



Raspberry Pi Cloud: Building a Cloud

April 2016

Dissertation submitted in partial fulfilment for the degree of *BSc (Hons) Software Engineering*

Student Name: Heather McWha

Student Number: 441242

Project Supervisor: Dr Posco Tso

Department of Computing Science and Mathematics

Liverpool John Moore's University

Abstract

In recent years, there has been a large shift in the world of computing with a large number of companies moving from in-house data centers for software, storage and processing requirements to using offsite Cloud data centers and adapting their business' to the models available, such as pay-as-you-go licencing for applications. This has led to a reduction in operating costs, along with an increase in flexibility, making this approach popular with some of the world's largest companies such as Microsoft, Amazon and Google.

A second emerging technology is that of low cost Single Board Computers, with the Raspberry Pi being at the forefront of this in both cost and size. Due to this reduction in costs and power requirements, it has become commonplace for developers to bring these two technologies together. This is done by building utilising a large number of these devices in order to build high-powered cluster at a low cost, rather than having to purchase a "supercomputer" for their processing requirements. This is especially evident in the academic sector, as this allows for testing, development, deployment, scalability and even teaching of Cloud infrastructure and principals at a low cost.

However, due to these Single Board Computers typically using an ARM architecture, there is no readily available orchestration tools for this type of Cloud infrastructure as the majority of developers to not release versions of their software tailored to this instruction set.

Therefore, this project focuses on creating a set of easy to use Cloud infrastructure and management tools in order to show developers that there is a place for ARM devices in the ever changing Cloud environment. With a 5-node Raspberry Pi 2 cluster, this project implements such tools in addition to extending into the live migration of running containers.

Acknowledgements

Most importantly, enormous gratitude is due to my supervisor, Posco Tso, who knew that at the outset of this project, I had extremely little experience with Linux distributions, had zero understanding of Docker and containers at all. I went to him wanting a challenge, and that is precisely what he gave me. To that I am most appreciative, the opportunities that I have been offered since embarking on this project in September are things I could never have envisioned myself being offered, and begged in some cases to take up on. These opportunities would not have been possible without the challenges Posco had set, even pushing for the impossible of container live migration, yet we were so close!

Thank you.

To the University of Glasgow Raspberry Pi Cloud team,

I am much obliged to the team for permitting me to hijack the Raspberry Pi Cloud word press website and publish my work from this project for the world to see, not overlooking the astounding work they had beforehand done to be my starting point for this project and is consistently utilized all throughout and referenced back to when I reached a dead-end.

Thank you.

To my family, for knowing to stay clear of me in April. Thank you.

To Ashley,

I am forever indebted to you through the support you have given me through this entire project, from the very beginning to the very end.

Dealing with my many child-like tantrums of crying insanely on the kitchen floor at whatever point I broke a Pi with a simple command then coming back and saying “fixed it” when I decided to give up (numerous, multiple occasions!)

Thank you.

P.S However, I will never forgive you for writing off my car a couple of days before the deadline!!

Table of Contents

1	Introduction.....	1
1.1	Project Management	2
1.2	Overview of Dissertation	3
2	Literature Survey	4
2.1	Cloud Computing.....	4
2.2	Virtualization	6
2.3	Raspberry Pi.....	9
2.4	Computer Clusters	11
2.5	ARM Architecture	11
2.6	Linux	11
2.7	Docker.....	12
2.8	Kubernetes	16
2.9	Live Migration	17
2.10	Case Studies.....	18
3	Analysis.....	19
4	Design and Development.....	23
4.1	Cluster Considerations.....	23
4.2	Hardware Setup.....	23
4.3	Installing Arch Linux ARM.....	23
4.4	Installing Docker.....	25
4.5	Configuring Docker	26
4.6	Building Docker Images	28
4.7	Running Kubernetes.....	29
4.8	Configuring Master and Worker Services for Kubernetes	30
4.9	Automating Cluster Deployment	31
4.10	Kubernetes Dashboard.....	31
4.11	Installing Kubernetes Dashboard Prerequisites	32
4.12	Building Go.....	32
4.13	Upgrading GO using Bootstrap and Source.....	33
4.14	Building the Dashboard	34
4.15	Running Dashboard using Screen.....	36
4.16	Dashboard as a Docker Image	36
4.17	Criu and P. Haul.....	37
4.18	Implementing Dashboard Resource Monitoring.....	38
5	Evaluation	40
5.1	Contributions	41
6	Discussion and Conclusion.....	44
7	References and Bibliography.....	45
8	Appendices.....	50
8.1	Project Management	50
8.2	Code and Scripts	55

List of Illustrations

Figure 1: Cloud Computing Model (NIST 2011)	4
Figure 2: Guest Operating System (Virtuatopia)	7
Figure 3: Type 1 and Type 2 Hypervisors (IBM 2011)	8
Figure 4: Raspberry Pi 2 Model B (PC Tech Mag 2016)	9
Figure 5: Simple Beowulf Cluster Diagram (Citizendium)	11
Figure 6: Timeline of Docker Success	13
Figure 7: Virtual Machines vs Docker Comparison (Docker 2016)	14
Figure 8: Docker Containers (Docker 2016)	15
Figure 9: Cluster Analysis on Kube-UI (Kube-UI 2016)	20
Figure 10: Kubernetes Dashboard - Running Containers	21
Figure 11: Kubernetes Dashboard - Pod Details	21
Figure 12: cAdvisor User-Interface	22
Figure 13: Docker Hub	29
Figure 14: Go Version	33
Figure 15: Hello World	33
Figure 16: Go Folder	34
Figure 17: Dashboard - Gulp Serve	35
Figure 18: Kubernetes Dashboard	35
Figure 19: Docker Images	36
Figure 20: Kubernetes Dashboard: Heapster Resource Usage	39
Figure 21: Kubernetes Dashboard - Version 1.0.1	40
Figure 22: Dashboard GitHub Collaboration	42
Figure 23: Dashboard Blog Reference	43
Figure 24: Dashboard source-code change	43

List of Tables

Table 1: Raspberry Pi Model Comparison..... 10

Table 2: Hardware Specification Summary 23

1 Introduction

With the release of Docker and various cluster management tools, constructing a miniature Cloud-based Data Centre is extremely popular as it allows the user to explore the impact of virtualization and containers without the cost of high-end components.

However, for the ARM community, this is no simple task. As the majority of Cloud solutions are dominated by the use of i86 and x64 architectures, developed by Intel and AMD, there are limited cluster orchestration tools immediately available. Therefore, to construct and run a cloud solution on an ARM infrastructure requires an extensive knowledge of programming.

One of the most notable projects within the ARM community concentrating on a cloud solution is The Glasgow Raspberry Pi project. This project is based on a “scale-model” of a Data Centre and is comprised of 56 Raspberry Pi devices. This was later called the “Pi-Cloud”. The Pi-Cloud emulated all layers of a cloud stack, from resource virtualization to network behaviour, thus providing a fully-functional platform for cloud computing research and education.

The objective of this project is to extend upon the ground-breaking work previously done by the University of Glasgow’s Raspberry Pi Cloud team. This would include the simplification of previous methods used to implement the software requirements of a cluster using the Raspberry Pi platform, along with develop and deploy cluster management tools.

The initial scope of this project is to implement a cluster management tool and publish the solution to make it readily available for the ARMv7 community. At the outset of the project, none of the current cloud management tools had an ARMv7 compatible release. However, as there were several community projects already dedicated to porting cluster management software to the ARMv7 platform, it was apparent that there was a large need for this type of development. In order to do this, manually reconfiguring and building from the binary sources would be required. However, this was reliant upon the software being open-source so that these changes were possible. Therefore, one aim of the project was to test a variety of tools to establish which tools could be migrated to work on a ARMv7 platform, using a Raspberry Pi cluster.

This project excluded the physical building of a Raspberry Pi Cluster as this was already achieved previously by the Glasgow Raspberry Pi team, along with excluding various services required such as DHCP and operating system compilation. This was done by using readily available networking hubs and readily available distributions of the Linux operating systems. This project also had to exclude the use of any closed-source cloud platforms such as Microsoft Azure.

1.1 Project Management

Due to the fast-changing, rolling-releases of software used within this project, there was no traditional way of managing the project. This is shown in the monthly meeting reports submitted to the project supervisor. [Shown in Appendices 8.1]

A Gantt chart was created during the initial project specification. However, as time went on in the project, the milestones changed due to the developments, both the initial and the further development milestones are outlined below:

Initial Project Milestones:

- Set up a single master and slave cluster using 2x Raspberry Pi 2's.
- Configure and deploy a management platform to the master Pi.
- Scale the cluster from two to five units for testing.
- Measure the performance of the Pi Cloud.

Further Developments:

- Install Arch Linux ARM and Docker to each Pi.
- Research and understanding of ARM architecture.
- Research and understand of Docker.
- Install Kubernetes onto the Raspberry Pi 2.
- Hack/Play with Kubernetes.
- Kubernetes Dashboard development.
- Experiment with Containers and LXC
- Research on Orchestration tools: Ansible, Shipyard, Kubernetes.
- Feasible study of container/Docker live migration.
- Research P. Haul.
- Research CRIU.
- Research ETCD stability as it kept failing when running for long periods of time.
- Create custom kernel for Arch Linux to enable compatibility with CRIU.
- Cluster increased to 5.
- Two blog posts posted on the University of Glasgow's Raspberry Pi Cloud blog.
- Await release of CRIU 2.0

1.2 Overview of Dissertation

Chapter 1: Introduction

The introduction includes the background of the project, with a brief description of the work undertaken, the scope of the project, and any limits that may have been imposed.

Chapter 2: Literature Survey

Presented within this chapter is the series of research required in order to complete the project. Starting with the basics of Cloud computing and ending with the main aspects of the project such as the software implemented of Docker and Kubernetes. This chapter also incorporates various case-studies in which the overall software is used in real-world situations.

Chapter 3: Analysis

Within this chapter, the importance of the work and main work done within the project is outlined as well as any problems encountered, with the solutions to these problems. Also, the decision to extend the project further by incorporating a never tried before feature of container-based checkpoint and restore.

Chapter 4: Design and Development

This chapter shows the design and development of the Raspberry Pi cluster, as well as how various software had been implemented on the ARMv7 architecture step-by-step. It further provides an explanation of the decision to further extend the project with container live migration tools.

Chapter 5: Contributions

Within this chapter, the contributions are shown from the work done in this project used within the ARM community, as well as the developments made in collaboration with Google.

Chapter 6: Discussion and Conclusion

This chapter summarises and discusses the project as a whole. It further signposts possible future developments for the overall project.

Chapter 7: References and Bibliography

This chapter is a list of any sources, or literature used whilst completing the project.

Chapter 8: Appendices

The appendices provide all the relevant source-code or scripts used within the project. There are referenced throughout the report.

2 Literature Survey

In order to better understand the project, research was required across many areas. This chapter of this report concentrates on the main research areas taken from a variety of sources.

2.1 Cloud Computing

The “Cloud” is a metaphor for the “Internet”. It implies data, programs or applications which are stored on servers connected to the internet.

The National Institute of Standards and Technology (NIST) has released an official definition and model of “Cloud Computing”, this definition and model is considered as the industry standard. Figure 1, (NIST 2011) captures the Cloud Computing Model.

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models” - (NIST 2011)

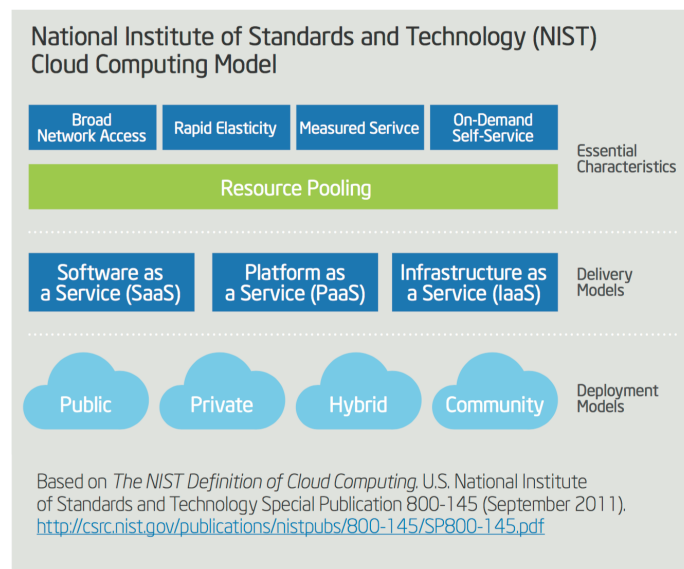


Figure 1: Cloud Computing Model (NIST 2011)

2.1.1 Characteristics

- **On-demand self-service:** Give users a basic and adaptable approach to provision networks, storage, software and computing power. It permits users to expand their resources over time and should be possible to be performed without human interaction, dependant on the cloud provider’s architecture and available resources.
- **Broad network access:** Gives users access to the cloud resources through numerous gadgets, for example, mobile phones, laptops and PDAs.
- **Resource pooling:** The provider gives users temporary and adaptable services, which can be conformed to suit each user’s needs, with no evident changes for the user. E.g. Storage, processing, memory, network bandwidth, and virtual machines.
- **Rapid elasticity:** The resources can be provisioned rapidly and is known for its flexibility with resources being used proportionally dependant on the user’s needs. The cloud resources appear to the user as boundless and a user can obtain as much resource as they need without any unnecessary delay.

- **Measured service:** Cloud systems control the usage and resources by monitoring the capability such as the storage and processing power. The user and provider can control, observe and respond to the resource usage.

2.1.2 Deployment Models

The deployment models indicate how the resources used within the cloud are being shared. The four types of cloud deployment models are; private cloud, community cloud, public cloud, and hybrid cloud. Each one of the models can impact the connected scalability, reliability, security and cost.

- **Private Cloud:** The infrastructure is confined to an individual or single company, hence the name “Private”. With a private cloud, the owner has full access and control over their cloud infrastructure.
- **Community Cloud:** The community cloud model is a cloud that is shared between a number of owners who have related concerns i.e. security or compliance. It can be managed internally, or by a third-party.
- **Public Cloud:** The public cloud is built for universal access by the general public. The public cloud owner, such as a government or business, can build, manage and operate a public cloud service using their own business models.
- **Hybrid Cloud:** The hybrid cloud model is a mixture of two or more of the cloud deployment models. The models remain different entities, but are joined by regulated or private technology which allows data and application transferability i.e. cloud bursting for load balancing between clouds.

2.1.3 Service Models

Users or applications can interact with a Cloud in many ways through services. There are three models of services that have appeared with the rapid development of Cloud computing.

- **Software as a Service (SaaS):** A business provides their software or application over the internet through a web browser. This is known as a service on demand and is typically provided through a subscription or pay-as-you-go business model. More recently, there has been an increase of free services as companies can generate revenue from streaming media or displaying advertisements. Due to the ease of access at low or no cost, Software-as-a-Service is quickly gaining popularity. However, it is crucial that users understand what SaaS is along with its advantages and disadvantages.

There are three main advantages of SaaS. Primarily, SaaS applications are simple to access as users only need an internet connection and a web browser to use the service. Furthermore, SaaS is cost efficient as any hardware requirements are taken care of by the data centre in which the SaaS resides. Finally, scalability is simplified as users can add or remove user licences as required. A monthly subscription is typically cheaper than an upfront proprietary licence and as such would work out cost effective when using temporary staff for example.

However, the main disadvantage of SaaS is the perception of security issues. This means that users who are new to the Cloud may not feel secure or comfortable with storing their data in the cloud, same for businesses.

- **Platform as a Service (PaaS):** The Platform as a Service model is similar to SaaS however it is more beneficial for developers. PaaS model removes the developer’s requirement to buy and manage hardware as well as installing and managing operating systems and/or database software. These required computing resources occupy virtual space in the Cloud instead of physical space in a data centre. Furthermore, the resources can be scaled appropriately based on the developer’s application or software demand. The business will only have to pay for what

resources they use on a pay-per-use service. PaaS also eliminates the need for developers to be concerned about hardware and as a result, their software can be deployed quicker, enabling greater productivity.

The basic characteristics of PaaS are; Within the integrated development environment, software can be developed, tested, deployed as well as hosted and maintained. The web-based user interface allows for users to create, modify, test and deploy different user-interface scenarios. Lastly, it should provide built-in scalability of the deployed software or application, including load balancing and failover.

PaaS has many advantages, the first being the financial advantage of companies not having to purchase, upgrade or maintain their IT equipment and as a result allows the business flexibility to grow and downsize dependant on their needs. Secondly, PaaS is internet based, as such it does not depend on location within the business and can be equally accessed throughout all business areas.

However, such as with SaaS and the Cloud in general, there is a perception of data security issues that businesses face, especially with having applications hosted via a third party.

- **Infrastructure as a Service (IaaS):** IaaS is where a third-party will host all components that are needed for infrastructure such as software and hardware, on behalf of the user. Not only do IaaS providers host the user's components, they will also host the applications and handle tasks on their behalf such as; system maintenance. The infrastructures can be adjusted-on-demand and is suitable for companies' projects that are temporary, experimental or subject to change unexpectedly. Businesses pay for the service as a pay-per-use basis from by the hour, to the week or month. Dependant on the on the provider they can also charge for the amount of virtual space used. A wide range of both large and small-scale applications are hosted by the Cloud. Users are able to deploy a "virtually limitless" amount of applications to the cloud and the majority of companies are in the process of moving their key applications to the cloud, instead of using internal data centres which are costly.

2.2 Virtualization

Cloud computing and virtualization come hand in hand. Virtualization makes the delivery of services much easier by providing a platform to scale complicated IT resources, this is what makes cloud computing a cost effective option.

Large software companies such as Google, Microsoft and Amazon all use virtualization to create high-performance cloud layers of their infrastructure. However, this technology is not only utilized only in data centres, but utilized on personal computers by millions of people every day.

Virtualization has been utilized in data centres for quite some time as a successful IT technique for reducing the number of physical servers required to cope with the increase of user demand. Utilized all the more comprehensively to group infrastructure resources.

To improve a cloud environments flexibility and agility, virtualization can be used to give the required building blocks. Today, server virtualization is the most focused on technology, with the virtualization of storage and networks becoming industry standard.

However, virtualization is not the same as cloud computing. Virtualization uses physical computer resources such as processors, memory, storage and network connections in order to create and scale virtualised machines dependent on the cloud's demands at the time. The cloud has control and decides which of these virtual resources are shared, delivered and used. Virtualization provides a quick-way of scaling resources that non-virtualization cannot achieve.

Essentially, virtualization contrasts from cloud computing since virtualization is software that controls hardware, whilst cloud computing alludes to a service that outcomes from that control.

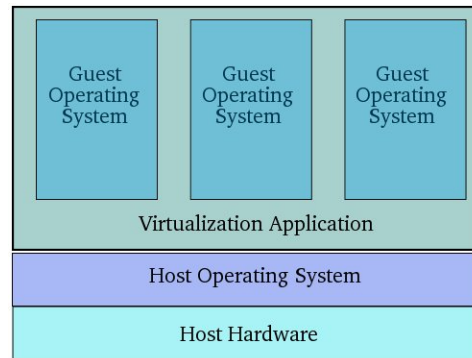


Figure 2: Guest Operating System (Virtuatopia)

2.2.1 Definition

Virtualization utilizes a PC's hardware and software to make a virtual version of a resource or device. For instance, partitioning a hard drive, as you are taking one physical hard drive and dividing it to make two separate hard drives.

