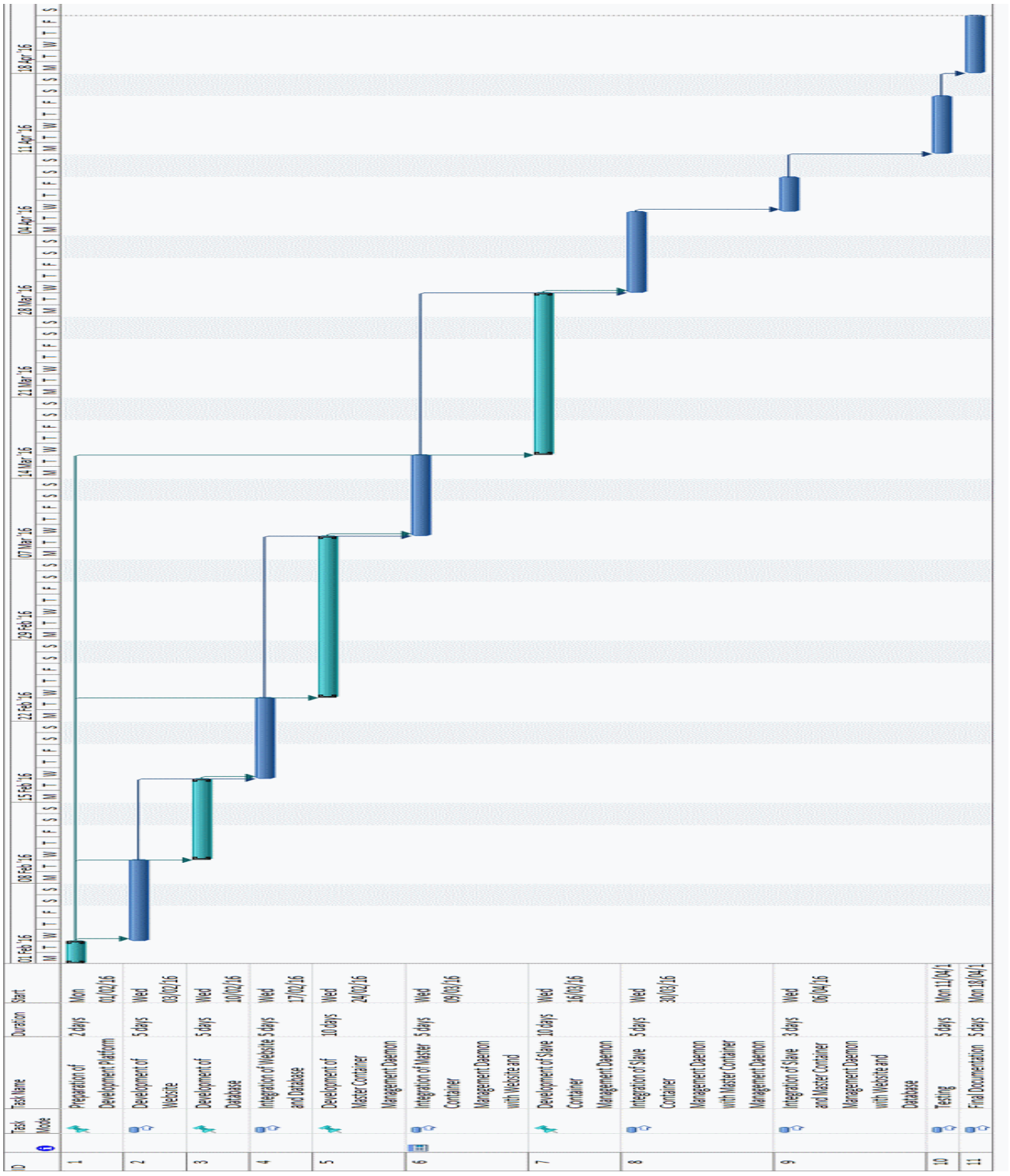
6.5 - Project Schedule

Table K: Project Schedule

Task ID	Task	Duration (Hours)	Immediate Predecessor
1	Preparation of Development Platform	7.0	None
2	Development of Website	14.0	1
3	Development of Database	14.0	1
4	Integration of Website and Database	14.0	2,3
5	Development of Master Container Management Daemon	28.0	1
6	Integration of Master Container Management Daemon with Website and Database	14.0	4,5
7	Development of Slave Container Management Daemon	28.0	1
8	Integration of Slave Container Management Daemon with Master Container Management Daemon	14.0	6,7
9	Integration of Slave and Master Container Management Daemon with Website and Database	7.0	8
10	Testing	14.0	9
11	Final Documentation	14.0	10
	Total	168.0	