2.2.2 Types of Virtualization

- **Server:** Dividing one physical server into littler virtual servers, which thusly makes one server appear as many. The littler virtual servers can run either the same or various operating systems. The upsides of this is that it can increase CPU utilization, less power consumption and supports multiple operating systems.
- **Desktop:** Users can change between various operating systems inside the same PC. The operating system which occupies a virtualized environment is known as the guest operating system, Figure 2 shows the infrastructure of a guest OS (Virtuatopia). This is leverage for software developers, technical support teams and testers. Desktop virtualization gives a superb answer for developers, application testers, and help desk support personnel who must support multiple operating systems. As opposed to having multiple desktop systems around their work area, with each system running a particular operating system, the user can rather utilize a solitary desktop PC with various virtual operating systems.
- **Virtual Networks:** A user can connect to a company network and their resources through the internet securely, without the need to be physically present on the company's premises. These virtual networks are also known as Virtual Private Networks (VPN's).
- **Virtual Storage:** Provides the user with scalable and redundant physical storage through disk drives or file systems, or a database interface.

2.2.3 Virtualization Characteristics

Virtualization can be applied to most things such as; memory, networks and operating systems just to name a few.

- **Partitioning:** Used to assist various applications and OS's in a solidarity physical system.
- **Isolation:** Each machine is protected from viruses, bugs and crashes as the virtual machines are independent of each other, regardless of sharing the same physical hardware.
- **Encapsulation:** Prevents applications from interfering with other applications. Encapsulation makes it easier for a virtual machine to be identified and shown to other applications.

2.2.4 Virtual Machines

Virtual machines are operating systems or applications that are installed on software which copies dedicated hardware. Users would have the exact same experience on a virtual machine as what they would if they had dedicated hardware.

Examples of these are; Oracle VirtualBox, Vmware, Parallels, QEMU, and Windows Virtual PC.

2.2.5 Hypervisors

Hypervisors can be software, hardware or firmware which permits the user to run different virtual machines on single hardware. Within each virtual machine, or OS, the user will be able to run its own programs as if it is using the host machines processor, memory, and network amongst other things.

However, rather than permitting direct access to the physical resources, the hypervisor is simply assigning access to the resources for the virtual machines at the time required. Essentially, a hypervisor allows the user to have several virtual machines all running on one solitary PC. This is done by allocating resources such as the hosts systems processor, memory and other assets to what each operating system requires.

- **Type 1:** The Type 1 hypervisor is more commonly known as a “native” or “bare metal” hypervisor. As the hypervisor works as a “slim layer”, which its purpose is to open hardware resources to VM, it provides a better execution and adaptability, this in turn lowers the overheads which are needed to run the hypervisor itself. Examples of Type 1 hypervisors are; VMware and Citrix or Xen Server.
- **Type 2:** The Type 2 hypervisor is otherwise known as a “hosted” hypervisor. These types of hypervisors run on a host OS that provide virtualization services such as; Input/output device backing. The Type 2 hypervisor is reliant on the host OS for any operations. Examples of Type 2 hypervisors are; VMware Workstation and Oracle Virtual Box.

Ordinarily, a Type 1 hypervisor is more effective than a Type 2 hypervisor, yet from multiple points of view they both give a similar level of usefulness since they are both capable of running the same VMs. Truth be told, you can more often than not move a VM from a host server running a Type 1 hypervisor to one running a Type 2 hypervisor and the other way around. A conversion might be required, however the procedure works. As Type 1 runs straightforwardly on the hardware, it can bolster hardware virtualization. Though, Type 2 hypervisors keep running as an application on top of an operating system, they can perform software virtualization.

Figure 3 demonstrates the contrasts between the Type 1 and Type 2 hypervisors (IBM 2011).

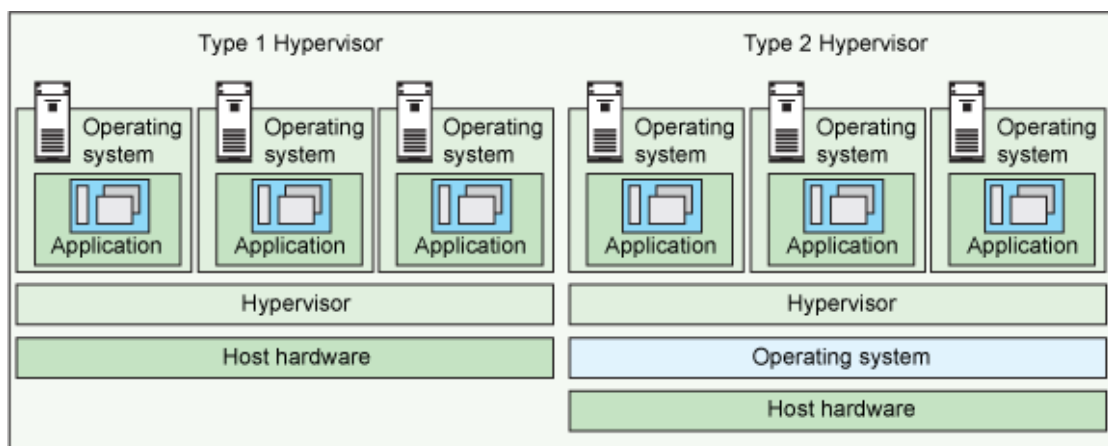


Figure 3: Type 1 and Type 2 Hypervisors (IBM 2011)

2.2.6 Linux Containers

Linux containers, otherwise called LXC, is an alternate type of virtualization. It permits the running of a Linux system inside another Linux system. A container is a gathering of processes on a Linux host, set up together in a separated environment. Basically they are virtual machines without the requirement of a hypervisor.

At the end of the day, LXC gives operating-system-level virtualization through a virtual environment that has its own particular process and network space, rather than making an undeniable virtual machine. By exploiting different features including chroots, cgroups, namespaces and SELinux profiles, the Linux container have application process which will oversee constrained resource types, and protecting them from achieving their own file systems.

The infrastructure of Linux containers is an on-going development within the cloud technology due to fast and lightweight process virtualization. The purpose is to give users an environment that is extremely similar to a standard Linux distribution. Linux containers are more lightweight than VM's, thus meaning densities are higher with containers than the VMs which are on the same host.

2.3 Raspberry Pi

In 2012, the Raspberry Pi was created by the Raspberry Pi Foundation. The origin behind the Pi was to promote and teach the basics of Computer Science across the UK.

The Raspberry Pi is a “credit-card size computer”, this was an idea based on the BBC Micro in 1981. The aim of the project was to make a cheap, low cost Single Board Computer (SBC) device which could be used in education to advance programming skills and the understanding of hardware.

The Raspberry Pi quickly became a must-have gadget due to its size and price and was fast sold worldwide and used for ideas or projects that a basic microcontroller was incapable of, for instance, Arduino device.

Whilst the Raspberry Pi is low in computing power compared to basic laptops and desktops, it is still a device which is very capable to run a large number of operating systems and software packages for a low initial cost and low ongoing costs due to its low power requirements.

Furthermore, everything on the Pi board is “open hardware”, the only exception being the Broadcom System on a Chip (SoC), which is the main chip and runs the majority of the primary components of the Pi such as the CPU and graphics.

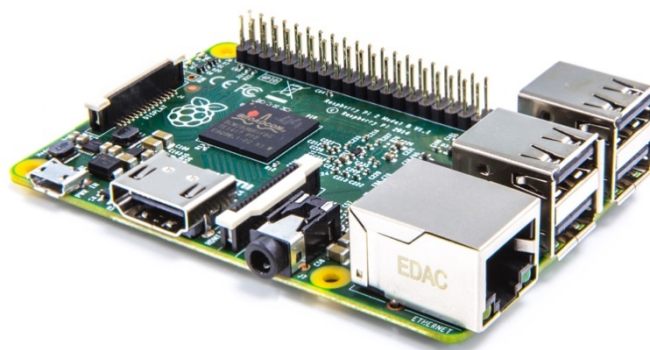


Figure 4: Raspberry Pi 2 Model B (PC Tech Mag 2016)

2.3.1 Model Comparison Table

Since the first release in 2012, the Raspberry Pi foundation have released a number of Raspberry Pi models differing in performance and resources available. Table 1 shows a comparison of available Raspberry Pi models, with the exception of the recently released Raspberry Pi 3 in February 2016.

Table 1: Raspberry Pi Model Comparison

	RP1 Model A	RP1 Model A+	RP1 Model B	RP1 Model B+	RP2 Model B
Release Date:	Feb-12	Nov-14	N/A	Jul-14	Feb-15
CPU	700 MHz single-core ARMv6	700 MHz single-core ARMv6	700 MHz single-core ARMv6	700 MHz single-core ARMv6	900 MHz quad-core ARMv7
Memory	256mb	256mb	512mb	512mb	1GB
USB Ports	1	1	2	4	4
Video Output	HDMI	HDMI	HDMI	HDMI	HDMI
GPIO	8	8	8	17	17
Power Ratings	300mA (1.5w)	200mA (1w)	700mA (3.5w)	600mA (3w)	800mA (4w)
Power Source	5V Micro USB	5V Micro USB	5V Micro USB	5V Micro USB	5V Micro USB
Size	85.60 mm × 56.5 mm	65 mm × 56.5 mm	85.60 mm × 56.5 mm	85.60 mm × 56.5 mm	85.60 mm × 56.5 mm

2.3.2 Operating Systems

The officially supported operating systems for the Raspberry Pi's is known as Raspbian. This is a custom built Linux operating system based on Debian. There are a large number of third party operating systems available for the Raspberry Pi from Ubuntu to Windows 10 IoT. One of the largest supported third-party operating systems is Arch Linux Arm. Based on Arch Linux, a Linux operating system which requires very minimal hardware overheads, this is a popular choice for developers who want to utilise the Raspberry Pi's minimal hardware without being subject to large overhead requirements of the OS.

2.3.3 Projects/Uses

The Raspberry Pi has been such a success that a company called PA Consulting have started an awards ceremony. Their second annual Raspberry Pi ceremony commenced in late 2015. The aim of the awards ceremony is to allow school children and college students to “put their programming skills to the test” and utilize the Raspberry Pi to “make the world a better place”.

The winning projects from the 2014/15 competition, which includes projects from youngsters as youthful as eight include;

- A robot dog that urges children to work out called “FitDog”.
- Automatic prescription dispenser that works with QR codes called “Pi-Scripton”.
- A system that tracks the users eye movements to control their PC called “Revolution Pi”.

It is not simply hobbyists who have exploited the Raspberry Pi's abilities;

- GCHQ made a Raspberry Pi cluster called “Bramble”, which comprises of “blocks” of 8 Pi's, which GCHQ named “OctaPi”. The OctaPi's can be standalone or associated together to form a larger cluster, utilizing 64 Raspberry Pi's and 2 head nodes makes it a total of 66 Raspberry Pi's utilized.
- The University of Southampton made a Raspberry Pi cluster called “Iridis-Pi”, which comprises of 64 Raspberry Pi Model B nodes interconnected with 100Mbps Ethernet joins.
- The University of Glasgow made a Raspberry Pi cloud cluster, connecting 56 Raspberry Pi boards in racks produced using Lego, making a working model of a multi-million-pound cloud computing platform.

2.4 Computer Clusters

A cluster can be defined as a collection of resources controlled by a single master. Computing clusters typically consist of multiple nodes or computers which are connected together through a local area network (LAN). The networked nodes act as a single machine but due to their shared resources, they are more powerful than a single machine. These types of clusters are commonly referred to as a “supercomputer”. A cluster enables accelerated processing speed due to sharing the work between multiple CPU’s, along with other items such as more storage and a large resource pool. Clusters, however, are not cheap to implement or maintain. The main disadvantage of using a cluster is the large overhead costs and physical space required for multiple nodes in contrast to the cost of and space required for a single node.

In relation to the Raspberry Pi, a cluster is a number of Pi’s that are connected together and work together so they become one system. Clusters use orchestration software to have each node set to perform a task which is then controlled and scheduled by that piece of orchestration software. Clusters are normally connected to each other through “Local Area Networks” (LAN). Each node will then run its own instance of an operating system.

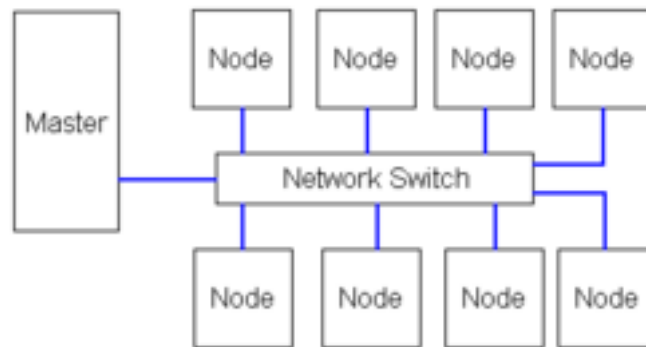


Figure 5: Simple Beowulf Cluster Diagram (Citizendium)

Figure 5 shows a simple Beowulf cluster diagram (Citizendium). A Beowulf cluster consists of identical mid-range computers, which just like described earlier, are networked together via a LAN. Each of the computers have libraries and software installed which permits a shared processing pool. The result of which is a high performance computing cluster made from mid-ranged personal computing hardware.

2.5 ARM Architecture

ARM holdings is a British company that develop “RIS (reduced instruction set) architecture and cores for computer processors”. Originally ARM stood for “Acorn RISC Machine” but then later was renamed to “Advanced RISC Machine”.

The company then licenses their developments to various companies, who will use them in their own-designed products, for instance; SoC (Systems on Chips) which collaborate memory, interfaces etc.

ARM architecture is the basis behind every ARM processor. As time has gone by, the ARM architecture has advanced to include new features in response to the demand for new functionality, built-in security, performance measures etc.

2.6 Linux

Linux is an operating system, it is similar to Windows and OS X, however, there are a number of differences between them.

Linux is a huge contender in the computing industry, it is used in the New York Stock Exchange as well as consumer devices.

Linux is an open OS which means it is developed collaboratively, with many different distributions being available. Over 1000 contributors, from at least 100 unique companies have contributed to every kernel release. Over the last 2 years alone, more there were 3200 contributions to the core kernel which is only a small part of a Linux distribution.

2.6.1 Arch Linux ARM

Arch Linux ARM is a distribution of Linux made for the ARM instruction sets, for instance, the ARMv7 Broadcom processor used on the Raspberry Pi 2.

Arch Linux ARM began development in 2009 as a distribution called “PlugApps”. This further advanced to become the first ARM port of the Arch Linux distribution with support for ARMv5. By 2011, it was extended to work on ARM architecture which included a “hard-float” point, such as the ARMv7 chipset and thus was renamed Arch Linux ARM.

Arch Linux ARM gives competent users full control and responsibility over their system as it continues the Arch Linux approach of minimal overheads for maximum performance.

2.6.2 systemd

systemD gives a basic process for configuring and controlling which software starts on a Linux system once it is switched on. This is not the only feature behind systemD however, it can also create a log of the systems activity and network amongst others.

SystemD is an “init” system which is optimized by other Linux distributions to bootstrap user space and provide management of all processes.

2.6.3 Sudo

Sudo stands for “super user do!”. Sudo is a command that gives the user “elevated privileges” which are sometimes required to perform specific administrative tasks. It is Linux’s equivalent to the Windows user account control and Mac’s security dialog box.

2.7 Docker

Docker is not the easiest of software to explain. However, it is easy to use once it is understood.

One of the great features of Docker is that it is open-source. This is extremely handy for developers as they can use Docker in the way that they want and also contribute to the project if they so wish.

Docker allows a user to build, run, test and deploy applications into a container which consists of simply the applications code, required libraries and system tools, without the requirements of an operating system to control the physical resources. This resource management is done by the Docker software itself, allocating the required resources as needed by the Docker application.

Docker is known for its collaboration of Docker images, which are made available by developers to other users through both the public Docker index registry, Docker Hub and private Docker registries. It is a manager for infrastructure, at present for Linux containers, yet in future it will incorporate KVM, Hyper-V, and Xen, just to give some examples.

Docker is a new container technology and is famous due to its abilities in bringing numerous applications from servers. Docker is most popular in data centres and cloud storages. Software companies are starting to adopt Docker technology from virtual machines at an exceptional rate, for example, Amazon, Google, and Microsoft.

2.7.1 History

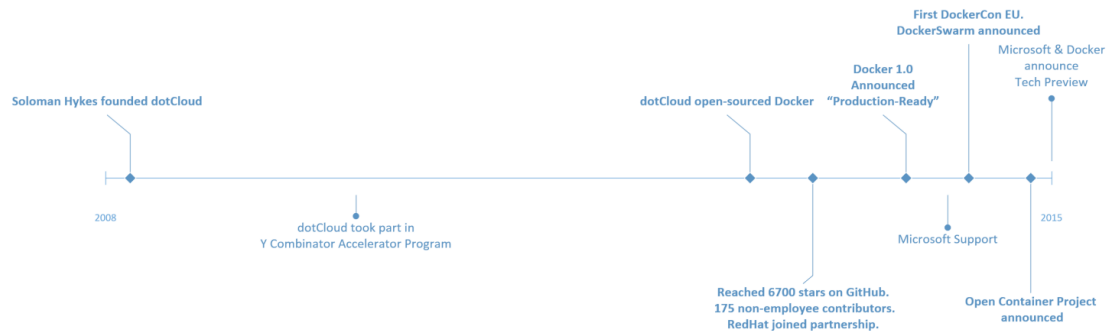


Figure 6: Timeline of Docker Success

In 2008, Solomon Hykes established dotCloud to construct a language-agnostic Platform-as-a-Service (PaaS) offering.

In March 2013, dotCloud open-sourced Docker, the centre building piece of dotCloud. Early forms of Docker were nothing more than a wrapper around LXC matched with a union file system, yet the uptake and velocity of development was shockingly quick.

Inside of six months, it had more than 6,700 stars on GitHub and 175 non-employee contributors. This drove dotCloud to change its name to Docker, Inc. also to refocus its business model.

Docker 1.0 was declared in June 2014, only 15 months after the 0.1 release; already in use by major companies including Spotify and Baidu.

2.7.2 Docker vs Virtual Machines

Recalling that a virtual machine (VM) is a separate computer i.e. guest computer, which runs above the host computer, the VM is lying over the virtualization layer. Docker, on the other hand, is detached from the host machine but shares the hosts OS and libraries.

The VMs are constructed firmly in light of virtual hardware technology and have mass system requirements. The shared OS technique utilized in Docker containers make them more productive than the virtual machines as you need not to utilize the virtualization of hardware. VM's absorb a full guest OS, applications, crucial libraries and binaries, all of which could be many GBs in size. Whereas, containers absorb the application and all dependencies, however they will distribute the kernel with the other containers. Containers will run as a standalone process of the host OS's user space and are only limited by the processor instruction sets available. Because of this, it is not possible to run i86 or x64 Docker images on ARM architecture. Docker containers can be run on any computer, cloud or infrastructure are not limited to physical requirements, such as requiring processors with hypervisor technology built into them.