6.6 - Gantt Chart

Figure 28: *Gantt Chart*



6.7 - Conclusion

The project plan and schedule are based upon estimated time-frames for each of the stages involved. It should be noted that time for unforeseen problems has been taken into account but previous experience suggests that is impossible to determine exactly how much time a possible delay will take up. With that noted, any change in schedule will be logged, the reason for that change noted and the schedule updated accordingly.

While delays cannot be planned, moving more quickly than expected is also something that cannot be planned. The project has been planned conservatively, and if it turns out to have been too conservative the author will be able reschedule and add some of the deferred requirements to this development cycle.

The project will be using the Agile methodology, this involves many face to face meetings with the stakeholders, who at these meetings will be notified of any changes and the reasons for those changes.

7 - Project Conclusion

With the conclusion of the project research phase and with the definition of the requirements, an initial design and project plan, one can now see a good outline of the overall path of the project and beyond. Upon reflection, the author must admit that the project has potentially far more scope than was originally envisaged and the author must thank his project supervisor, Meabh O'Connor, for helping narrow the scope of the project. At times, thoughts regarding the project extended to creating an entire data centre and all that would entail. Acting on the project supervisors advice allowed the author to pare the project back to its original goal.

An unexpected benefit came from this wide-ranging thinking on the project. That benefit was to think about how the project might be extended by further, and later, development cycles and what could be done at the first design stage to benefit those later development cycles. The author will concede that this is advice provided to students at the outset of the project and would answer that finding a way to do so, was not obvious at first.

Looking forward to the end of the implementation phase and the demonstration of the project, imagining that all requirements have been met, what might one expect to see? The project will be demonstrated in two ways; one way will be via an internet browser and second will be via the command line to show some features that just cannot be demonstrated via a browser. The browser demonstration will see the author picking the options on a web page, typing in some values such as a user-name, password and then hitting a button. After a couple of moments, the browser will redirect to another tab, and there will be the customers Wordpress installation, fully configured into Apache and MySQL.

Assuming the project requirements are met during the implementation phase, what would be the next projects requirements? That phase of development would involve more fully integrating the control Daemons into each component/ attribute of the customers VM instance, e.g., allowing them to set how many CPUs per VM or the amount of RAM. This stage would also be looking to provide the customer will more access to the Apache server and Wordpress settings. These new features would need to be available from a browser.

Development cycles would continue in this manner, first focusing on the core functionality between the administrator/ customer and the VM instances via the control Daemons. The following stages would begin to expand outwards from that core functionality, adding new tiers to the architecture as required. In the authors opinion the seeds for the long term success, or not, of the project have been sown at this stage.

8 - Appendix

8.1 - References

1. Matsudaira K. Building Scalable Web Architecture and Distributed Systems [Internet]. Dr. Dobb's. 2012 [cited 2015 Dec 30]. Available from: http://www.drdobbs.com/web-development/building-scalable-web-architecture-and-d/240142422

2. AWS. AWS_Well-Architected_Framework.pdf [Internet]. Amazon Inc.; 2015 [cited 2015 Dec 31]. Available from: https://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

3. Padnick J. A Comprehensive Guide to Building a Scalable Web App on Amazon Web Services - Part 1 [Internet]. 2015 [cited 2015 Dec 31]. Available from: https://www.airpair.com/aws/posts/building-a-scalable-web-app-on-amazon-web-services-p1

4. FlocPort. LXC Getting Started Guide Part II [Internet]. 2014 [cited 2016 Jan 8]. Available from: https://www.flockport.com/lxc-advanced-guide/

5. Craver N. What it takes to run Stack Overflow [Internet]. 2013 [cited 2016 Jan 1]. Available from: http://nickcraver.com/blog/2013/11/22/what-it-takes-to-run-stack-overflow/

6. Best Practices For Horizontal Application Scaling [Internet]. OpenShift Blog. [cited 2016 Jan 7]. Available from: https://blog.openshift.com/best-practices-for-horizontal-application-scaling/

7. Michael M, Moreira JE, Shiloach D, Wisniewski RW. Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: Parallel and Distributed Processing Symposium, 2007 IPDPS 2007 IEEE International. 2007. p. 1–8.

8. Khare A, Huang Y, Doan H, Kanwal MS. A Fresh Graduate's Guide to Software Development Tools and Technologies [Internet]. National University of Singapore; Unknown [cited 2016 Jan 1]. Available from: http://www.comp.nus.edu.sg/~seer/book/2e/Ch06.%20Scalability.pdf

9. Microservices architecture: advantages and drawbacks [Internet]. Cloud Academy Blog. [cited 2016 Jan 7]. Available from: http://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/

10. Arango M, Kaponig B. Ultra-scalable architectures for Telecommunications and Web 2.0 services. In: 13th International Conference on Intelligence in Next Generation Networks, 2009 ICIN 2009. 2009. p. 1–7.

11. Strickland J. How Server Virtualization Works [Internet]. HowStuffWorks. 2008 [cited 2016 Jan 8]. Available from: http://computer.howstuffworks.com/server-virtualization.htm

12. Hwang J, Zeng S, Wu FY, Wood T. A component-based performance comparison of four hypervisors. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013). 2013. p. 269–76.

13. Gomes Xavier M, Veiga Neves M, Fonticielha de Rose CA. A Performance Comparison of

Container-Based Virtualization Systems for MapReduce Clusters. In: 2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2014. p. 299–306.

14. Hess K. Hypervisor comparisons: It’s harder than you think [Internet]. ZDNet. [cited 2016 Jan 8]. Available from: http://www.zdnet.com/article/hypervisor-comparisons-its-harder-than-you-think/

15. Citrix. XenServer_6.5.0_Technical_FAQ [Internet]. Citrix; 2016 [cited 2016 Jan 8]. Available from: http://support.citrix.com/content/dam/supportWS/kA560000000Ts7qCAC/XenServer_6.5.0_Technical_FAQ.pdf

16. Microsoft. Hyper-V scalability in Windows Server 2012 and Windows Server 2012 R2 [Internet]. 2015 [cited 2016 Jan 8]. Available from: https://technet.microsoft.com/en-us/library/jj680093.aspx

17. VMwaere. vsphere-6.0-configuration-maximums [Internet]. VMware; 2015 [cited 2016 Jan 8]. Available from: http://www.vmware.com/pdf/vsphere6/r60/vsphere-60-configuration-maximums.pdf

18. RedHat. rhev-feature-guide [Internet]. RedHat; 2015 [cited 2016 Jan 8]. Available from: https://www.redhat.com/en/files/resources/en-rhev-feature-guide-12302817.pdf

19. Metcalf C. Docker Demo Webinar: FAQ [Internet]. Docker Blog. 2015 [cited 2016 Jan 8]. Available from: https://blog.docker.com/2015/08/docker-demo-faq/

20. Banerjee T. Understanding the key differences between LXC and Docker [Internet]. 2014 [cited 2016 Jan 8]. Available from: https://www.flockport.com/lxc-vs-docker/

21. Docker. Docker run reference [Internet]. 2016 [cited 2016 Jan 8]. Available from: https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources

22. Ubuntu. LXC - Ubuntu 14.04 Server Guide [Internet]. 2014 [cited 2016 Jan 8]. Available from: https://help.ubuntu.com/lts/serverguide/lxc.html

23. Ubuntu. LXD crushes KVM in density and speed [Internet]. Ubuntu Insights. 2015 [cited 2016 Jan 8]. Available from: https://insights.ubuntu.com/2015/05/18/lxd-crushes-kvm-in-density-and-speed/

24. celebrateubuntu. LXD vs KVM: OpenStack Vancouver 2015 [Internet]. OpenStack Vancouver; 2015 [cited 2016 Jan 8]. Available from: https://www.youtube.com/watch?v=wJpsZ6LS7dQ&feature=youtu.be&t=12m53s

25. Beserra D, Moreno ED, Takako Endo P, Barreto J, Sadok D, Fernandes S. Performance Analysis of LXC for HPC Environments. In: 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS). 2015. p. 358–63.