Essentially, the main contrast between VM's and Docker are that the VM's use the whole system resource pool of the host, whereas Docker will dynamically use the resource required. Figure 7 shows the infrastructure comparison of Docker vs Virtual machines (Docker 2016).

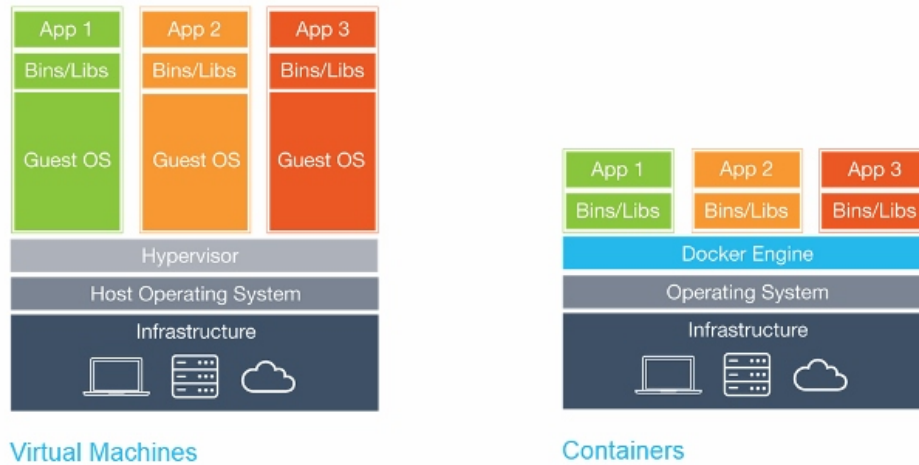


Figure 7: Virtual Machines vs Docker Comparison (Docker 2016)

2.7.3 Docker Elements and Docker Parts

Docker contains numerous parts and elements. The parts are the applications and the elements are utilized by all these parts. The Docker parts and elements lie on top of the libraries, functionalities and the frameworks.

Major Docker Parts

- **Docker daemon:** It is utilized for overseeing Docker containers such as LXC on the host system in which the Docker runs.
- **Docker image index:** This is a public or a private repository, which is utilized for storing Docker images.
- **Docker CLI:** This is utilized to make commands furthermore to communicate with the Docker daemon.

Major Docker Elements

- **Docker containers:** The Docker containers are directories that contain all parts of an application.
- **Docker files:** These are the scripts which automates the compiling of images.
- **Docker images:** These contain different snapshots or images of applications and are run inside containers.

2.7.4 Docker Containers

The complete procedure of making portable applications utilizing Docker depends on containers. Docker containers are directories which are bundled together to utilize a host machines resources, regardless of host platform. Docker containers can therefore be shared across multiple host machines to allow for rapid scaling of required resources. The only requirement is having Docker installed on the host machines. The containment is accomplished through LXC or Linux Containers. Figure 8 shows the infrastructure of a Docker container (Docker 2016).

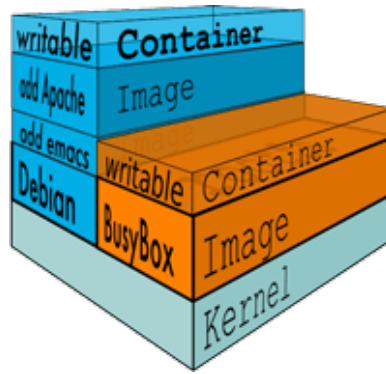


Figure 8: Docker Containers (Docker 2016)

The Docker containers use Linux containers or LXC. The different components of Docker containers are;

- Portability of the application
- Isolation of application processes
- Preventing the containers from accessing external resources
- Resource consumption management

The components are not restricted to the above. The Docker containers need less assets than the virtual machines which are useful in segregating applications.

Docker containers never permit:

- Mingling up with different processes
- It never permits dependency hell
- Docker containers never permit vulnerability to external attacks and never abuse system resources.

Docker containers are subject to Linux containers. These containers resemble system directories, however they are not at all like standard directories due to the way in which containers are formatted and formed. The Docker containers empower application portability and continuous building of the containers.

Each Docker container is layered on top of another, taking after onions and every progression inside of a container includes topping another piece over the old one. This gets translated to a simple format inside the file system.

The real favourable advantage of Docker containers is flexibility in launching and creation of various new images and containers, which are light-weight. The light-weight nature of these containers is due to their layered and continuous build-up. As the containers are dependable on file concepts, performance roll-back and snapshots are effectively done in a matter of seconds and never present trouble to resources. For instance, version control systems of VCS. The base for each Docker container is the Docker image element on top of which different layers are built.

2.7.5 Docker Images

Docker images form a base of the container and just from here alternate layers start to shape. These are the same as kernel disk images that are useful in running applications on desktop PCs or servers.

These base images are helpful in smooth and ceaseless portability of applications over different systems. These constitute an unbending, predictable and dependable base with all that is fundamental for running applications. On the off chance that everything is independent and the system level modification dangers have stayed away from, the Docker container will get to be imperviousness to exposures outside which will avert dependency hell.

At the point when more layers, like applications and tools, get included on top of the Docker images, the new images get focused on continuous changes. At the point when another

Docker container is made from the updated image, the processing will begin from where it was left off. At the point when working with Docker containers, the union file system is valuable in uniting all as a single element.

The Docker images are begun expressly in Docker CLI for making the new containers or they can be characterized inside of a Docker file for prompt and automatic image build-up.

2.7.6 Docker Files

Docker files are blocks of scripts which contain sets of directions, instructions and commands that are begun (executed) from the Docker image (new image). Each order of execution is layered as a new product. This is useful in building the finished product. They, in this way, form a substitution of the conventional manual or rehashed process of update. At the point when a Docker file finishes execution, it will end with the image formation which is valuable for making new containers.

2.8 Kubernetes

Whilst Docker is an elegant solution for building applications as micro services, it does not include all the necessities to deploy and manage a large sum of containers that work together and scale up as the demand increases.

Kubernetes brings together Docker containers, which make up an application, into units to allow for easier management and discovery.

Kubernetes is not a Platform-as-a-Service tool, however, it serves as a basic framework, allowing users to choose the types of application frameworks, languages, monitoring and logging tools, and other tools of their choice. With this, Kubernetes can be used as the basis for a complete PaaS to run on top of.

Kubernetes is written in the “Go” programming language. Due to Kubernetes being open-source, the source code can be browsed and contributed to on GitHub. Currently exceeding 700 different contributors and over 13,500 stars on GitHub. many of the contributors are engineers from Google, Red Hat, and other companies. Also, to add, the Cloud Native Computing Foundation, a project of the Linux Foundation, operates to provide a common home for the development of Kubernetes and other applications seeking to offer modern application infrastructure solutions.

2.8.1 History

Due to the huge demand for a piece of software that can develop, deploy, manage and scale containerized applications, Kubernetes was released as an open-source project by Google in 2014 with the aim at standardizing how containerized applications are managed.

The aim of Kubernetes is to act as an overlay on top of a company’s data centre, and other open source projects could work on solving other aspects of container management i.e. creating streamlined operating systems or graphical administration tools.

2.8.2 Features

- **“Automatic bin-packing”**: This feature automates the process of ordering the containers based on their requirements from resources and constraints whilst not relinquishing availability. It will automatically juggle the critical and best-effort workloads to increase usability and save resources.
- **“Horizontal scaling”**: This feature allows the user’s application change scalability dependant on their CPU usage or with the use of a command or user-interface.
- **“Automated rollouts and rollbacks”**: With this feature, Kubernetes will automatically roll out application changes or configuration changes whilst keeping an eye on the health of the application to make sure it doesn’t end all the instances at the same time. In case of an error, Kubernetes rolls back the changes automatically.

- **“Storage orchestration”**: This feature allows Kubernetes to mount the user-preferred storage system automatically from either local storage, a public cloud service for instance; Google Cloud Platform or Amazon Web Services, or a network storage system i.e. Flocker or Cinder.
- **“Self-healing”**: If containers throw an error, Kubernetes will restart the containers. If the nodes do not respond and affect the containers unresponsive to a health check defined by the user, Kubernetes will replace and reschedule the containers. Also, with this feature, it will not show the containers to clients unless they are “ready to serve”.
- **“Service discovery and load balancing”**: With this feature, Kubernetes assigns containers a unique IP address and a solidarity DNS name for a group of containers, with the option of load-balancing across them. This means that the user does not need to change their application to use an unknown to them service discover mechanism.
- **“Secret and configuration management”**: This features allows the deployment and update of secrets and application configurations without the need to rebuild an image. With this, the secrets in a stack configuration will not be shown.
- **“Batch execution”**: This feature is in place in case containers fail. Kubernetes will be able to manage CI workloads and batches by replacing the failed containers, if the user requests.

2.8.3 Core Components

- **Etcd**: A “distributed key value store”, this stores data across a cluster reliably. It can handle machine failure and control network partitions leader elections. Etcd is open-source and available on GitHub.
- **Flannel**: “Flannel is a virtual network that gives a subnet to each host for use with container runtimes. Platforms such as Google’s Kubernetes assume that each container (pod) has a unique, routable IP inside the cluster. The advantage of this model is that it reduces the complexity of doing port mapping.” (CoreOS)
- **Hyperkube**: A container for Kubernetes server components. “It combines all of Kubernetes server components into a single binary where the user selects which components to run in any individual process.” (godoc)
- **Pause**: A network name space container. Essentially, if a container fails but is then restarted, all the network setup will still be available.

2.9 Live Migration

Live Migration is the process of moving a container from one data centre to another with minimal downtime and configuration.

2.9.1 Criu and P. Haul

Criu and P. Haul permits the freezing of any running application and relevant operating data, known as checkpointing. With this checkpoint made, the application and any open network connections can be transferred to another location. Once transferred, the application, resources and connections are restored to the same point they was frozen at and continue running as before. The relocation of the application varies. However, whilst relocation occurs, any incoming requests are entered into a queue. The aim of checkpoint restore is that only a short delay in responsiveness is experienced by the users, rather than users being disconnected from the running application and made to re-launch a new connection to an alternatively located application. With addition of the traffic load-balancing solution, the theory is that many containers can be migrated without redeploying, thus limited downtime is experienced.

There are many examples of migrating live containers with Docker online, however, the most notable example was at DockerCon 2015, Michael Crosby and Arnaud Porterie demonstrated moving a “Quake 3” container around the world using Criu.

Both Criu and P. Haul are open-source and the source-codes are available on GitHub. Criu is a checkpoint and restore module for Linux, whereas P. Haul is an interactive web-based user-interface.

2.10 Case Studies

- **Dockers Ease in Trying New Software:** Developers are always trying out new software. Trying out new software isn't always a pleasant experience; as it involves setting up new configurations after the initial download. A developer's time is extremely valuable so a simple way to install and configure new software is a necessity. Docker is an extremely simple way of running software, which in the background, takes care of fetching the image and running it. However, it is not always about new software. For instance, having a database server such as MySQL set up quickly on a laptop. Docker makes this an extremely simple task which could save many hours of time by just executing a simple command such as;

1. `Docker run -d -p 3306:3306`

- **eBay Simplifies Application Deployment using Docker:** eBay previously used VM's to try and optimize their development process. However, recently, due to restrictions using VM's, Docker has been utilized by eBay for their "continuous integration process". They utilize Docker containers to speed up the path from the developers through to the test and QA. eBay now have a working process which works on the developer's laptops, communal resources and in production.
- **GE switched to Docker from Virtual Machines:** GE Appliances now use Docker containers. Utilizing their ease of use and portability to deploy their applications in any environment, GE Appliances now use Docker as their base for their application "Voyager". The company can now write a Docker file, use the applications user-interface to create a new request, and then be notified of a running service via the service discovery layer. Within only 4 months of using Docker, GE migrated more than 60% of their legacy data centre and are now using a hybrid cloud environment.

3 Analysis

The original goal of the project was to extend the work done by the University of Glasgow's Raspberry Pi Cloud team. The team had already successfully ported Docker to run on their Raspberry Pi cluster, as well as an early version of Kubernetes.

With this in mind, the project goals were to build a miniature cloud-based cluster using 15-20 Raspberry Pi 2's. With this set as a cluster, a cluster visualisation platform such as LXC and Docker would need to be used to manage and provision the cluster. The aim was to look at using cluster management tools such as Shipyard, Ansible and Kubernetes whilst restricting the cluster to the Arch Linux ARM operating system.

The problems faced whilst undertaking this project were in their masses. There were no existing management tools compatible with the ARMv7 processor architecture and any management tool needed to be ported to the ARM architecture. With this, the main work behind this project is to provide a solution for the ARM community, to be able to use a fully functioning cluster management tool on their ARMv7 devices.

The Raspberry Pi has become a valuable due to its low-cost, miniature-size and also being Linux based, it has opened up a world of opportunity for research into how cloud infrastructures work within Data Centers.

In terms of the importance of this project, by creating a small 5-node cluster, with the ability to install ground-breaking software, such as Docker and Kubernetes, has provided a platform for research and educational-uses such as setting up a miniature cloud-based platform in a school and allowing the students to hack, learn and understand the software and technology used in large-scale Data Center's. With this also, Containers have been around for a long-time, however, their full functionality has not been utilized until the recent releases of Docker. Containers are now, one of the most valuable assets to cloud computing. As Docker is available on a low-cost single-board computer, containers are much more accessible to a standard user, therefore the knowledge-base will grow, especially with the use of Kubernetes and now the Dashboard, to easily display and deploy the containers on a user-interface instead of a command-line.

During the first three months of the project, the majority of time was spent learning about Docker and LXC and setting up the Raspberry Pi cluster. With that, came trial and error of the cluster management tools. Unable to get any of the management tools such as Shipyard and Ansible working with the ARMv7 architecture and realising this was using the majority of time, the project then changed focus to Kubernetes.

There was a lot of chatter regarding Kubernetes online and a huge community of developers and hobbyists forming around it. Therefore, it made sense to focus on what was being described as the "next big thing" in the cloud computing world of containers and virtualization. Not only that, the Raspberry Pi Cloud team had already managed to get Kubernetes working in the early stages, therefore this project just needed to extend on this, which gave back a lot of time to focus on the management side of the cluster.

Once Kubernetes was up and running on the cluster, it was limited to extremely basic use. Initially, there wasn't a compatible user-interface as Kube-UI was hard coded to i86 & x64 systems, which therefore required the use of SSH and learning terminal commands in order to manage the cluster itself. Furthermore, crucial elements of Kubernetes such as Heapster were unavailable for ARMv7. With this, the main focus for this project naturally became creating a user-interface for the cluster.

The Kubernetes project had a feature called "Kube-UI", which essentially was a cluster management dashboard user-interface which allowed the user or administrator to easily verify and understand the performance of a cluster and the jobs running. This is shown in figure 9 (Kube-UI 2016).

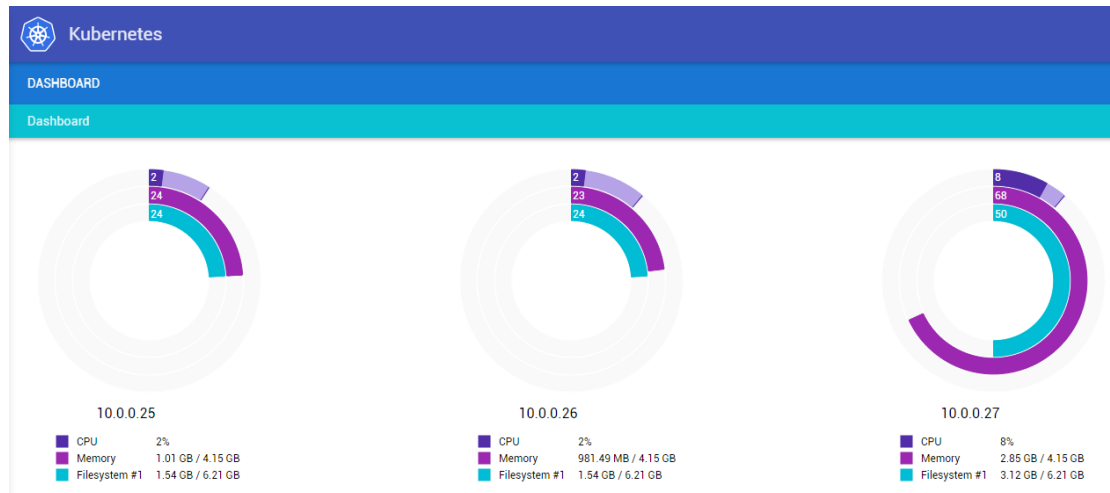


Figure 9: Cluster Analysis on Kube-UI (Kube-UI 2016)

With discussions with the project supervisor, who is part of the “University of Glasgow’s Raspberry Pi Cloud” team, this is what the team needed to extend on their cluster.

The “Kube-UI” was unable to build on ARMv7. A problem which is faced often within this project. The problem with kube-dash is that it had no backend network for ARM such as flannel. At the time, with little documentation from the Google developers, the problem could not be easily located; suspected to be either a problem with Heapster as this was not built for ARM, as at the time, Heapster was also in its early stages of development and as such, it too was hard-coded for i86 or x64 systems with no ARM support. Without, many hours of development being dedicated to Kube-UI, it would not be possible to re-write both it and the early versions of Heapster due to the time constraints faced.

Unable to get Kube-UI working on the Raspberry Pi cluster, alternative solutions were sourced. One such solution for tracking and displaying the resource costs on each node is “cAdvisor”. However, it is not integrated with Kubernetes, but it did have native support for Docker. As cAdvisor was developed by Google on the GO language, implementation of this from source was simple. cAdvisor only provides container users an understanding of the resource usage and performance characteristics of their host hardware however. It is a running daemon that collects, aggregates, processes and exports information which can then be displayed via a web page. This is shown in figure 12.

However, the problem faced with cAdvisor, is that it would only show a single node, it would not show the whole cluster. With this, a new search began for a solution in which a user-interface would show the analytics of the whole cluster and could be integrated with Kubernetes.

Upon research, a new add-on was being developed by Kubernetes called “Dashboard”. At the time of finding, Dashboard had only recently started development and there was no official release or BETA available. The same problem was faced with no support for ARMv7 architecture. With the add-on still being in development, there was limited documentation at the time. However, with this being an open source product hosted on GitHub, the ability to communicate with the team and main developers behind the project quickly became an advantage.

Even without a BETA or ARM release of the Dashboard, a new goal of the project was to get this built into ARM devices as Google had ceased development on Kube-UI in order to concentrate on this new Dashboard feature, and thus it became the single User-Interface essential to have on clusters running Kubernetes.

Through many hours of trial and error, in part due to no documentation being made available from the developers, prodding at Docker commands, working from errors such as “Go” not existing and discovering how to build “Go 1.5” manually from source, having to build images manually on a daily basis, updating source code daily due to it rapidly changing if something new was added or changed in the back-end then it may not have worked and would need to rebuild to allow for it to work again, and rebuilding Heapster from scratch, a solution was finally found. This was all done prior to the BETA release at the end of February. Not only that, but the solution this project presents created the world’s first version of Kubernetes Dashboard compatible and fully working on the ARM architecture. Although the dashboard was only basic at the time of implementation on the Raspberry Pi, with limited functionality shown in figure 19, the new ARM compatibility provided a basis for the Google team to implement support for ARM devices in the source code, not only for Dashboard, but for Kubernetes itself. This is explained in more detail in section 6 of the report.