26. Lakew EB, Klein C, Hernandez-Rodriguez F, Elmroth E. Towards Faster Response Time Models for Vertical Elasticity. In: 2014 IEEE/ACM 7th International Conference on Utility

and Cloud Computing (UCC). 2014. p. 560–5.

27. Walters JP, Younge AJ, Kang DI, Yao KT, Kang M, Crago SP, et al. GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD). 2014. p. 636–43.

28. NVIDIA. Parallel Programming and Computing Platform | CUDA | NVIDIA|NVIDIA [Internet]. NVIDIA. 2015 [cited 2016 Jan 10]. Available from: https://www.nvidia.com/object/cuda_home_new.html

29. OpenCL. OpenCL - The open standard for parallel programming of heterogeneous systems [Internet]. khronos. 2015 [cited 2016 Jan 10]. Available from: https://www.khronos.org/opencl/

30. Nanz S, Furia CA. A Comparative Study of Programming Languages in Rosetta Code. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). 2015. p. 778–88.

31. LXC. go-ubuntu library [Internet]. GitHub. [cited 2016 Jan 10]. Available from: https://github.com/golang/go

32. golang. go-mysql library [Internet]. GitHub. [cited 2016 Jan 10]. Available from: https://github.com/go-sql-driver/mysql

33. golang. go-os library [Internet]. golang. 2016 [cited 2016 Jan 10]. Available from: https://golang.org/pkg/os/

34. mikespook. go-php library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/mikespook/goemphp

35. LXC. go-lxc library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/lxc/go-lxc

36. golang. go-tls library [Internet]. golang. 2016 [cited 2016 Jan 10]. Available from: https://golang.org/pkg/crypto/tls/

37. Ubuntu. java-ubuntu library [Internet]. launchpad. [cited 2016 Jan 10]. Available from: https://launchpad.net/ubuntu/+source/openjdk-7

38. mysql. java-mysql library [Internet]. dev.mysql. 2016 [cited 2016 Jan 10]. Available from: https://dev.mysql.com/downloads/connector/j/

39. oracle. java-os library [Internet]. oracle. 2016 [cited 2016 Jan 10]. Available from: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html

40. PHP/Java Bridge. Java-php library [Internet]. PHP/Java Bridge. 2016 [cited 2016 Jan 10]. Available from: http://php-java-bridge.sourceforge.net/pjb/

41. waseemh. java-lxc library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from:

https://github.com/waseemh/lxc-java

42. oracle. java-ssl library [Internet]. oracle. 2016 [cited 2016 Jan 10]. Available from: https://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLSocket.html

43. Golden D. perl-os library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~dagolden/Perl-OSType-1.009/lib/Perl/OSType.pm

44. Tsuchida T. perl-mysql library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/dist/Net-MySQL/

45. O'Brien M. perl-php library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~mob/PHP-0.15/PHP.pm

46. Schulz D. perl-lxc library [Internet]. CPAN. 2016 [cited 2016 Jan 10]. Available from: https://metacpan.org/pod/Linux::Virt::Plugin::LXC

47. Ulrich S. perl-ssl library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~sullr/IO-Socket-SSL-2.022/lib/IO/Socket/SSL.pod

48. Python. python2-ubuntu [Internet]. Python.org. [cited 2016 Jan 10]. Available from: https://www.python.org/download/releases/2.6.9/

49. Python. python-mysql library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi/MySQL-python/

50. Python. python-OS Library [Internet]. python. 2016 [cited 2016 Jan 10]. Available from: https://docs.python.org/2/library/os.html

51. Python. Python-php library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi?:action=browse&c=208

52. LXC. python2-lxc library [Internet]. GitHub. [cited 2016 Jan 10]. Available from: https://github.com/lxc/python2-lxc

53. Python. python-ssl library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi/ssl/1.16

8.2 - Bibliography

1. 18 KH for VS| M, Hardware 2012-- 07:00 GMT| Topic: Hypervisor comparisons: It’s harder than you think [Internet]. ZDNet. [cited 2016 Jan 8]. Available from: http://www.zdnet.com/article/hypervisor-comparisons-its-harder-than-you-think/

2. Arango M, Kaponig B. Ultra-scalable architectures for Telecommunications and Web 2.0 services. In: 13th International Conference on Intelligence in Next Generation Networks, 2009 ICIN 2009. 2009. p. 1–7.