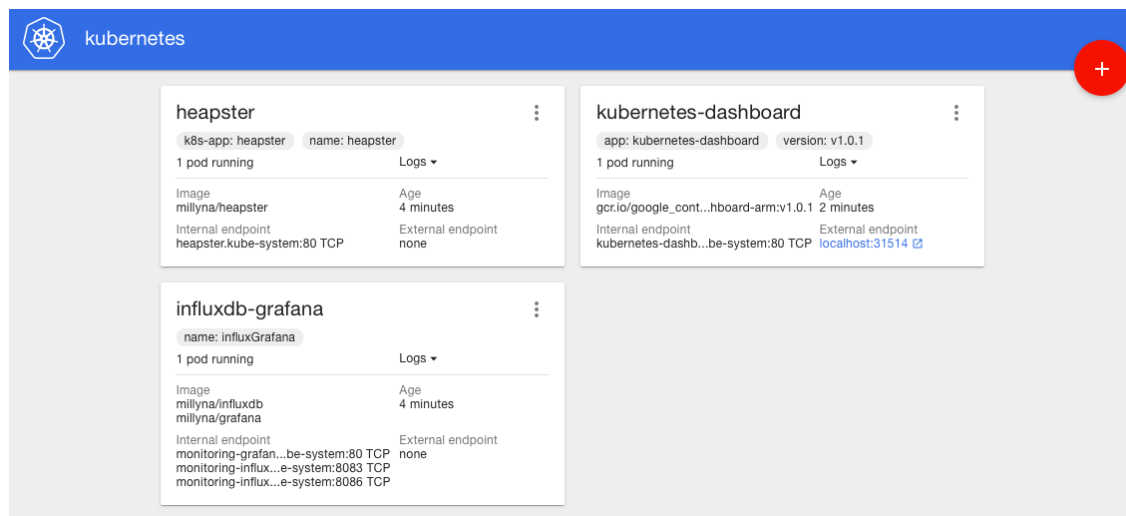


Figure 10: Kubernetes Dashboard - Running Containers

As the development continued on the Dashboard, a production version, 1.0 was officially released. With this, Google published the Dashboard as a Docker Image for various platforms, including ARM, which could be pulled from the Docker hub. This is due to the Google team seeing the effort and extremes ARM users were taking to build the Dashboard. By Google publishing a working ARM version of the dashboard, based in part on the work done by this project, this made the process extremely easy for any updates or release thereafter.

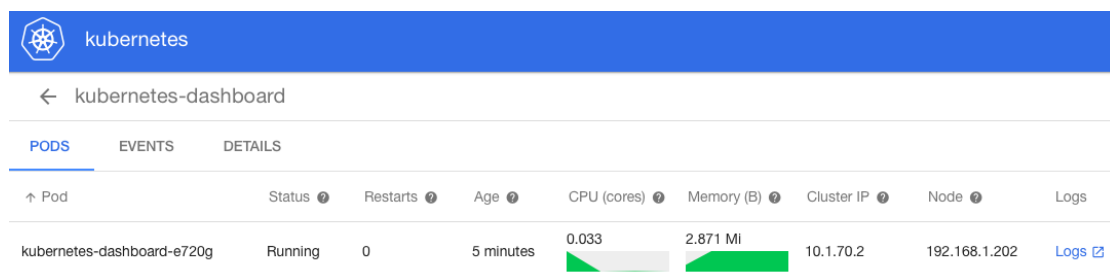


Figure 11: Kubernetes Dashboard - Pod Details

With the Dashboard now fully working on ARM devices, the decision was made to try and extend the project further by looking into the live migration of Docker containers. With research, the primary focus for container migration was a checkpoint and restore module called “Criu” with the UI “P. Haul”. Live migration is due to be a huge game-changer in the cloud computing world, as well as huge change for the world of gaming as discussed with the example of Criu and P. Haul in chapter 2.9 of the report. Since DockerCon 2015, there were

Shares 1024 shares

Memory

Limit 902.34 MiB

Swap Limit 1024.00 MiB

Usage

Overview

Processes

User	PID	PPID	Start Time	CPU %	MEM %	RSS	Virtual Size	Status	Running Time	Command	Container
root	2122	2418	13:29	0.00	5.85	44.41 MiB	917.73 MiB	Idle	00:00:18	hyperbahn	/system.slice/hyperb...
root	2418	1	13:29	0.40	3.50	22.29 MiB	955.46 MiB	Idle	00:00:53	docker	/system.slice/docke...
root	2266	2227	13:29	1.30	1.30	12.49 MiB	781.48 MiB	Idle	00:00:16	etcd	/system.slice/etcd
root	1	0	12:11	0.40	0.40	4.18 MiB	22.50 MiB	Idle	00:00:24	systemd	/system.slice/syste...
root	2227	1	13:12	0.20	2.50	12.50 MiB	922.46 MiB	Idle	00:00:03	docker	/system.slice/syste...
root	321	2	12:12	0.10	7.00	0.00 B	0.00 B	S	00:00:07	sleep/0	/system.slice/syste...
root	179	1	12:12	0.10	1.60	13.34 MiB	27.57 MiB	Idle	00:00:09	systemd-journald	/system.slice/syste...
root	2274	1	13:29	0.10	2.10	19.83 MiB	816.46 MiB	Idle	00:00:01	docker	/system.slice/syste...
root	2273	1	13:29	0.10	2.10	19.12 MiB	816.46 MiB	Idle	00:00:01	docker	/system.slice/syste...
root	1959	1	13:29	0.10	2.10	19.28 MiB	820.46 MiB	Idle	00:00:01	docker	/system.slice/syste...
root	2	0	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kthreadd	/system.slice/syste...
root	3	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:01	kworker/0/0	/system.slice/syste...
root	5	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	hwclockd/0/0	/system.slice/syste...
root	7	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:03	rcu_sched	/system.slice/syste...
root	8	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	rcu_bh	/system.slice/syste...
root	9	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	migration/0	/system.slice/syste...
root	10	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	migration/1	/system.slice/syste...
root	11	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/0/1	/system.slice/syste...
root	13	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/0/0	/system.slice/syste...
root	14	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	migration/2	/system.slice/syste...
root	15	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/0/2	/system.slice/syste...
root	17	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/2/0	/system.slice/syste...
root	18	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	migration/3	/system.slice/syste...
root	19	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/0/0	/system.slice/syste...
root	21	2	12:11	0.00	0.00	0.00 B	0.00 B	S	00:00:00	kworker/0/0	/system.slice/syste...

CPU

Total Usage

Usage per Core

Usage Breakdown

22

4 Design and Development

Within this chapter of the report, the various steps and issues encountered whilst building the 5-node Raspberry Pi cluster will be discussed.

4.1 Cluster Considerations

At the time of commencing the project, a number of details were considered. At the initial project meeting with the project supervisor, it was decided that a 20-node cluster would be produced using the Arch Linux ARM operating system, this was due to deciding to represent the ground-breaking work by creating a miniature version of the “PiCloud” to work with.

4.2 Hardware Setup

The hardware setup for the Cloud cluster consisted of a five Model B version of Raspberry Pi 2 boards. Chapter 2.3 provides the details regarding the Raspberry Pi boards.

The original proposal of the project stated to use twenty Raspberry Pi’s as a cluster, however, as the project changed rapidly due to the software being used as ground-breaking, it was then decided against building such a large cluster, and to focus more on the software implementation vs the cluster size.

The Raspberry Pi devices are connected through the home network via Ethernet to a ASUS DSL-AC68U router along with a five-port 10/100Mbps TP-LINK TL-SF1005D unmanaged desktop switch in order to provide a LAN infrastructure incorporating both DNS and internet access.

As key building blocks of the cluster, each of these microcomputers runs Arch Linux ARM from a SanDisk 16GB SD card storage.

A powered USB Hub was used to provide power to the boards via micro-USB cables.

The power consumption was not monitored as it was not relevant to the project. However, as the Raspberry Pi 2 only requires an average of 0.25A at idle to 0.5A under load, this was not taken into consideration.

The summary of the hardware specification for the cluster is shown in Table 2.

Table 2: Hardware Specification Summary

Processor	Clock	Memory	Storage	NIC	Hostname
Quad-core ARMv7	900 MHz	1GB	SD 16GB	10/100 Mbps	Kube-Master01
Quad-core ARMv7	900 MHz	1GB	SD 16GB	10/100 Mbps	Kube-Worker01
Quad-core ARMv7	900 MHz	1GB	SD 16GB	10/100 Mbps	Kube-Worker02
Quad-core ARMv7	900 MHz	1GB	SD 16GB	10/100 Mbps	Kube-Worker03
Quad-core ARMv7	900 MHz	1GB	SD 16GB	10/100 Mbps	Kube-Worker04

4.3 Installing Arch Linux ARM

The Raspberry Pi 2 B+ has a quad-core ARM cortex-A7 CPU, which is an ARMv7 processor. Due to having an ARM processor, it is supported by a limited number of operating systems.

For this project, Arch Linux ARM was the choice of operating system.

Some of the advantages for using Arch Linux ARM for the operating system are; it is a rolling release, which means it is updated regularly compared to some other operating systems. It is cleaner packaging and also systemd by default.

The Raspberry Pi is an ARM-based device and therefore needs binaries compiled for this architecture. These binaries are provided by the Arch Linux ARM project which ports Arch Linux to ARM-based devices.

The Arch Linux ARM website provides step by step instructions with how to install the operating system on a Raspberry Pi using the command line.

“With this set of instructions, replacing sdX with the device name for the SD card as it appears on the computer.

1. Start fdisk to partition the SD card:

```
2. fdisk /dev/sdX
```

2. At the fdisk prompt, delete old partitions and create a new one:
 - a. Type **o**. This will clear out any partitions on the drive.
 - b. Type **p** to list partitions. There should be no partitions left.
 - c. Type **n**, then **p** for primary, **1** for the first partition on the drive, press **ENTER** to accept the default first sector, then type **+100M** for the last sector.
 - d. Type **t**, then **c** to set the first partition to type W95 FAT32 (LBA).
 - e. Type **n**, then **p** for primary, **2** for the second partition on the drive, and then press **ENTER** twice to accept the default first and last sector.
 - f. Write the partition table and exit by typing **w**.

3. Create and mount the FAT filesystem:

```
1. mkfs.vfat /dev/sdX1
2. mkdir boot
3. mount /dev/sdX1 boot
```

4. Create and mount the ext4 filesystem:

```
1. mkfs.vfat /dev/sdX2
2. mkdir root
3. mount /dev/sdX2 root
```

5. Download and extract the root filesystem (as root, not via sudo):

```
1. wget http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-
   latest.tar.gz
2. bsdtar -xpf ArchLinuxARM-rpi-latest.tar.gz -C root
3. sync
```

6. Move boot files to the first partition:

```
1. mv root/boot/* boot
```

7. Unmount the two partitions:

1. `umount boot root`

8. Insert the SD card into the Raspberry Pi, connect the Ethernet and apply 5v power.
9. Use the serial console or SSH to the IP address given to the board by the router.
 - Login to the default user *alarm* with the password *alarm*.
 - The default root password is *root*.”

(Arch Linux ARM 2016)

Arch Linux ARM comes packaged with a *pacman* package manager tool. *pacman* is one of the main tools used within Arch Linux. Pacman integrates a binary package within a simple build system. pacman aims for simplicity with package management of repositories whether official or simple. Pacman syncs with any known compatible software repositories resulting in a system which is updated regularly. Pacman enables a user to install or download any packages with all the packages needed dependencies, with a simple command.

4.3.1 Updating the Operating System

Updating the operating system is essential before doing any other installations on the Raspberry Pi board due to it being a rolling release operating system, any bug fixes and compatibility with other software packages are implementing in each release.

To update the operating system, it is a simple command as the root user:

1. `pacman -Syy`

Or

1. `pacman -Syu`

4.4 Installing Docker

As the project involved working with manually built Docker images, to build these on the Raspberry Pi it requires a dedicated “swap file”. Through testing, it was discovered that using a 1GB swap file is optimum.

1. Create the 1GB swap file:

1. `dd if=/dev/zero of=swapfile bs=1M count=1024`

2. Enable the swap file and set the permissions for system only use:

1. `mkswap /swapfile`
2. `chmod 600 /swapfile`
3. `swapon /swapfile`

3. Now that the swap file is created, it needs to be added to the list of disks to make it accessible each time the Pi is rebooted. This is done by editing the fstab file found in the /etc directory:

1. `nano /etc/fstab`

The following line needs to be added to the bottom of the file:

1. `/swapfile none swap defaults 0 0`

This will tell the Pi on-boot that it has to re-enable the 1GB swap file created.

4. At this point, in order to ensure that the swap file is working and enabled at boot, it is best to reboot the Pi.

Docker can now be installed from local package within the Arch Linux package repository. The latest build was released 12/03/2016. With this, steps 1 – 4 are now redundant, due to the dependencies being packaged within the Docker image. However, there are known issues with the compatibility of Docker and the ARMv7 instruction set, which was later overcome by using a statically built Docker binary from the source code.

Docker depends on several packages which are specified as dependencies. The core dependencies needed to install Docker are:

- bridge-utils
 - Device-mapper
 - Iproute2
 - LXC
 - sqlite
1. Install the required dependencies that are listed above. This needs to be done as the root user or as sudo.
1. pacman -S bridge-utils device-mapper iproute2 lxc sqlite
 2. Install NTP to ensure the Pi's have the correct time to download the docker images. This needs to be done as the root user or as sudo.
1. pacman -S ntp
 3. Reboot the Raspberry Pi board.
 4. Log back into the Raspberry Pi via SSH
 5. Install Docker. This needs to be done as the root user or as sudo.
1. pacman -S docker

4.5 Configuring Docker

Due to an incompatibility with the version of Docker and Arch Linux available at this point in the project, Docker would not run by default. Docker would throw various errors such as “No such files/directories”. To get around this issue, extra lines were needed to be added to the docker.service:

1. Stop Docker. This needs to be run at either root user or sudo.
1. systemctl stop docker
2. Open docker.service with nano.
1. nano /lib/systemd/system/docker.service
3. Change the line “ExecStart=/usr/bin/docker -d -H fd://” to read:
1. ExecStart=/usr/bin/docker -d -H fd:// --exec-opt native.cgroupdriver=cgroupfs

4. Save and exit the file.
5. Reload the Docker Daemon.

```
1. systemctl daemon-reload
```

6. Start Docker. This needs to be run at either root user or sudo.

```
1. systemctl start docker
```

At this point, Docker would load and work. This was a known issue at the time of first working with this and has since been patched by the Docker team. Later in the project, it was discovered that for compatibility and speed of expanding the cluster, it was in the best interests to build a static Docker image from the source code, including changes required to enable full ARM compatibility. This static Docker binary and associated service files were published to GitHub to allow ease of building nodes.

1. Download the custom Docker binary from GitHub.

```
1. curl -sSL https://github.com/Millyna/rpi-fyp/releases/download/1.0/docker-1.10.0/usr/bin/docker
```

2. Give the binary the ability to run:

```
1. chmod +x /usr/bin/docker
```

3. Create a Docker group:

```
1. groupadd -f docker
```

4. Create a Docker service:

```
1. nano /usr/lib/systemd/system/docker.service
```

and enter the following text:

```
1. [Unit]
2. Description=Docker Application Container Engine
3. Documentation=https://docs.docker.com
4. After=network.target docker.socket
5. Requires=docker.socket
6.
7. [Service]
8. Type=notify
9. ExecStart=/usr/bin/docker daemon -H fd://
10.     MountFlags=slave
11.     LimitNOFILE=1048576
12.     LimitNPROC=1048576
13.     LimitCORE=infinity
14.     TimeoutStartSec=0
15.
16.     [Install]
```

17. `WantedBy=multi-user.target`
5. Save and exit
6. Create a Docker socket:
1. `nano /usr/lib/systemd/system/docker.socket`

Paste the following:

1. `[Unit]`
2. `Description=Docker Socket for the API`
3. `PartOf=docker.service`
- 4.
5. `[Socket]`
6. `ListenStream=/var/run/docker.sock`
7. `SocketMode=0660`
8. `SocketUser=root`
9. `SocketGroup=docker`
- 10.
11. `[Install]`
12. `WantedBy=sockets.target`

7. Save and exit
8. Reload the service

1. `systemctl daemon-reload`

There is a systemd service unit created for Docker.
(This needs to be run as either root user or sudo)

1. To start Docker:

1. `systemctl start docker`

2. To start Docker on system boot:

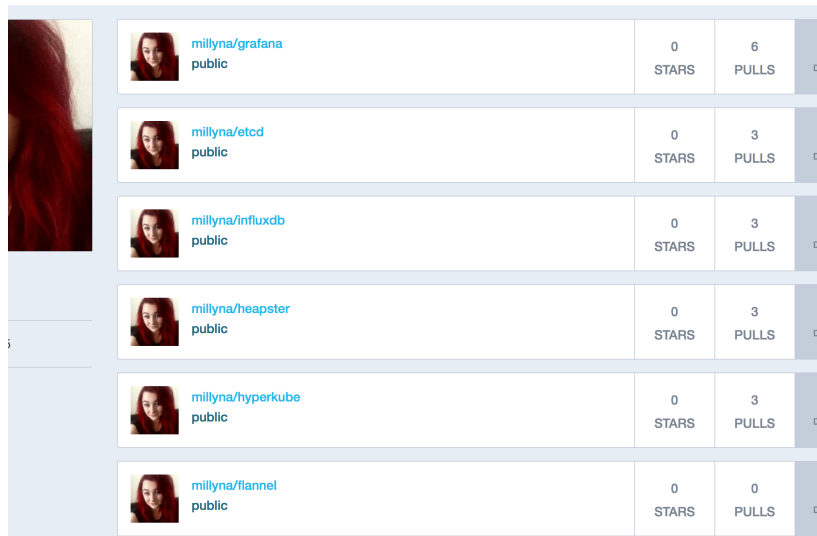
1. `systemctl enable docker`

4.6 Building Docker Images

1. To build the Docker images for Kubernetes, firstly, “GO” needs to be installed. (See section 4.12 of the report for building GO to the latest 1.6 and add this to the path variables).
2. Once done, the latest .tar files need to be downloaded from the respective sources GitHub repositories and then built into the Docker images.
(See Appendices 8.2.1: Building Docker Images).
3. Once all the binaries were built, they needed to be wrapped into Docker files. This was done by copying the binary into the local build folder and wrapping it into a Docker image:

1. `cp /build/bin/etcd`
2. `docker build -t millyna/etcd .`

- The Docker files are built for each of the required Docker modules. To speed this process up, the Docker images were uploaded to the docker hub registry. This is shown in figure 13.









 millyna/grafana public	0 STARS	6 PULLS	DE
 millyna/etcd public	0 STARS	3 PULLS	DE
 millyna/influxdb public	0 STARS	3 PULLS	DE
 millyna/heapster public	0 STARS	3 PULLS	DE
 millyna/hyperkube public	0 STARS	3 PULLS	DE
 millyna/flannel public	0 STARS	0 PULLS	DE

Figure 13: Docker Hub

Therefore, to simplify the future process, all that needs to be done is to pull the relevant Docker image using:

- `docker pull <<IMAGE>>`

4.7 Running Kubernetes

- The first step was setting up the “Master” Pi by bringing up the Docker bootstrap, initially using the Glasgow Raspberry Pi’s project published method. This command created a socket for everything to run on.

```
1. sh -c 'docker -d -H unix:///var/run/docker-bootstrap.sock -p
/var/run/docker-bootstrap.pid --exec-opt
native.cgroupdriver=cgroupfs --ip=127.0.0.1 --iptables=false -
-ip-masq=false --bridge=none --graph=/var/lib/docker-bootstrap
2> /var/log/docker-bootstrap.log 1> /dev/null &'
```

- Once done, the next step was to bring up the ETCD Docker image and bind it to an IP address and storage location.

```
1. docker run --net=host millyna/etcd:latest/bin/etcd --
addr=127.0.0.1:4001 --bind-addr=0.0.0.0:4001 &
```

- With the ETCD container up and storage location set, the next step required reserving an internal network for the cluster to virtualize on. This is done by reserving an IP address range stored within ETCD itself:

```
1. docker run -d --net=host millyna/etcd:latest etcdctl set
/coreos.com/network/config '{ "Network": "10.1.0.0/16" }'
```

- Once running, Docker needed to stop serving the containers to allow for reconfiguring of the network to use the Flannel network service:

1. `systemctl stop docker`

5. To ensure the availability of the network, it was best to run flannel as part of the docker-bootstrap. This command restarted Docker, using the Flannel container as the internal network for the cluster. This would also print out a long string which was related to the container Flannel is being run inside.

1. `docker run -d --net=host --privileged -v /dev/net:/dev/net millyna/flannel flannel`

6. To continue, in order to get the Flannel subnet information from the container for the networking portion, the following command was used:

1. `docker exec <long-string-output-from-last-command> cat /run/flannel/subnet.env`

This output the Flannel subnet information, as shown below:

1. `FLANNEL_SUBNET=10.1.78.1/24`
2. `FLANNEL_MTU=1472`
3. `FLANNEL_IPMASQ=false`

7. Now Flannel was configured and the Docker container sets were running, Docker needed to be reconfigured to use the new network rather than the one supplied via DHCP.

1. `nano /usr/lib/systemd/system/docker.service`

8. Once open, the next step was to locate the line which starts with “ExecStart”, and alter it to include the above network information:

1. `ExecStart=/usr/bin/docker -bop=FLANNEL_SUBNET -mtu=FLANNEL_MTU -d -H fd://`

9. At this point, to bring both Docker and Flannel networks back up, the system commands needed to be used to take down the interface and restart Docker:

1. `/sbin/ifconfig`
2. `docker0 down`
3. `brctl delbr docker0`
4. `systemctl restart docker`

As this was initially a long process, development was done to automate this process as previously stated.

4.8 Configuring Master and Worker Services for Kubernetes

To manually build the “master service”, first, a new file was created with nano:

1. `nano /usr/lib/systemd/system/k8-master.service`

Then the required script was added. (Shown in Appendices 8.2.5: Master Service).

To manually build the “worker service”, just as with the master service, it required creating a new file with nano:

```
1. nano /usr/lib/systemd/system/k8-worker.service
```

Then adding the required script. (Shown in Appendices 8.2.6: Worker Service).

Once this was complete, these services were then used with the “Run Kube” script discussed in chapter 4.9 of the report.

4.9 Automating Cluster Deployment

With more development being done by the Google project team on Kubernetes, it became apparent that attempting to run the cluster by hand each time it needed to be restarted or another node being added would become time consuming and prone to errors, with more and more Docker containers being required along with configuration.

To easily enable a master or worker node, a run-kube bash script was developed, allowing the typing of simple commands which would then automate the deployment process. The script was placed in the bin folder to ensure that universal access to the commands were possible. (Shown in Appendices 8.2.8: Run Kube).

The “Run-Kube” script allows for three simple commands, “run-kube start-master”, “run-kube start-worker” and “run-kube stop-node”. Error checking to ensure that all required images are available is used throughout, along with simplifications ensuring that Docker is working through the system-docker.service and starting the etcd and Flannel services for networking purposes where required by each node.

Placing these scripts inside of the respective folders when creating a new SD card for a Raspberry Pi allows to quickly add extra nodes to the cluster by permitting the user to set up a base Arch Linux image, update the software once logged in, set the hostname and time-zone and then install the static Docker binary mentioned earlier.

Once these steps were done, simply running “run-kube start-master” would enable the master and all required Docker images, along with the network and “run-kube start-worker” would add a worker to the cluster as long as the master IP address was provided.

This simplified expansion of the cluster allowed for quick node building or rebuilding when issues were encountered or add new nodes if required.

4.10 Kubernetes Dashboard

At the outset of this project, Kubernetes Dashboard did not exist. Through continued development, it became clear to Google that the current Kube-UI dashboard that they had previously implemented would not fulfil their requirements for cluster management and Kubernetes Dashboard was put into development. However, with the first Beta of the Dashboard not scheduled for release until the end of February 2016 and at the time, the project team had not considered any ARM based versions of their software. Therefore, a large amount of project time was dedicated to migrating the open source code onto the ARM infrastructure.

This involved a large amount of trial and error, locating dependencies and where required, building these by hand for ARM. This led to the publication of a blog post which is discussed later in chapter 5.1 of the report.

4.11 Installing Kubernetes Dashboard Prerequisites

To configure and run the Kubernetes Dashboard from source, with Docker already installed, the first step to get the Kubernetes Dashboard working includes installing all the software dependencies required and configuring them first. The dependencies that need to be installed are;

- Go 1.5+
- NodeJS 4.2.2+
- NPM 1.3+
- Etcd 2.2+
- Java 7+
- Gulp 3.9+
- Python
- Python2
- Make
- Bower
- GCC

These can be installed with a simple command:

```
1. pacman -S nodejs npm jdk7-openjdk gulp python python2 make  
   bower gcc
```

When testing the later versions of Java, there were issues with compiling the dashboard, therefore it was decided to use JDK 7.

4.12 Building Go

At the time of building the Dashboard for ARM, there was no working version of Go 1.5 for ARMv7. To get around this, it meant firstly installing the older Go version 1.4.3 from the public repository and then manually building 1.5 on an ARMv7 platform. This was later replaced with GO 1.6, but the process remains the same.

1. The first step is to clone the “Go” repository.

```
1. git clone https://go.googlesource.com/go
```

2. Change the directory to “Go”

```
1. cd go
```

3. Checkout “Go 1.4.3”

```
1. git checkout go1.4.3
```

4. Change the directory to “src”

```
1. cd src
```

5. Then run the bash script in order to build Go 1.4.3

```
1. ./all.bash
```

6. As Go is installed in the home directory, to reference this it needed to be added to the \$PATH variable.

1. `nano /home/alarm/.bashrc`

Then input at the bottom of the file:

1. `export GOROOT=$HOME/go`
2. `export PATH=$PATH:$GOROOT/bin`
7. The next step is to log out and back in to the shell as user “alarm”, not “root”.
8. Check that “Go” has installed the version correctly and the \$PATH variable has been setup. It returns the version as shown in figure 14.

1. `go version`

```
[alarm@KubeMaster ~]$ go version
go version go1.4.3 linux/arm
[alarm@KubeMaster ~]$
```

Figure 14: Go Version

9. In order to check that “Go” is not installed and working corrected, a simple “Hello World” script was created.

1. `nano helloWorld.go`

Then input the following into the file:

1. `package main`
- 2.
3. `import "fmt"`
4. `func main() {`
5. `fmt.Printf("hello, world\n")`
6. `}`

Once this is done, save and exit the file. This should return “hello, world” as shown in figure 15. If this does not display and an error shows stating “Go not found”, then a re-check of the \$PATH variable is needed from step 6.

```
[alarm@KubeMaster ~]$ go run helloWorld.go
hello, world
[alarm@KubeMaster ~]$
```

Figure 15: Hello World

4.13 Upgrading GO using Bootstrap and Source

With Go 1.4.3 installed, it needs to be upgraded to Go 1.6 for the Kubernetes Dashboard. This can be done in either two ways, by creating a customer bootstrap, which another machine is needed for, or thanks to the work of Dave Cheney, he has published a public bootstrap for ARM on his website. To save on time, considering the constraints so far, it was decided to use the published bootstrap for this process. To use the published bootstrap, the commands can either be run one by one, however, a customer script was created and run. (Shown in Appendices 8.2.9: Upgrade Go).

10. Once the script has been created and saved, the next step was to give it execute permissions:

1. `chmod +x upgradeGo`

11. The script needed to be run as the “root” user. This was a time-consuming process, and does not update on progress at times. It also failed on some of the tests but once it completed, running `/root/go/bin/go version` returned “go version go1.5 linux/arm”.

1. `./upgradeGo`

12. As the script was running at the “root” user, Go needed to be moved from the root folder it was built in, to the alarm home folder. Firstly, Go 1.4.3 was removed from the home directory of alarm.

1. `rm -rf /home/alarm/go`

13. Secondly, the folder needed to be moved from `root/go` to `alarm/go`.

1. `mv /root/go /home/alarm/go`

14. Finally, a check was made to ensure the folder had been moved correctly:

1. `ls /home/alarm/go`

If the folder had been moved correctly, it should show the contents of the folder, as shown in figure 16.

```
[root@KubeMaster alarm]# ls /home/alarm/go
AUTHORS      CONTRIBUTORS  PATENTS     VERSION     bin  favicon.ico  misc  robots.txt  test
CONTRIBUTING.md LICENSE      README.md   api         doc  lib         pkg   src
```

Figure 16: Go Folder

In order to see if it was still running correctly, a test of the “hello world” was done, and a recheck on the Go version, to ensure it has changed correctly to Go 1.6.

With the full preparation done, the Dashboard could now be built.

4.14 Building the Dashboard

1. For security reasons, the Dashboard needs to be run as a standard user, not as root. Therefore, starting as alarm, the latest Git release for Dashboard was cloned.

1. `git clone https://github.com/kubernetes/dashboard.git`

2. Change to the dashboard directory:

1. `cd dashboard`

3. The next step is to install the dashboard’s packages. This took around 3 hours and required the terminal to stay active throughout the process. If the terminal became inactive or a remote shell is lost, the process had to be restarted.

1. `npm install`

The issue faced with this was an error being thrown back regarding “node sass”, to get around this, node sass was manually downloaded once the NPM was finished. This took an estimated time of around 15 minutes.

1. `npm install node-sass`

4. The next step was to log in as the root user and install bower. This had to be done as root, but once installed, root needed to be exited.

1. `bower install --allow-root`

5. Now, it was ready to run the dashboard.

1. `gulp serve`

Once complete, the terminal will show access URLs as shown in figure 17.

This displays the test/debug version of the Kubernetes dashboard. Accessed by going to the UI address listed in the terminal i.e. `http://<<IP ADDRESS>>:3001`

```
alarm@kubemaster dashboard$ gulp serve
22:37:35] Requiring external module babel-core/register
22:38:10] Using gulpfile ~/dashboard/gulpfile.babel.js
22:38:11] Starting 'package-backend-source'...
22:38:11] Starting 'kill-backend'...
22:38:11] Finished 'kill-backend' after 1.01 ms
22:38:11] Starting 'scripts'...
22:38:12] Starting 'styles'...
22:38:36] Finished 'scripts' after 25 s
22:38:36] Finished 'package-backend-source' after 26 s
22:38:36] Starting 'backend'...
22:38:36] Finished 'styles' after 24 s
22:38:36] Starting 'index'...
22:38:37] Finished 'index' after
22:38:37] Starting 'watch'...
22:38:38] Finished 'watch' after
22:43:56] Finished 'backend' after 5.32 min
22:43:56] Starting 'spawn-backend'...
22:43:56] Finished 'spawn-backend' after 31 ms
22:43:56] Starting 'serve'...
RROR: logging before flag.Parse: 10109 22:43:56.172685 13417 dashboard.go:38] Starting HTTP server on port 9091
22:43:56] Finished 'serve' after
BS] [BrowserSync SPA] Running...
BS] Access URLs:
-----
Local: http://localhost:9090/
External: http://192.168.1.201:9090/
-----
UI: http://localhost:3001
UI External: http://192.168.1.201:3001
-----
BS] Serving files from: /home/alarm/dashboard/.tmp/serve
BS] Serving files from: /home/alarm/dashboard/src/app/frontend
BS] Serving files from: /home/alarm/dashboard/src/app
```

Figure 17: Dashboard - Gulp Serve

In order to remove the debugging console and run a production version of the Kubernetes dashboard the following commands were needed:

1. `gulp build`

Once the dashboard is built, it is placed within the dist folder.

To run the production version of Kubernetes the following command needed to be used:

1. `gulp serve:prod`

A log similar to that seen in figure 16, will appear. Within this the UI port had changed. i.e. `http://<<IP ADDRESS>>:9090`.

The dashboard was now up and running, shown in figure 18.

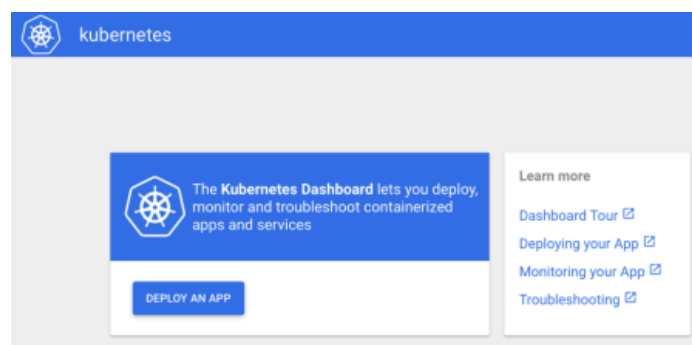


Figure 18: Kubernetes Dashboard

4.15 Running Dashboard using Screen

At this point of the project, an active terminal session was required to run the Kubernetes Dashboard. In order to bypass this, “Screen” was used. Screen runs a detached non-active terminal in the background, which meant there was no need for an active running terminal. In order to do this, screen needs to be installed to the Pi.

```
1. pacman -S screen
```

Once installed, to use screen to run our gulp serve:prod dashboard, the following command is required.

```
1. screen -fa -d -m gulp serve:prod
```

To view what is happening on the detached terminal, the following command can be used

```
1. screen -r
```

In order to detach the screen session, the following key press sequence must be used:

Hold ctrl key then press and release a, followed by pressing and releasing d.

4.16 Dashboard as a Docker Image

Through further collaboration with the Google project team, it became possible to build the Dashboard into a Docker image on the ARM architecture, as outlined in a second blog post published on the Glasgow Pi Project website.

Due to this collaboration and changes to the Dashboard source code, it became possible to build the Dashboard into an ARMv7 compatible Docker image. By doing so, this removed the requirement of an active terminal and enabled easier packaging and distribution for the ARM architecture.

In order to build the dashboard into a Docker image, the latest version of Kubernetes Dashboard source code is required from the GitHub repository. This can be cloned to the machine by running the following command inside of the Dashboard directory:

```
1. git pull https://github.com/kubernetes/dashboard.git
```

This command downloaded and checked the latest files against the current installed files and updated them accordingly.

The next step was to build and run the Docker image:

```
1. gulp docker-image
```

Then check that the Docker image has been built and entered into the local Docker images repository using the following command, as shown in figure 19:

```
1. docker images
```

```
[root@kube-master dashboard]# docker images
REPOSITORY          TAG          IMAGE ID          CREATED          VIRTUAL SIZE
kubernetes/dashboard latest       b546c53bbe91     14 seconds ago  25.73 MB
```

Figure 19: Docker Images

Lastly, to run the Docker image, the following command was used:

```
1. docker run -d -p :9090:9090 kubernetes/dashboard -apiserver-  
host=http://192.168.1.201:8080
```

The Dashboard was now running in a tiny docker image, without the need for “screen” or an active running terminal.

This now allowed the publication of the Docker image to the Docker hub, making the Dashboard available to other members of the ARM community without the need to install the Dashboard prerequisites and build from source.

4.16.1 Further Development

Towards the end of this project, through collaboration with the Google Project team, Google now compile an ARM Docker image on each release. Therefore, this pre-built image can now be downloaded and run inside of Kubernetes as a “sublet” through use of a YAML instruction file.

This requires creating a file:

```
1. nano dashboard.yaml
```

Entering the required data. (Shown in Appendices 8.2.10: Dashboard).
Finally, running with:

```
1. kubectl create -f dashboard.yaml
```

This makes the dashboard accessible by going to the node’s IP address using port 8080/ui.
i.e. <http://<<IP-ADDRESS>>:8080/ui>

4.17 Criu and P. Haul

As the initial port of Dashboard was completed and awaiting further development by Google, with just less than four months remaining on the project, the decision was taken to extend the project further. The area of extension was agreed to be container live migration, as discussed in section 2.9 of the report, on the ARMv7 architecture.

As there was a lack of documentation regarding Criu and P. Haul. Discussions with the developers through their mailing list led to the following list being revised to create a custom kernel configuration, the revised list was then shared with the Criu developers who have since updated their website to include all the prerequisites, as shown below:

“General setup options

- CONFIG_CHECKPOINT_RESTORE=y (Checkpoint/restore support)
- CONFIG_NAMESPACES=y (Namespaces support)
- CONFIG_UTS_NS=y (Namespaces support -> UTS namespace)
- CONFIG_IPC_NS=y (Namespaces support -> IPC namespace)
- CONFIG_PID_NS=y (Namespaces support -> PID namespaces)
- CONFIG_NET_NS=y (Namespaces support -> Network namespace)
- CONFIG_FHANDLE=y (Open by fhandle syscalls)
- CONFIG_EVENTFD=y (Enable eventfd() system call)
- CONFIG_EPOLL=y (Enable eventpoll support)

Networking support -> Networking options options for sock-diag subsystem

- CONFIG_UNIX_DIAG=y (Unix domain sockets -> UNIX: socket monitoring interface)
- CONFIG_INET_DIAG=y (TCP/IP networking -> INET: socket monitoring interface)
- CONFIG_INET_UDP_DIAG=y (TCP/IP networking -> INET: socket monitoring interface -> UDP: socket monitoring interface)
- CONFIG_PACKET_DIAG=y (Packet socket -> Packet: sockets monitoring interface)
- CONFIG_NETLINK_DIAG=y (Netlink socket -> Netlink: sockets monitoring interface)

Other options

- CONFIG_INOTIFY_USER=y (File systems -> Inotify support for userspace)” (CRIU, 2016)

Due to the requirement to download and build Criu with each kernel change, in order to simplify things, two bash scripts were made to automate the processes. These scripts were based on the work done by hand in order to discover the required dependencies along with altering the stored location of various Python modules required by Criu and P. Haul. (Shown in Appendices 8.2.11: getCriuDeps and Appendices 8.2.12: getCriu).

The first script had to be run as root:

1. getCriuDeps

Then the second script as standard user, this will clone the criu and p. haul repository, then build and install both programs from the source code:

1. getCriu

Lastly, once the previous steps were complete, the p.haul/service and p.haul/wrap files needed their first line to be changed from:

1. #!/usr/bin/env python

to:

1. #!/usr/bin/env python2

4.18 Implementing Dashboard Resource Monitoring

Towards the end of the project, the Kubernetes Dashboard implemented resource monitoring through the use of Heapster. As an ARM version of Heapster was not made available, the Heapster binary had to be built from scratch in order to get it working, as well as the influx dB and grafana binaries used for collecting and storing the data from all nodes. The building of binaries and Docker images was discussed earlier in section 4.2.2 of the report. Once the docker image was available, it was a simple command with “kubect!” which simplified the addition and management.