3. AWS. AWS_Well-Architected_Framework.pdf [Internet]. Amazon Inc.; 2015 [cited 2015 Dec 31]. Available from: https://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

4. Badola V. Microservices architecture: advantages and drawbacks [Internet]. Cloud Academy Blog. 2015 [cited 2016 Jan 7]. Available from: http://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/

5. Banerjee T. Understanding the key differences between LXC and Docker [Internet]. 2014 [cited 2016 Jan 8]. Available from: https://www.flockport.com/lxc-vs-docker/

6. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Comput. 2014 Sep;1(3):81–4.

7. Beserra D, Moreno ED, Takako Endo P, Barreto J, Sadok D, Fernandes S. Performance Analysis of LXC for HPC Environments. In: 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS). 2015. p. 358–63.

8. Bustos-Jimenez J, Alonso R, Faundez C, Meric H. Boxing experience: Measuring QoS and QoE of multimedia streaming using NS3, LXC and VLC. In: 2014 IEEE 39th Conference on Local Computer Networks Workshops (LCN Workshops). 2014. p. 658–62.

9. celebrateubuntu. LXD vs KVM: OpenStack Vancouver 2015 [Internet]. OpenStack Vancouver; 2015 [cited 2016 Jan 8]. Available from: https://www.youtube.com/watch?v=wJpsZ6LS7dQ&feature=youtu.be&t=12m53s

10. Chistyakov A. A Software Architecture for Large Multi-simulation Experiments over Ad Hoc Networks Using NS-3 Discrete-Event Network Simulator. In: Modelling Symposium (EMS), 2014 European. 2014. p. 403–8.

11. Citrix. XenServer_6.5.0_Technical_FAQ [Internet]. Citrix; 2016 [cited 2016 Jan 8]. Available from: http://support.citrix.com/content/dam/supportWS/kA560000000Ts7qCAC/XenServer_6.5.0_Technical_FAQ.pdf

12. Correa CNA, de Lucena SC, de A.Leao Marques D, Rothenberg CE, Salvador MR. An experimental evaluation of lightweight virtualization for software-defined routing platform. In: 2012 IEEE Network Operations and Management Symposium (NOMS). 2012. p. 607–10.

13. Craver N. What it takes to run Stack Overflow [Internet]. nickcraver. 2013 [cited 2016 Jan 1].

Available from: http://nickcraver.com/blog/2013/11/22/what-it-takes-to-run-stack-overflow/

14. Docker. Docker run reference [Internet]. Docker. 2016 [cited 2016 Jan 8]. Available from: https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources

15. Dutta A, Gnawali O. Large-scale network protocol emulation on commodity cloud. In: 2014 IEEE Global Communications Conference (GLOBECOM). 2014. p. 1114–9.

16. Eramo V, Cianfrani A, Miucci E, Listanti M, Carletti D, Gentilini L. Virtualization and virtual router migration: Application and experimental validation. In: Teletraffic Congress (ITC), 2014 26th International. 2014. p. 1–6.

17. FlocPort. LXC Getting Started Guide Part II [Internet]. 2014 [cited 2016 Jan 8]. Available from: https://www.flockport.com/lxc-advanced-guide/

18. Gerard M. The Ultimate White Paper Template [Free Download] | Content Marketing Forum [Internet]. curata. 2015 [cited 2015 Dec 21]. Available from: http://www.curata.com/blog/the-ultimate-white-paper-template-free-download/

19. Ghulati S. Best Practices For Horizontal Application Scaling [Internet]. OpenShift Blog. 2013 [cited 2016 Jan 7]. Available from: https://blog.openshift.com/best-practices-for-horizontal-application-scaling/

20. golang book. Concurrency — An Introduction to Programming in Go | Go Resources [Internet]. golang-book. 2015 [cited 2016 Jan 10]. Available from: https://www.golang-book.com/books/intro/10

21. golang. go-mysql library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/go-sql-driver/mysql

22. golang. go-os library [Internet]. golang. 2016 [cited 2016 Jan 10]. Available from: https://golang.org/pkg/os/

23. golang. go-tls library [Internet]. golang. 2016 [cited 2016 Jan 10]. Available from: https://golang.org/pkg/crypto/tls/

24. Golden D. perl-os library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~dagolden/Perl-OSType-1.009/lib/Perl/OSType.pm

25. Gomes Xavier M, Veiga Neves M, Fonticielha de Rose CA. A Performance Comparison of Container-Based Virtualization Systems for MapReduce Clusters. In: 2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2014. p. 299–306.