In order to manually build Heapster, a new Kubernetes YAML file was created with nano:

1. nano heapster.yaml

Then adding the required data. *[Shown in section 9.7 of the appendices]*
Lastly, running with the command:

```
2. kubectl create -f heapster.yaml
```

Once both Heapster and the Dashboard have been deployed, the resource usage can be seen for each running container through the Dashboard web page, this is shown in figure 20.

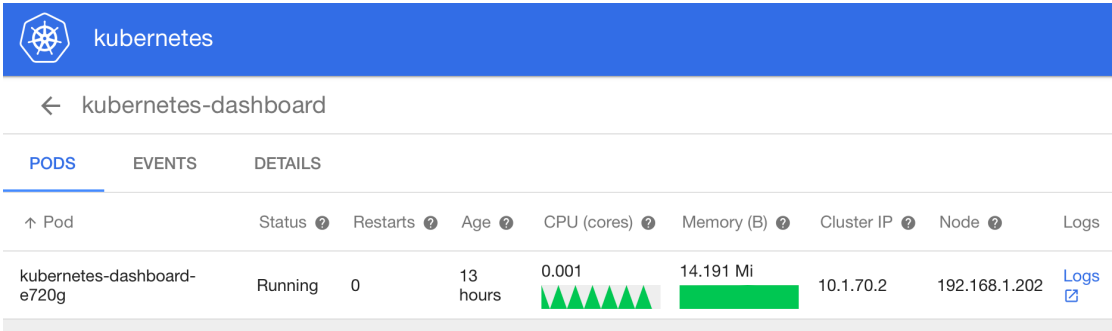


Figure 20: Kubernetes Dashboard: Heapster Resource Usage

5 Evaluation

The original work done by the Glasgow Raspberry Pi Cloud team has now been improved by updating the original versions of software, with the change in methods of simplifying building Docker and Kubernetes on the cluster, as well as extending on this with a new feature which will be utilized on their “PiCloud” cluster.

Kubernetes was the most challenging part of the project to configure. There are many contributors to Kubernetes from the Raspberry Pi community, thus there are many notes and projects out there to help run Kubernetes on the Raspberry Pi.

As the aim of this project was to limit to the operating system to that of Arch Linux ARM, it posed its own difficulties as described in chapter 5 of this report.

To set up Kubernetes at the outset of this project, the steps required were taken from and later improved upon the work done previously by the Glasgow Pi Cloud project which is visible on their blog post online. However, to simplify this work, the two steps regarding Flannel and Etcd were packaged into respective services, which could then be called rather than having to retype full commands by hand. Both services have built-in sanity checkers to ensure that the service is not already running or running in part. If it is, it will kill the process before attempting to spawn a new one. Lastly, a service was required that was dedicated to bringing Docker up or down and ensuring it would not attempt to use any other network configuration. (Shown in Appendices 8.2.2: Etcd Service, Appendices 8.2.3: Flannel Service and Appendices 8.2.4: Docker Service).

When configuring Kubernetes, the simplest way to add a worker node to the cluster would be by using the services from the previous steps listed without the ETCD image to enable a worker, attaching it to the masters Flannel network by feeding it the respective IP address and port number. However, through further development of the Kubernetes system by Google, it became apparent that in order to simplify the deployment of nodes process, more modules were going to be required for each of the master and worker. Therefore, in order to ensure the base modules were easily started, both the master and a worker service was devised. Shown in chapter 4.8 of the report. Again, this was improved upon the work done by the Glasgow Pi project and the end-version includes three extra modules “hyperkube”, “pause” and lastly “Heapster”, which are required by the later versions of Kubernetes to monitor a cluster resources and display them on the Kubernetes Dashboard.

The Kubernetes Dashboard has come a long way since being initially built on ARMv7. At first, there was little functionality, it was just a standard user-interface where the links did not work. Through continued development, the Kubernetes Dashboard has had four major releases, been announced as “production ready” and is now on production version 1.0.1. Many updates and bug fixes have been implemented including building images for ARM platforms. The latest version can be seen in figure 21.

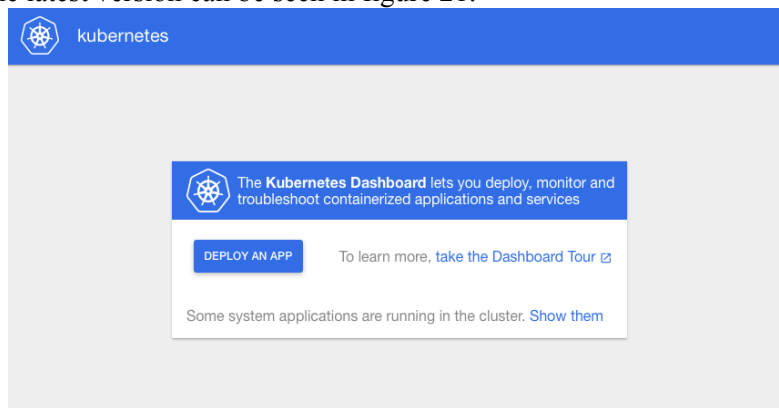


Figure 21: Kubernetes Dashboard - Version 1.0.1

With regards to the attempt on live migrating containers with Criu and P. Haul, this was an extremely time-consuming process of trial and error to discover the kernel configuration as extensive research was required due to the lack of documentation provided by the developers. The Raspberry Pi's had to be rebuilt several times, with attempting to build Kubernetes and the Dashboard on Hypriot OS and Raspbian OS, to see if the checkpoint and restore software would work on these OS's. The checkpoint and restore modules were not due to be updated till the end of March for their new releases, this left little time to develop once these were released, and due to this the progression so far was left at a halt. However, the advantage point of this is that, in theory from the research done and progress made, it will be possible to live migrate containers on a Raspberry Pi using Arch Linux ARM, especially with the upcoming release of Criu 2.0 and the updated P. Haul modules release at the end of March.

Due to the lack of checkpoint and restore functionality within the default Arch Linux kernel, any checkpoint and restore functionality was not immediately available. Therefore, a custom kernel needed to be built with the following changes to the basic configuration. This posed its own issues to the project.

When building the initial images, this was done on the Pi itself. As this took over 20 hours each time to compile, it became apparent that this would be a very time-consuming process. Therefore, the use of an i7 MacBook Pro was enlisted. This led to further research into cross compilation along with locating and downloading pre-built tool chains for the ARM architecture. Secondly, in order to discover the options required within the custom kernel, multiple kernels were required. Once each new kernel was compiled and deployed, rebuilding and testing Criu was required in order to see if all options were now available. Once error logs were available, these then had to be understood to discover any further dependencies or kernel configuration required. This issue was compounded by a lack of documentation by the developers of Criu and resulted in communication and collaboration with the developers in order to discover the full list of the kernel configuration shown in section 4.17 of the report.

The majority of the project's development was focused around the method of "Trial and Error". It was an impossible task to structure this into the project management due to the rolling-updates of all software used within the cluster. Docker had a total of nine different releases, not including any pre-releases, and Kubernetes had a total of eleven releases, again not including any pre-releases, since the start of the project. Each release had bugs, new features, and many changings in the source-code. Due to this, the majority of time was spent going back and forth tweaking various binaries and code to allow for each release to still be compatible with the ARMv7 architecture.

5.1 Contributions

As the Dashboard feature was a "worlds-first" build on ARMv7, the projects supervisor requested a blog post to be added to the Glasgow Raspberry Pi Cloud's word press website. With this came recognition from Google and thus started a working relationship with Piotr Bryk, who is the main Google developer behind the Kubernetes Dashboard. Piotr later stated that he had followed the instructions from the blog post, when another user had pointed that the issue Google were facing could be overcome with the steps and code used by this project. Due to this, the issue Google were facing with running npm install and bower install was fixed and implemented into the Google Kubernetes Dashboards official source-code. This is shown in Figures 22-24.

Since the publication of the Glasgow Pi Cloud Project blog post on the 12th January 2016, various other project websites have copied the steps taken and reproduced this work as their own without due referencing. However, due to the dates of publication and evident throughout the wording of many of these publications, it is obvious that they are direct copy and pastes of the original post located on the Glasgow Pi Cloud project. Throughout a number of these publications there are some slight differences however, such as base OS used, hence

the authors claiming the work as their own without providing reference to the Glasgow Pi Cloud project.

An example of this can be seen at the RPI Cloud guide of installing Kubernetes Dashboard on Hypriot OS by Kasper Nissen published on 1st February 2016. However, this is not the only blog post where it is evident that the author has followed the guide presented by this project on the Glasgow Pi blog.

Further to this, there are various similarities between the methods used within this project to that of the Kubernetes-on-arm project by Lucas Käldestrom, as in the early days collaboration and discussions were ongoing between the two projects and as such some amount of code may be present in both. However, as the Lucas' project was concentrating on using already available binaries provided by the Glasgow Pi project to find a simplified way to enable a Kubernetes cluster to be run on ARM, this project took a different path as it was focused on adding functionality such as the Kubernetes Dashboard. Again, this can be further seen as once Google made an ARM image available for the Kubernetes Dashboard, the Lucas' project has added this as what it deems an "Add-On".

This shows the desire of the Raspberry Pi community to have enterprise software available on their single-board computers, this is a positive point for this project to be the first in the world to publish a working method to get the entire Kubernetes orchestration software including the Dashboard working on ARMv7 architecture, and as such, the traction gained as seen by Google, as it resulted in ARMv7 images being made readily available by Google for anybody to use, rather than just those with extensive programming knowledge. The resulting factor is now that rather than having to follow the blog post and have a high level of programming, the end-user can now deploy the Kubernetes Dashboard with one single line of code.

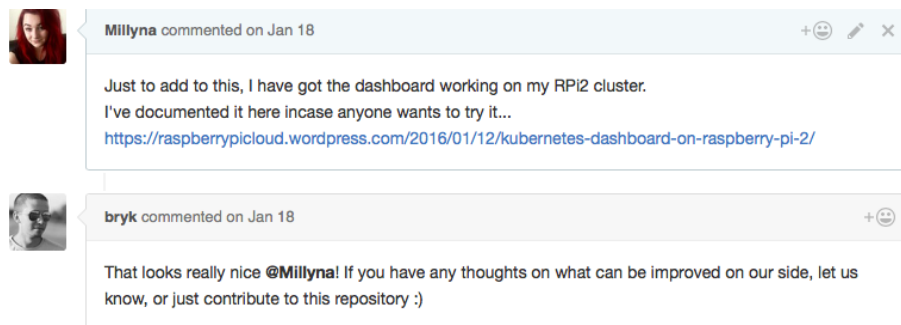


Figure 22: Dashboard GitHub Collaboration

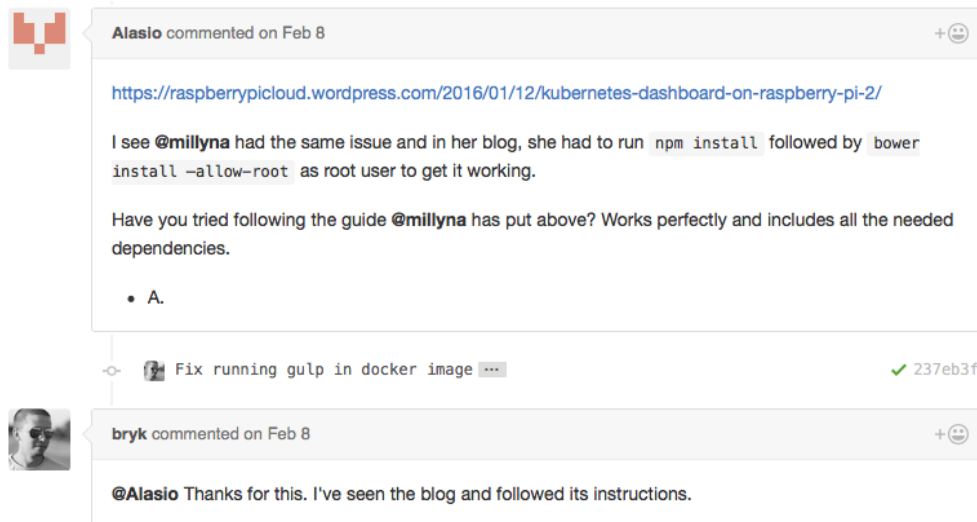


Figure 23: Dashboard Blog Reference

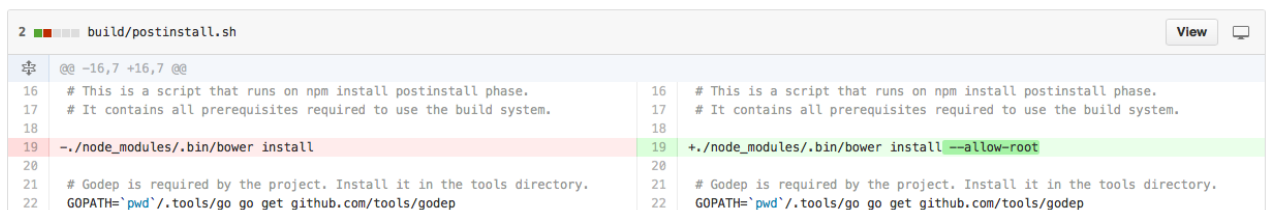


Figure 24: Dashboard source-code change

6 Discussion and Conclusion

To conclude, the entire project was a larger success than initially anticipated, achieving much more than the initial project specification through continued development and meetings with the project supervisor.

As this project focused on further development of the work previously done by the University of Glasgow's Raspberry Pi Cloud project, this enabled the simplification of cluster deployment and management. This was then further extended by incorporating later versions of the Kubernetes software and all of its dependencies, keeping the Glasgow Raspberry Pi Cloud project at the forefront of cloud development on ARM architecture.

This project enabled the building of a new feature of Kubernetes, namely the Dashboard, for ARMv7 architecture before the BETA or production release of the software. At the outset, Google had no intention of developing their software on the ARM architecture, but through persistence of development through source code and the publication of the relevant blog posts, this gained recognition from Google and as such, they took the decision to work in collaboration with this project and others in order to bring Kubernetes to the ARM architectures as easily as it was made available to i86 and x64 architectures.

Although the original scope of the project was to build a 20-node cluster, this was not implemented due to the focus of the project switching from cluster building to cluster management. Further, through the simplification of adding new nodes to the cluster, it was agreed that a 20-node cluster was no longer needed, nor relevant to the project.

Disappointingly, the checkpoint and restore functionality was not able to be built on ARMv7 during the course of this project due to the time-constraints. However, as stated in the evaluation, this should be a possible extension to the work done within this project at a future date. This possibility is further backed up with the recent release of Criu 2.0. However, as there are talks of implementing Criu into Docker as a core feature in the near future, this development may be unnecessary.

On completion of this project, as Docker and Kubernetes are rolling-releases, there will still be much more work to be done on the ARMv7 architecture. As new features are added weekly, continual development will be required to keep the Glasgow Raspberry Pi Project up to date. Also, with ARMv7 being not the primary architecture for the majority of large companies' software development, it will still require work from the ARM community to incorporate and to push for implementation similar to what was done with the Kubernetes Dashboard.

7 References and Bibliography

1. Archlinuxarm.org. (2016). Arch Linux ARM. [online] Available at: <https://archlinuxarm.org/> [Accessed 15 Apr. 2016].
2. Carstensen, J., Morgenthal, J. and Golden, B. (2012). Cloud Computing. Ely: IT Governance Publishing.
3. Computing, H. (2016). How to Use Virtualization with Cloud Computing - For Dummies. [online] Dummies.com. Available at: <http://www.dummies.com/how-to/content/how-to-use-virtualization-with-cloud-computing.html> [Accessed 15 Apr. 2016].
4. Cox, S., Cox, J., Boardman, R., Johnston, S., Scott, M. and O'Brien, N. (2013). Iridispi: a low-cost, compact demonstration cluster. Cluster Computing, 17(2), pp.349-358.
5. Difference?, V. (2014). Virtualization vs. Cloud Computing: What's the Difference?. [online] Business News Daily. Available at: <http://www.businessnewsdaily.com/5791-virtualization-vs-cloud-computing.html> [Accessed 15 Apr. 2016].
6. Docker. (2015). What is Docker?. [online] Available at: <https://www.docker.com/what-docker> [Accessed 15 Apr. 2016].
7. En.citizendium.org. (2016). Beowulf cluster - encyclopedia article - Citizendium. [online] Available at: http://en.citizendium.org/wiki/Beowulf_cluster [Accessed 15 Apr. 2016].
8. Gchq.gov.uk. (2016). GCHQ's Raspberry Pi 'Bramble' - exploring the future of computing. [online] Available at: http://www.gchq.gov.uk/press_and_media/news_and_features/pages/gchqs-raspberry-pi-bramble.aspx [Accessed 15 Apr. 2016].
9. GitHub. (2016). kubernetes/kube-ui. [online] Available at: <https://github.com/kubernetes/kube-ui> [Accessed 15 Apr. 2016].
10. Goasguen, S. (2015). Docker Cookbook. O'Reilly Media.
11. http://en.citizendium.org/images/thumb/a/a0/Simple_Beowulf_Cluster_Diagram.png/250px-Simple_Beowulf_Cluster_Diagram.png, (n.d.). Simple Beowulf Diagram. [image] Available at: http://en.citizendium.org/images/thumb/a/a0/Simple_Beowulf_Cluster_Diagram.png/250px-Simple_Beowulf_Cluster_Diagram.png [Accessed 15 Apr. 2016].
12. Ibm.com. (2016). Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment. [online] Available at: <https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/> [Accessed 15 Apr. 2016].
13. Ibm.com. (2016). Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment. [online] Available at: <https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/> [Accessed 15 Apr. 2016].
14. Kiepert, J. (2013). Creating a Raspberry-Pi Beowulf Cluster. 1st ed. [ebook] Available at: http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry-Pi-Based.Beowulf.Cluster_v2.pdf [Accessed 15 Apr. 2016].
15. Learn Docker in a Day. (2016). .
16. Linuxjournal.com. (2016). Linux Containers and the Future Cloud | Linux Journal. [online] Available at: <http://www.linuxjournal.com/content/linux-containers-and->

- future-cloud [Accessed 15 Apr. 2016].
17. Magazine, P. (2016). Scientists are using Xbox Kinect to let orangutans play video games. [online] PC Tech Magazine. Available at: <http://pctechmag.com/2016/02/scientists-are-using-xbox-kinect-to-let-orangutans-play-video-games/> [Accessed 15 Apr. 2016].
 18. McCallion, J. (2016). Raspberry Pi and Lego cloud cooked up by Glasgow University. [online] IT PRO. Available at: <http://www.itpro.co.uk/iaas/19989/raspberry-pi-and-lego-cloud-cooked-glasgow-university> [Accessed 15 Apr. 2016].
 19. Monk, S. (n.d.). Raspberry Pi cookbook.
 20. Mouat, A. (n.d.). Using docker.
 21. National Institute of Standards and Technology, D. (2011). The NIST Definition of Cloud Computing. 1st ed. [ebook] Available at: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf> [Accessed 15 Apr. 2016].
 22. Nazario, K., Suleman, K. and Shepherd, A. (2016). Raspberry Pi: Top 20 projects to try yourself. [online] IT PRO. Available at: <http://www.itpro.co.uk/mobile/21862/raspberry-pi-top-20-projects-to-try-yourself> [Accessed 15 Apr. 2016].
 23. Negus, C. (n.d.). Docker containers.
 24. Opensource.com. (2016). What is a Raspberry Pi?. [online] Available at: <https://opensource.com/resources/what-raspberry-pi> [Accessed 15 Apr. 2016].
 25. PA Consulting Group. (2016). Raspberry Pi competition 2015. [online] Available at: <http://www.paconsulting.com/events/raspberry-pi-competition/> [Accessed 15 Apr. 2016].
 26. Pluralsight.com. (2016). Virtualization 101: What is a Hypervisor?. [online] Available at: <https://www.pluralsight.com/blog/it-ops/what-is-hypervisor> [Accessed 15 Apr. 2016].
 27. Raspberry Pi Foundation, (2016). Raspberry Pi 2 Model B. [image] Available at: https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_-comp.jpeg [Accessed 15 Apr. 2016].
 28. Suleman, K. (2016). Raspberry Pi Awards 2014 winners revealed. [online] IT PRO. Available at: <http://www.itpro.co.uk/mobile/21981/raspberry-pi-awards-2014-winners-revealed> [Accessed 15 Apr. 2016].
 29. Techopedia.com. (2016). What is Computer Cluster? - Definition from Techopedia. [online] Available at: <https://www.techopedia.com/definition/6581/computer-cluster> [Accessed 15 Apr. 2016].
 30. Virtualization and Cloud Computing. (2013). 1st ed. [ebook] Available at: <http://www.intel.co.uk/content/dam/www/public/us/en/documents/guides/cloud-computing-virtualization-building-private-iaas-guide.pdf> [Accessed 15 Apr. 2016].
 31. Virtuatopia.com. (2016). An Overview of Virtualization Techniques - Virtuatopia. [online] Available at: http://www.virtuatopia.com/index.php/An_Overview_of_Virtualization_Techniques [Accessed 15 Apr. 2016].
 32. Wikipedia. (2016). LXC. [online] Available at: <https://en.wikipedia.org/wiki/LXC> [Accessed 15 Apr. 2016].