26. Henderson C. Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications. 1st ed. San Francisco: O'Reilly Media; 2006.

27. Hoff T. StackOverflow Update: 560M Pageviews a Month, 25 Servers, and It's All About Performance - High Scalability - [Internet]. StackOverflow. 2014 [cited 2016 Jan 1]. Available from: http://highscalability.com/blog/2014/7/21/stackoverflow-update-560m-pageviews-a-

month-25-servers-and-i.html

28. Hwang J, Zeng S, Wu FY, Wood T. A component-based performance comparison of four hypervisors. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013). 2013. p. 269–76.

29. Khare A, Huang Y, Doan H, Kanwal MS. A Fresh Graduate's Guide to Software Development Tools and Technologies [Internet]. National University of Singapore; Unknown [cited 2016 Jan 1]. Available from: http://www.comp.nus.edu.sg/~seer/book/2e/Ch06.%20Scalability.pdf

30. Lakew EB, Klein C, Hernandez-Rodriguez F, Elmroth E. Towards Faster Response Time Models for Vertical Elasticity. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC). 2014. p. 560–5.

31. LXC. go-ubuntu library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/golang/go

32. LXC. python2-lxc library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/lxc/python2-lxc

33. LXC. go-lxc library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/lxc/go-lxc

34. Matsudaira K. Building Scalable Web Architecture and Distributed Systems [Internet]. Dr. Dobb's. 2012 [cited 2015 Dec 30]. Available from: http://www.drdobbs.com/web-development/building-scalable-web-architecture-and-d/240142422

35. Memari N, Samsudin KB, Hashim SJB. Towards virtual honeynet based on LXC virtualization. In: 2014 IEEE Region 10 Symposium. 2014. p. 496–501.

36. Metcalf C. Docker Demo Webinar: FAQ [Internet]. Docker Blog. 2015 [cited 2016 Jan 8]. Available from: https://blog.docker.com/2015/08/docker-demo-faq/

37. Meyerson J. The Go Programming Language. IEEE Softw. 2014;31(5):104–104.

38. Michael M, Moreira JE, Shiloach D, Wisniewski RW. Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: Parallel and Distributed Processing Symposium, 2007 IPDPS 2007 IEEE International. 2007. p. 1–8.

39. Microsoft. Hyper-V scalability in Windows Server 2012 and Windows Server 2012 R2 [Internet]. technet.microsoft. 2015 [cited 2016 Jan 8]. Available from: https://technet.microsoft.com/en-us/library/jj680093.aspx

40. mikespook. go-php library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/mikespook/goemphp

41. mysql. java-mysql library [Internet]. dev.mysql. 2016 [cited 2016 Jan 10]. Available from: https://dev.mysql.com/downloads/connector/j/

42. Nanz S, Furia CA. A Comparative Study of Programming Languages in Rosetta Code. In:

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). 2015. p. 778–88.

43. Niyaz Q, Sun W, Xu R, Alam M. Light VN: A Light-Weight Testbed for Network and Security Experiments. In: 2015 12th International Conference on Information Technology - New Generations (ITNG). 2015. p. 459–64.

44. NVIDIA. Parallel Programming and Computing Platform | CUDA | NVIDIA|NVIDIA [Internet]. NVIDIA. 2015 [cited 2016 Jan 10]. Available from: https://www.nvidia.com/object/cuda_home_new.html

45. O'Brien M. perl-php library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~mob/PHP-0.15/PHP.pm

46. OpenCL. OpenCL - The open standard for parallel programming of heterogeneous systems [Internet]. khronos. 2015 [cited 2016 Jan 10]. Available from: https://www.khronos.org/opencl/

47. oracle. Lesson: Concurrency (The Java™ Tutorials > Essential Classes) [Internet]. oracle. 2015 [cited 2016 Jan 10]. Available from: https://docs.oracle.com/javase/tutorial/essential/concurrency/

48. oracle. java-os library [Internet]. oracle. 2016 [cited 2016 Jan 10]. Available from: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html

49. oracle. java-ssl library [Internet]. oracle. 2016 [cited 2016 Jan 10]. Available from: https://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLSocket.html

50. Padnick J. A Comprehensive Guide to Building a Scalable Web App on Amazon Web Services - Part 1 [Internet]. airpair. 2015 [cited 2015 Dec 31]. Available from: https://www.airpair.com/aws/posts/building-a-scalable-web-app-on-amazon-web-services-p1

51. Perl. Perl- ubuntu library [Internet]. PERL. 2016 [cited 2016 Jan 10]. Available from: https://www.perl.org/get.html

52. perl6. Concurrency [Internet]. perl6. 205AD [cited 2016 Jan 10]. Available from: http://doc.perl6.org/language/concurrency

53. PHP/Java Bridge. Java-php library [Internet]. PHP/Java Bridge. 2016 [cited 2016 Jan 10]. Available from: http://php-java-bridge.sourceforge.net/pjb/

54. Podzimek A, Bulej L, Chen LY, Binder W, Tuma P. Analyzing the Impact of CPU Pinning and Partial CPU Loads on Performance and Energy Efficiency. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). 2015. p. 1–10.

55. python. Concurrency - Python Wiki [Internet]. python. 2015 [cited 2016 Jan 10]. Available from: https://wiki.python.org/moin/Concurrency

56. Python. python2-ubuntu [Internet]. Python.org. 2016 [cited 2016 Jan 10]. Available from: https://www.python.org/download/releases/2.6.9/

57. Python. python-mysql library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi/MySQL-python/

58. Python. python-OS Library [Internet]. python. 2016 [cited 2016 Jan 10]. Available from: https://docs.python.org/2/library/os.html

59. Python. Python-php library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi?:action=browse&c=208

60. Python. python-ssl library [Internet]. Python. 2016 [cited 2016 Jan 10]. Available from: https://pypi.python.org/pypi/ssl/1.16

61. Qi Z, Dong H, Sun W, Dong Y, Guan H. Multi-Granularity Memory Mirroring via Binary Translation in Cloud Environments. IEEE Trans Netw Serv Manag. 2014 Mar;11(1):36–45.

62. RedHat. Virtualization limits for Red Hat Enterprise Virtualization - Red Hat Customer Portal [Internet]. redhat. 2014 [cited 2016 Jan 9]. Available from: https://access.redhat.com/articles/906543

63. RedHat. rhev-feature-guide [Internet]. RedHat; 2015 [cited 2016 Jan 8]. Available from: https://www.redhat.com/en/files/resources/en-rhev-feature-guide-12302817.pdf

64. Sarzyniec L, Buchert T, Jeanvoine E, Nussbaum L. Design and Evaluation of a Virtual Experimental Environment for Distributed Systems. In: 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2013. p. 172–9.

65. Schulz D. perl-lxc library [Internet]. CPAN. 2016 [cited 2016 Jan 10]. Available from: https://metacpan.org/pod/Linux::Virt::Plugin::LXC

66. Selimi M, Freitag F, Pueyo Centelles R, Moll A. Distributed Storage and Service Discovery for Heterogeneous Community Network Clouds. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC). 2014. p. 204–12.

67. Silva N. UML Diagram Types With Examples for Each Type of UML Diagrams [Internet]. Creately Blog | Diagramming Articles and Tips on How to Draw Diagrams. 2012 [cited 2015 Dec 21]. Available from: http://creately.com/blog/diagrams/uml-diagram-types-examples/

68. Smith B. Understanding Horizontal and Vertical Scaling [Internet]. blakesmith. 2012 [cited 2016 Jan 1]. Available from: http://blakesmith.me/2012/12/08/understanding-horizontal-and-vertical-scaling.html

69. Strickland J. How Server Virtualization Works [Internet]. HowStuffWorks. 2008 [cited 2016 Jan 8]. Available from: http://computer.howstuffworks.com/server-virtualization.htm

70. Tsuchida T. perl-mysql library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/dist/Net-MySQL/

71. Ubuntu. LXC - Ubuntu 14.04 Server Guide [Internet]. ubuntu. 2014 [cited 2016 Jan 8]. Available from: https://help.ubuntu.com/lts/serverguide/lxc.html