33. Zimara, S. (2013). The Five Essential Characteristics of Cloud Computing - ERPBloggers. [online] ERPBloggers. Available at: <http://erpbloggers.com/2013/07/the-five-essential-characteristics-of-cloud-computing/> [Accessed 15 Apr. 2016].
34. Anderson, T. (n.d.). Live Migration of Linux Containers. 1st ed. [ebook] Available at: <https://opensource.com/resources/what-is-kubernetes> [Accessed 15 Apr. 2016].
35. Anon, (2016). [online] Available at: <https://aws.amazon.com/docker/> [Accessed 15 Apr. 2016].
36. Archlinux.org. (2016). Arch Linux - docker 1:1.10.3-1 (x86_64). [online] Available at: https://www.archlinux.org/packages/community/x86_64/docker/ [Accessed 15 Apr. 2016].
37. Archlinuxarm.org. (2016). Raspberry Pi | Arch Linux ARM. [online] Available at: <https://archlinuxarm.org/platforms/armv6/raspberry-pi> [Accessed 15 Apr. 2016].
38. Archpi.databse.com. (2016). Archlinux on Raspberry PI2e10ab65. [online] Available at: <http://archpi.databse.com> [Accessed 15 Apr. 2016].
39. Balakrishnan, N. (2012). Building and benchmarking a low power ARM cluster. PhD. The University of Edinburgh.
40. Cheney, D. (2012). Installing Go on the Raspberry Pi | Dave Cheney. [online] Dave.cheney.net. Available at: <http://dave.cheney.net/2012/09/25/installing-go-on-the-raspberry-pi> [Accessed 15 Apr. 2016].
41. Cheney, D. (2015). Bootstrapping Go 1.5 on non Intel platforms | Dave Cheney. [online] Dave.cheney.net. Available at: <http://dave.cheney.net/2015/10/16/bootstrapping-go-1-5-on-non-intel-platforms> [Accessed 15 Apr. 2016].
42. Cheney, D. (2015). Building Go 1.5 on the Raspberry Pi | Dave Cheney. [online] Dave.cheney.net. Available at: <http://dave.cheney.net/2015/09/04/building-go-1-5-on-the-raspberry-pi> [Accessed 15 Apr. 2016].
43. Comparing Public, a. (2016). Comparing Public, Private, and Hybrid Cloud Computing Options - For Dummies. [online] Dummies.com. Available at: <http://www.dummies.com/how-to/content/comparing-public-private-and-hybrid-cloud-computin.html> [Accessed 15 Apr. 2016].
44. Coreos.com. (2016). CoreOS. [online] Available at: <https://coreos.com/flannel/docs/latest/> [Accessed 15 Apr. 2016].
45. Coreos.com. (2016). CoreOS. [online] Available at: <https://coreos.com/etcd/> [Accessed 15 Apr. 2016].
46. Ctl.io. (2016). What is Docker and When to Use It - CenturyLink Cloud Developer Center. [online] Available at: <https://www.ctl.io/developers/blog/post/what-is-docker-and-when-to-use-it/> [Accessed 15 Apr. 2016].
47. Docker. (2015). What is Docker?. [online] Available at: <https://www.docker.com/what-docker> [Accessed 15 Apr. 2016].
48. Docker.com. (2016). eBay Simplifies Application Deployment | Docker. [online] Available at: <https://www.docker.com/customers/ebay-simplifies-application-deployment> [Accessed 15 Apr. 2016].
49. Docker.com. (2016). GE Uses Docker to Enable Self-Service For Their Developers | Docker. [online] Available at: <https://www.docker.com/customers/ge-uses-docker-enable-self-service-their-developers> [Accessed 15 Apr. 2016].
50. Felter, W., Ferreira, A., Rajamony, R. and Rubio, J. (2014). An Updated Performance Comparison of Virtual Machines and Linux Containers. [online] Available at: [http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf).
51. Freedesktop.org. (2016). systemd. [online] Available at: <https://www.freedesktop.org/wiki/Software/systemd/> [Accessed 15 Apr. 2016].
52. GitHub. (2016). docker/docker. [online] Available at: <https://github.com/docker/docker> [Accessed 15 Apr. 2016].

53. GitHub. (2016). google/cadvisor. [online] Available at: <https://github.com/google/cadvisor> [Accessed 15 Apr. 2016].
54. GitHub. (2016). kubernetes/dashboard. [online] Available at: <https://github.com/kubernetes/dashboard> [Accessed 15 Apr. 2016].
55. GitHub. (2016). kubernetes/kubernetes. [online] Available at: <https://github.com/kubernetes/kubernetes> [Accessed 15 Apr. 2016].
56. GitHub. (2016). luxas/kubernetes-on-arm. [online] Available at: <https://github.com/luxas/kubernetes-on-arm> [Accessed 15 Apr. 2016].
57. GitHub. (2016). Millyna/ARM-Criu. [online] Available at: <https://github.com/Millyna/ARM-Criu> [Accessed 15 Apr. 2016].
58. Godoc.org. (2016). Package hyperkube. [online] Available at: <https://godoc.org/k8s.io/kubernetes/pkg/hyperkube> [Accessed 15 Apr. 2016].
59. Golanghub.com. (2014). Comparison Type 1 vs Type 2 Hypervisor ~ GoLinuxHub. [online] Available at: <http://www.golanghub.com/2014/07/comparison-type-1-vs-type-2-hypervisor.html> [Accessed 15 Apr. 2016].
60. Holub, (2015). Docker Use Cases - Container Solutions. [online] Container Solutions. Available at: <http://container-solutions.com/docker-use-cases/> [Accessed 15 Apr. 2016].
61. iRomin. (2015). Docker Use Cases. [online] Available at: <https://rominirani.com/2015/04/09/docker-use-cases/> [Accessed 15 Apr. 2016].
62. Linuxacademy.com. (2016). Linux Commands For Beginners: SUDO – Linux Academy Blog. [online] Available at: <https://linuxacademy.com/blog/linux/linux-commands-for-beginners-sudo/> [Accessed 15 Apr. 2016].
63. Linuxfoundation.org. (2016). What is Linux. [online] Available at: <http://www.linuxfoundation.org/what-is-linux> [Accessed 15 Apr. 2016].
64. Moore, J. (2014). Arch Linux a different type of Linux. [online] Lions-wing.net. Available at: <http://www.lions-wing.net/lessons/arch-linux/arch.html?&session-id=1cd567ae3aec696e3faf2528701e4b14> [Accessed 15 Apr. 2016].
65. Opensource.com. (2016). What is Kubernetes?. [online] Available at: <https://opensource.com/resources/what-is-kubernetes> [Accessed 15 Apr. 2016].
66. raspberrypicloud. (2015). HOW TO: Kubernetes Multi-node on Raspberry Pi 2s. [online] Available at: <https://raspberrypicloud.wordpress.com/2015/08/11/how-to-kubernetes-multi-node-on-raspberry-pi-2s/> [Accessed 15 Apr. 2016].
67. raspberrypicloud. (2016). Kubernetes Dashboard – Run as Docker Image on Raspberry Pi 2.. [online] Available at: <https://raspberrypicloud.wordpress.com/2016/01/19/kubernetes-dashboard-run-as-docker-image-on-raspberry-pi-2/> [Accessed 15 Apr. 2016].
68. raspberrypicloud. (2016). Kubernetes Dashboard on Raspberry Pi 2. [online] Available at: <https://raspberrypicloud.wordpress.com/2016/01/12/kubernetes-dashboard-on-raspberry-pi-2/> [Accessed 15 Apr. 2016].
69. raspberrypicloud. (2016). raspberrypicloud. [online] Available at: <https://raspberrypicloud.wordpress.com/blog/> [Accessed 15 Apr. 2016].
70. SearchCloudApplications. (2016). What is cloud orchestrator? - Definition from WhatIs.com. [online] Available at: <http://searchcloudapplications.techtarget.com/definition/cloud-orchestrator> [Accessed 15 Apr. 2016].
71. SearchServerVirtualization. (2016). Virtualization hypervisor comparison: Type 1 vs. Type 2 hypervisors. [online] Available at: <http://searchservirtualization.techtarget.com/tip/Virtualization-hypervisor-comparison-Type-1-vs-Type-2-hypervisors> [Accessed 15 Apr. 2016].
72. SearchServerVirtualization. (2016). What is virtual machine? - Definition from WhatIs.com. [online] Available at: <http://searchservirtualization.techtarget.com/definition/virtual-machine> [Accessed 15 Apr. 2016].

73. SearchServerVirtualization. (2016). What's the difference between Type 1 and Type 2 hypervisors?. [online] Available at: <http://searchservervirtualization.techtarget.com/feature/Whats-the-difference-between-Type-1-and-Type-2-hypervisors> [Accessed 15 Apr. 2016].
74. The Virtualization Practice. (2015). Container Use Cases. [online] Available at: <https://www.virtualizationpractice.com/container-use-cases-35048/> [Accessed 15 Apr. 2016].
75. Wiki.archlinux.org. (2016). Git - ArchWiki. [online] Available at: <https://wiki.archlinux.org/index.php/git#Installation> [Accessed 15 Apr. 2016].
76. Wiki.archlinux.org. (2016). pacman - ArchWiki. [online] Available at: <https://wiki.archlinux.org/index.php/pacman> [Accessed 15 Apr. 2016].
77. Wiki.archlinux.org. (2016). Raspberry Pi - ArchWiki. [online] Available at: https://wiki.archlinux.org/index.php/Raspberry_Pi [Accessed 15 Apr. 2016].
78. CRIU. (2016). Installation. [online] Available at: <https://criu.org/Installation> [Accessed 15 Apr. 2016].

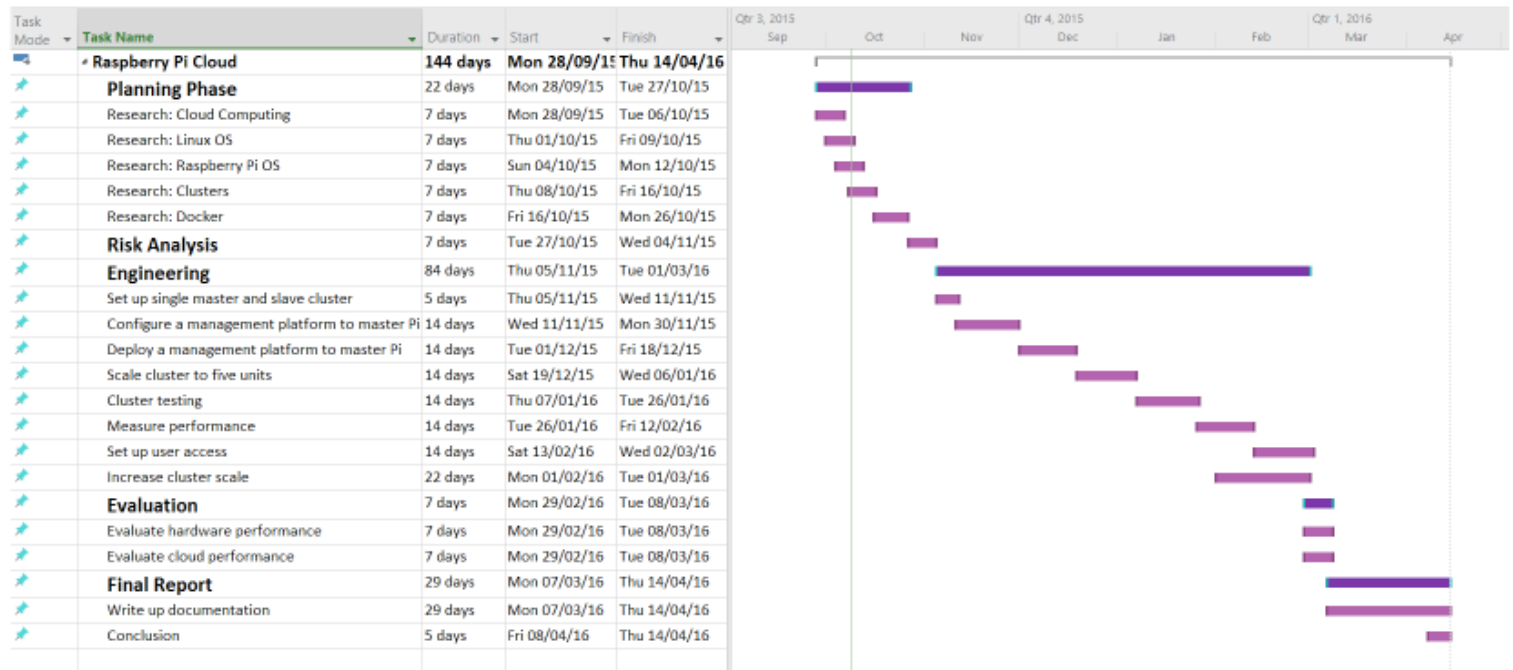
8 Appendices

8.1 Project Management

8.1.1 Initial Project Specification

Subjects to be studied.	I will be building a miniature cloud using 15-20 raspberry pi 2's. These will be set up as a cluster and will require management and provisioning through a cluster virtualisation platform such as LXC/Docker. I will also look to use cluster management tools such as Shipyard, Ansible and Kubernetes. This also requires the use of Arch Linux and the command line/terminal.
A brief description of the background.	The reason I chose to do this as my final year project is that I am interested in the 'Cloud' and would like to expand my knowledge on this subject. The cloud is now used by the majority of people with internet access, with many businesses and personal users now using 'Software as a Service' (SaaS) in everyday use. One popular example of SaaS is Office 365. Doing this project will enable me to further my knowledge and skills with Linux. This will increase my employability when applying for graduate positions as I will have experience in cluster computing, cloud architecture and management and multiple operating platforms.
Problems to be addressed.	There are many problems associated with cloud resource management which need to be addressed. The biggest problem facing me is that there are no existing management tools for clusters constructed with devices using the ARMv7 processor architecture. Therefore, I plan on testing and deploying a number of possible tools to establish which tools will work on the platform, their stability and performance. I plan on testing a variety of different cluster management software and benchmarking tools. At present, none of these have a native ARMv7 binary, but there are several community projects to configure these to the ARMv7 platform, so I will have to manually reconfigure and build these from the binary sources available. This will be time consuming and require the largest amount of time during this project.
Aims, milestones and initial ideas.	<p>I will be using the spiral model as a guide for my project milestones, documented through a Gantt chart, shown below. The Gantt chart will be modified in each stage to show any additional tasks done.</p> <p>The first milestones are as follows:</p> <ul style="list-style-type: none"> • Setting up a single master and slave Cluster using 2x Raspberry Pi 2's. • Configuring and deploying a management platform to the master Pi. • Scaling the cluster from two to five units and testing. • Measuring the performance of the Pi Cloud.
Software and hardware constraints. Are they available within the School of Computing and Maths?	<p>To complete this project, I will need access to:</p> <ul style="list-style-type: none"> - 20 x Raspberry Pi 2 Model B - A 32 Port Ethernet Switch with Bridge – 1000Base-T - A number of powered USB hubs to supply 5v / 2A power to the 20 Raspberry Pi 2's - 20 x USB to Micro USB cable - 20 x RJ45 network cables with TIA/EIA 568B Wiring - 20 x 8GB(minimum) high-speed micro-SD cards - Lego

8.1.2 Gantt Chart



8.1.3 November Monthly Report



6000PROJ / 6001PROJ Final Year Project Supervision Meeting Record

Student: Heather

McWha Date: 04/11/2015

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">• Setting up Pi to Pi Communication.• Installing Arch Linux ARM and Docker to each Pi• Research and understanding ARM architecture.• Research and understanding Docker.• Literature Review started: Cloud Computing, Raspberry Pi, Operating Systems, Cluster Computing.• Installed Kubernetes onto the Raspberry Pi 2's using a github source: luxus/kubernetes-on-arm.• Hack/Play with Kubernetes.• Experimenting with Containers/LXC• Research on Orchestration tools: Nomad, Ansible, Kubernetes, Shipyard, Swarm, Fleet.
Recommended actions
<ul style="list-style-type: none">• Complete write up on background, scope and objectives.• Work with NoMad/Fleet to see if can work with ARM on the Pi• Increase Cluster size from 2 to 5.• Have Master and Slave Pi's configured.
Deliverables for next time
Other comments

Supervisor signature: P Tso.....

Student signature: Heather McWha.....