72. Ubuntu. LXD crushes KVM in density and speed [Internet]. Ubuntu Insights. 2015 [cited 2016 Jan 8]. Available from: https://insights.ubuntu.com/2015/05/18/lxd-crushes-kvm-in-density-and-speed/

73. Ubuntu. java-ubuntu library [Internet]. launchpad. 2016 [cited 2016 Jan 10]. Available from: https://launchpad.net/ubuntu/+source/openjdk-7

74. Ulrich S. perl-ssl library [Internet]. cpan. 2016 [cited 2016 Jan 10]. Available from: http://search.cpan.org/~sullr/IO-Socket-SSL-2.022/lib/IO/Socket/SSL.pod

75. Vikash S. How to setup high density VPS hosting using LXC (Linux Containers) and LXD [Internet]. Bobcares. 2015 [cited 2016 Jan 8]. Available from: https://bobcares.com/blog/how-to-setup-high-density-vps-hosting-using-lxc-linux-containers-and-lxd/

76. VMwaere. vsphere-6.0-configuration-maximums [Internet]. VMware; 2015 [cited 2016 Jan 8]. Available from: http://www.vmware.com/pdf/vsphere6/r60/vsphere-60-configuration-maximums.pdf

77. Walters JP, Younge AJ, Kang DI, Yao KT, Kang M, Crago SP, et al. GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD). 2014. p. 636–43.

78. waseemh. java-lxc library [Internet]. GitHub. 2016 [cited 2016 Jan 10]. Available from: https://github.com/waseemh/lxc-java

79. Xavier MG, De Oliveira IC, Rossi FD, Dos Passos RD, Matteussi KJ, De Rose CAF. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In: 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2015. p. 253–60.

80. Xavier MG, Neves MV, Rossi FD, Ferreto TC, Lange T, De Rose CAF. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In: 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2013. p. 233–40.

81. Zamaere B, Da L, Kullberg E. On the design and implementation of a virtualized residential gateway. In: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet). 2012. p. 3349–52.

82. Zhang Z, Han J, Li B, Zhou W, Meng D. Lynn: A Multi-dimensional Dynamic Resource Management System for Distributed Applications in Clouds. In: 2013 International Conference on Cloud and Service Computing (CSC). 2013. p. 84–91.

8.3 - Initial Project Proposal

1. **Name:** David Monaghan
2. **Contact Details**
    i. david.monaghan@mycit.ie
    ii. 086-862-1584
    iii. 18 Cedar Grove, Bishopstown, Cork
3. **Project Title:** Website Hosting System using LXC (Linux Containers)
4. **Client Company and Contact Person :** N/A
5. **Project Background:** Current web-hosting platforms use full virtualisation technologies such as KVM to host independent servers. This entails a significant overhead in processing and storage resources. LXC is a new type of virtualisation technology that removes much of that overhead. Using LXC there is massive opportunity to drastically improve the performance of existing web-hosting both in terms of storage and computation costs.
6. **Overall Project Goal:** The project goal is to create a platform that will allow a person to quickly create and setup a Linux server, an Apache server and a CMS (Content Management System). The system should also be able to create backups, snapshots and the ability to restore a container to that backup or snapshot. To manage this functionality there will be a database driven website that will allow a user to create, manage and destroy their containers, servers and websites. The system should also be able to recover itself from a failure such as corruption of a container or a users site going down. The project will also be designed to be as secure as possible.
7. **Objectives:**
    i. A database driven website
    ii. Programmed services to manage the customers containers
    iii. System secured correctly
    iv. System capable of being recovered from failure
8. **Development Tools:**
    i. Frontend: HTML/ CSS/ PHP/ MySQL
    ii. Backend: Go and BASH
    iii. Version Control: Github

9. Learning Outcomes

i. LXC

ii. Linux server-side development

iii. Linux security

iv. SDN (Software Defined Networking)

v. MySQL

vi. Full stack web-development

10. Software/Hardware Requirements:

i. Ubuntu 14.04 Laptop/ Desktop (I have obtained these already)

ii. Ubuntu 14.04 Sever Edition (Free Open Source Download)

iii. LXC (Free Open Source Download)

iv. Go (Free Open Source Download)

v. MySQL (Free Open Source Download)

vi. BASH (Free Open Source Download)

vii. HTML/CSS/PHP (Free Open Source Download)