8.1.4 January Monthly Report



6000PROJ / 6001PROJ Final Year Project Supervision Meeting Record

Student: Heather McWha.....Date: 20/01/16

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">• Kubernetes Dashboard now fully working and up and running on the <u>RPI</u> Cluster.• Discusses of further technical aspects of the project such as <u>P.Haul</u>, a live migration tool.• Blog post made on the Raspberry Pi Glasgow project blog• Created own standalone images for Kubernetes to run (Was previously using Kubernetes-on-arm project from <u>github</u> to run Kubernetes, now not using)• Dashboard now running as a Docker image.
Recommended actions
<ul style="list-style-type: none">• <u>P.Haul</u> researched and try implementing this onto ARM.• Start full write up of final year project.• Research ETCD stability as it keeps failing when running a long period of time.
Deliverables for next time
<ol style="list-style-type: none">1. Feasibility study of container/docker live migration.2. Complete Literature Review section.
Other comments

Supervisor signature: Posco Tso

Student signature: Heather McWha

8.1.5 February Monthly Report



6000PROJ / 6001PROJ Final Year Project Supervision Meeting Record

Student: Heather McWha.....Date: 26/02/16

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">• Criu and <u>P.Haul</u> running on Arch Linux• Checkpoint/Restore is not compatible with Arch Linux• Created custom kernel for Arch Linux to try enable compatibility for Criu• Increased cluster size to 4• Kept up with Dashboard developments• Released 2 blog posts on Raspberry Pi – University of Glasgow Blog• Recognition from Google development team for Kubernetes Dashboard https://github.com/kubernetes/dashboard/issues/200#issuecomment-173520066• Google developers used blog post to fix issues on Dashboard with enabling bower –allow root. https://github.com/kubernetes/dashboard/pull/322• Recognition from other developers with blog post copied and amended on another blog for a different OS (<u>Hypriot</u>) http://raspberrypilots.com/guide-installing-kubernetes-dashboard-on-hypriotos/• Live Migration <u>Demonstration</u>: Unable to do due to Arch Linux being <u>uncompatible</u>. Attempted on <u>hypriot</u> – half the packages for dashboard were not available, same with Criu once these were built, the kernel was incompatible.• Working 4.1.18 kernel for Arch Linux has Criu dependencies turned on – all bar dirty memory tracking. Criu 2.0 includes an option to ignore this, therefore should be fully compatible.
Recommended actions
<ul style="list-style-type: none">• Complete write-up of project – own deadline of spring break.• Waiting for the release of Criu 2.0 – should be out 1st week of march.
Deliverables for next time
Other comments

Supervisor signature: Posco Tso

Student signature: Heather McWha

8.2 Code and Scripts

8.2.1 Building Docker Images

[The script for section 5.2.2 of the report]

```
#!/bin/bash

# First, set the version variable for each binary ### UPDATE WHEN NEW VERSION IS
AVAILABLE ###
K8S_VERSION='v1.2.2'
ETCD_VERSION='v2.2.5'
FLANNEL_VERSION='v0.5.5'
REGISTRY_VERSION='v2.3.1'
HEAPSTER_VERSION='v1.0.2'
INFLUXDB_VERSION='v0.12.1'
GRAFANA_VERSION='v2.6.0'
GO_VERSION='1.6'

# Set the build paths & output directory
K8S_DIR="$GOPATH/src/k8s.io/kubernetes"
K8S_CONTRIB="$GOPATH/src/k8s.io/contrib"
HEAPSTER_DIR="$GOPATH/src/k8s.io/heapster"
ETCD_DIR="$GOPATH/src/github.com/coreos/etcd"
FLANNEL_DIR="$GOPATH/src/github.com/coreos/flannel"
REGISTRY_DIR="$GOPATH/src/github.com/docker/distribution"
INFLUXDB_DIR="$GOPATH/src/github.com/influxdb/influxdb"
GRAFANA_DIR="$GOPATH/src/github.com/grafana/grafana"
OUTPUT_DIR="/build/bin"

# Make the build directories - Use -p to force where required
mkdir -p $OUTPUT_DIR \
    $K8S_DIR \
    $K8S_CONTRIB \
    $GOPATH/src/github.com/GoogleCloudPlatform \
    $ETCD_DIR \
    $FLANNEL_DIR \
    $REGISTRY_DIR \
    $HEAPSTER_DIR \
    $INFLUXDB_DIR

#Need to create a symlink from $GOPATH/src/k8s.io/kubernetes
and $GOPATH/src/github.com/GoogleCloudPlatform/Kubernetes
ln -s $GOPATH/src/k8s.io/kubernetes
$GOPATH/src/github.com/GoogleCloudPlatform/kubernetes

### Now the fun part, build the images ###

## ETCD ##
# Get the latest source code and build on ARM
```

```

curl -sSL https://github.com/coreos/etcd/archive/$ETCD_VERSION.tar.gz | tar -C
$ETCD_DIR -xz --strip-components=1
cd $ETCD_DIR

./build

# Copy over the binaries
cp bin/* $OUTPUT_DIR
echo "etcd built"

## FLANNEL ##
# Get the latest source code and build on ARM
curl -sSL https://github.com/coreos/flannel/archive/$FLANNEL_VERSION.tar.gz | tar -C
$FLANNEL_DIR -xz --strip-components=1
cd $FLANNEL_DIR

sed -e "s@go build -o \${GOBIN}/flanneld \${REPO_PATH}@go build -o
\${GOBIN}/flanneld -ldflags \"-extldflags '-static'\" \${REPO_PATH}@\" -i build
./build

cp bin/* $OUTPUT_DIR
echo "flannel built"

### KUBERNETES ###
# Just use the official binaries now they are released

### Prior to this, it was manually built from source code similar to the previous builds of
flannel etc. However, whilst working on the project and gaining traction with running
Kubernetes on an ARM based system, google saw the potential and they themselves now
release arm based binaries for hyperkube and kubectl. Due to this, its no longer required to
build from source. ###

curl -sSL https://storage.googleapis.com/kubernetes-
release/release/${K8S_VERSION}/bin/linux/arm/hyperkube > $OUTPUT_DIR/hyperkube
curl -sSL https://storage.googleapis.com/kubernetes-
release/release/${K8S_VERSION}/bin/linux/arm/kubectl > $OUTPUT_DIR/kubectl
chmod +x $OUTPUT_DIR/hyperkube $OUTPUT_DIR/kubectl

echo "kubernetes built"

## PAUSE ##
cd $K8S_DIR/build/pause
./prepare.sh

# Copy over the binaries
cp pause $OUTPUT_DIR
echo "pause built"

## HEAPSTER ##
# Get the latest source code and build on ARM
curl -sSL https://github.com/kubernetes/heapster/archive/$HEAPSTER_VERSION.tar.gz | tar
-C $HEAPSTER_DIR -xz --strip-components=1
cd $HEAPSTER_DIR

```

```

CGO_ENABLED=0 godep go build -a -installsuffix cgo ./...
CGO_ENABLED=0 godep go build -a -installsuffix cgo -o heapster ./metrics

cp heapster $OUTPUT_DIR
echo "heapster built"

## INFLUXDB ##
# Get the latest source code and build on ARM

curl -sSL https://github.com/influxdata/influxdb/archive/$INFLUXDB_VERSION.tar.gz | tar
-C $INFLUXDB_DIR -xz --strip-components=1
cd $INFLUXDB_DIR

go get github.com/sparrc/gdm

gdm restore -v

# Create symlink for client data:
ln -s $GOPATH/src/github.com/influxdata/usage-client $GOPATH/src/github.com/influxdb/

CGO_ENABLED=0 go build -a --installsuffix cgo --ldflags="-s" -o influxd ./cmd/influxd

cp influxd $OUTPUT_DIR
echo "influxdb built"

## Grafana ##
# Get the latest source code and build on ARM
curl -sSL https://github.com/grafana/grafana/archive/$GRAFANA_VERSION.tar.gz | tar -C
$GRAFANA_DIR -xz --strip-components=1
cd $GRAFANA_DIR

go run build.go setup
godep restore
go run build.go build

cp bin/grafana-server $OUTPUT_DIR
echo "grafana built"

```

8.2.2 Etcd Service

[The script for section 5.3 of the report]

This script brings up etcd and configures it to use the Flannel network.

[Unit]

Description=Etcd Master Data Store for Kubernetes Apiserver

After=system-docker.service

[Service]

EnvironmentFile=/etc/kubernetes/k8s.conf

ExecStartPre=-/usr/bin/docker -H unix:///var/run/system-docker.sock kill k8s-etcd

ExecStartPre=-/usr/bin/docker -H unix:///var/run/system-docker.sock rm k8s-etcd

ExecStartPre=-/bin/sh -c "mkdir -p /var/lib/kubernetes/etcd"

```

ExecStart=/usr/bin/docker -H unix:///var/run/system-docker.sock run \
    --net=host \
    --name=k8s-etcd \
    -v /var/lib/kubernetes/etcd:/var/etcd/data \
    millyna/etcd
ExecStartPost=/bin/sh -c "while [ $(curl -fs http://localhost:4001/v2/machines 2>&1
1>/dev/null; echo $? ) != 0 ]; do sleep 1; done; \
    docker -H unix:///var/run/system-docker.sock run \
        --rm \
        --net=host \
        millyna/etcd \
        etcdctl \
        set /coreos.com/network/config '{ \
            \"Network\": \"${FLANNEL_SUBNET}\", \
            \"Backend\": { \
                \"Type\": \"${FLANNEL_BACKEND}\" \
            } \
        }'"
ExecStop=/usr/bin/docker -H unix:///var/run/system-docker.sock stop k8s-etcd

[Install]
WantedBy=multi-user.target

```

8.2.3 Flannel Service

[The script for section 5.3 of the report]

This script creates the Flannel network and attaches to port 4001 for other devices to connect on.

```

[Unit]
Description=Flannel Overlay Network for Kubernetes

```

```

[Service]
EnvironmentFile=/etc/kubernetes/k8s.conf
ExecStartPre=-/usr/bin/docker -H unix:///var/run/system-docker.sock kill k8s-flannel
ExecStartPre=-/usr/bin/docker -H unix:///var/run/system-docker.sock rm k8s-flannel
ExecStartPre=-/usr/bin/sh -c "rm -rf /var/lib/kubernetes/flannel; mkdir -p
/var/lib/kubernetes/flannel"
ExecStart=/usr/bin/docker -H unix:///var/run/system-docker.sock run \
    --name=k8s-flannel \
    --net=host \
    --privileged \
    -v /dev/net:/dev/net \
    -v /var/lib/kubernetes/flannel:/run/flannel \
    millyna/flannel \
    /flanneld \
    --etcd-endpoints=http://${K8S_MASTER_IP}:4001 \
    --ip-masq=true

```


ExecStop=/usr/bin/docker -H unix:///var/run/system-docker.sock stop k8s-flannel

[Install]

WantedBy=multi-user.target

8.2.4 Docker Service

[The script for section 5.3 of the report]

This script tells the system to bring Docker up with no network configuration.

[Unit]

Description=Docker Application Container Engine

After=network.target system-docker.socket

Requires=system-docker.socket

[Service]

Type=notify

EnvironmentFile=/etc/kubernetes/k8s.conf

ExecStart=/usr/bin/docker daemon \

--host=unix:///var/run/system-docker.sock \

--storage-driver=\${DOCKER_STORAGE_DRIVER} \

--exec-opt native.cgroupdriver=cgroupfs \

--exec-root=/var/run/system-docker \

--pidfile=/var/run/system-docker.pid \

--iptables=false \

--ip-masq=false \

--bridge=none \

--graph=/var/lib/system-docker

MountFlags=slave

LimitNOFILE=1048576

LimitNPROC=1048576

LimitCORE=infinity

[Install]

WantedBy=multi-user.target

8.2.5 Master Service

[This is the script for section 5.3.1 of the report]

[Unit]

Description=The Master Components for Kubernetes

After=docker.service

[Service]

EnvironmentFile=/etc/kubernetes/k8s.conf

ExecStartPre=-/usr/bin/docker kill k8s-master

ExecStartPre=-/usr/bin/docker rm k8s-master

```

ExecStartPre=/bin/sh -c "mkdir -p /etc/kubernetes/static-pods/master; mount -B
/var/lib/kubelet /var/lib/kubelet; mount --make-shared /var/lib/kubelet"
ExecStart=/bin/sh -c "exec docker run \
    --name=k8s-master \
    --net=host \
    --pid=host \
    -v /etc/kubernetes/static-pods/master:/etc/kubernetes/manifests-multi \
    -v /sys:/sys:ro \
    -v /dev:/dev \
    -v /var/lib/docker:/var/lib/docker:rw \
    -v /var/lib/kubelet:/var/lib/kubelet:shared \
    -v /var/run:/var/run:rw \
    --privileged \
    millyna/hyperkube \
    /hyperkube kubelet \
    --allow-privileged \
    --pod_infra_container_image=millyna/pause \
    --api-servers=http://localhost:8080 \
    --cluster-dns=${DNS_IP} \
    --cluster-domain=${DNS_DOMAIN} \
    --v=2 \
    --address=0.0.0.0 \
    --enable-server \
    --hostname-override=$(ip -o -4 addr list eth0 | awk '{print $4}' | cut -d/ -f1) \
    --config=/etc/kubernetes/manifests-multi"

ExecStop=/usr/bin/docker stop k8s-master
Restart=on-failure
RestartSec=5

```

```

[Install]
WantedBy=multi-user.target

```

8.2.6 Worker Service

[This is the script for section 5.3.1 of the report]

```

[Unit]
Description=The Worker Components for Kubernetes
After=docker.service

```

```

[Service]
EnvironmentFile=/etc/kubernetes/k8s.conf
ExecStartPre=/usr/bin/docker kill k8s-worker k8s-worker-proxy
ExecStartPre=/usr/bin/docker rm k8s-worker k8s-worker-proxy
ExecStartPre=/bin/sh -c "mkdir -p /etc/kubernetes/static-pods/worker; mount -B
/var/lib/kubelet /var/lib/kubelet; mount --make-shared /var/lib/kubelet"
ExecStart=/bin/sh -c "exec docker run \
    --name=k8s-worker \

```

```

--net=host \
--pid=host \
-v /etc/kubernetes/static-pods/worker:/etc/kubernetes/manifests \
-v /sys:/sys:ro \
-v /dev:/dev \
-v /var/lib/docker:/var/lib/docker:rw \
-v /var/lib/kubelet:/var/lib/kubelet:shared \
-v /var/run:/var/run:rw \
--privileged \
millyna/hyperkube \
/hyperkube kubelet \
  --allow-privileged \
  --pod_infra_container_image=millyna/pause \
  --api-servers=http://${K8S_MASTER_IP}:8080 \
  --cluster-dns=${DNS_IP} \
  --cluster-domain=${DNS_DOMAIN} \
  --v=2 \
  --address=0.0.0.0 \
  --enable-server \
  --hostname-override=$(ip -o -4 addr list eth0 | awk '{print $4}' | cut -d/ -f1) \
  --config=/etc/kubernetes/manifests"

```

```

ExecStartPost=/usr/bin/docker run -d \
  --name=k8s-worker-proxy \
  --net=host \
  --privileged \
  millyna/hyperkube /hyperkube proxy \
  --master=http://${K8S_MASTER_IP}:8080 \
  --proxy-mode=iptables \
  --resource-container="" \
  --v=2
ExecStop=/usr/bin/docker stop k8s-worker k8s-worker-proxy
Restart=on-failure
RestartSec=5

```

```

[Install]
WantedBy=multi-user.target

```

8.2.7 Heapster

[This is the script for section 5.3.1 of the report]

```

apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    name: influxGrafana
    name: influxdb-grafana
    namespace: kube-system
spec:

```

```

replicas: 1
selector:
  name: influxGrafana
template:
  metadata:
    labels:
      name: influxGrafana
  spec:
    containers:
      - name: influxdb
        image: kubernetesonarm/influxdb
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /data
            name: influxdb-storage
      - name: grafana
        image: millyna/grafana
        imagePullPolicy: IfNotPresent
        env:
          - name: INFLUXDB_SERVICE_URL
            value: http://localhost:8086
          - name: GF_AUTH_BASIC_ENABLED
            value: "false"
          - name: GF_AUTH_ANONYMOUS_ENABLED
            value: "true"
          - name: GF_AUTH_ANONYMOUS_ORG_ROLE
            value: Admin
          - name: GF_SERVER_ROOT_URL
            value: /api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/
        volumeMounts:
          - mountPath: /var
            name: grafana-storage
    volumes:
      - name: influxdb-storage
        emptyDir: {}
      - name: grafana-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  labels: null
  name: monitoring-influxdb
  namespace: kube-system
spec:
  ports:
    - name: http
      port: 8083
      targetPort: 8083
    - name: api
      port: 8086
      targetPort: 8086
  selector:
    name: influxGrafana
---

```

```

apiVersion: v1
kind: Service
metadata:
  labels:
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: "monitoring-grafana"
  name: monitoring-grafana
  namespace: kube-system
spec:
  ports:
    - port: 80
      targetPort: 3000
  selector:
    name: influxGrafana
---
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    k8s-app: heapster
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  selector:
    k8s-app: heapster
  template:
    metadata:
      labels:
        k8s-app: heapster
    spec:
      containers:
        - name: heapster
          image: millyna/heapster
          imagePullPolicy: IfNotPresent
          command:
            - /heapster
            - --source=kubernetes
            - --sink=influxdb:http://monitoring-influxdb:8086
            - --metric_resolution=60s
          volumeMounts:
            - name: ssl-certs
              mountPath: /etc/ssl/certs
              readOnly: true
          volumes:
            - name: ssl-certs
              hostPath:
                path: /etc/ssl/certs
---
apiVersion: v1
kind: Service
metadata:
  labels:
    kubernetes.io/cluster-service: 'true'

```

```

    kubernetes.io/name: "Heapster"
  name: heapster
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    k8s-app: heapster

```

8.2.8 Run Kube

[This is the script for section 5.3.2 of the report]

```

#!/bin/bash

# Catch errors
trap 'exit' ERR

if [[ $K8S_DEBUG == 1 ]]; then
set -x
fi

KUBERNETES_DIR=/etc/kubernetes
ADDONS_DIR=$KUBERNETES_DIR/addons
BINARIES_DIR=$KUBERNETES_DIR/binaries
KUBERNETES_CONFIG=$KUBERNETES_DIR/k8s.conf
PROJECT_SOURCE=$KUBERNETES_DIR/source
KUBECTL=$BINARIES_DIR/kubectl
K8S_PREFIX="millyna"

DOCKER_DROPIN_DIR="/usr/lib/systemd/system/docker.service.d"

# The images that are required
WORKER_IMAGES=(
    "$K8S_PREFIX/flannel"
    "$K8S_PREFIX/hyperkube"
    "$K8S_PREFIX/pause"
)

MASTER_IMAGES=(
    ${WORKER_IMAGES[@]}
    "$K8S_PREFIX/etcd"
)

ALL_IMAGES=(
    ${MASTER_IMAGES[@]}
    "$K8S_PREFIX/heapster"
    "$K8S_PREFIX/influxdb"
    "$K8S_PREFIX/grafana"

```

```

)

TIMEOUT_FOR_SERVICES=20

# Check to see if a kube config file exists, else create this node as master
if [[ ! -f $KUBERNETES_CONFIG ]]; then
    cat > $KUBERNETES_CONFIG <<EOF
K8S_MASTER_IP=127.0.0.1
FLANNEL_SUBNET=10.1.0.0/16
FLANNEL_BACKEND=host-gw
DNS_DOMAIN=cluster.local
DNS_IP=10.0.0.10
DOCKER_STORAGE_DRIVER=overlay
EOF
fi

# Source our config
source $KUBERNETES_CONFIG

# Root is required to run this script
if [ "$EUID" -ne 0 ]; then
    echo "You need to be logged in as root"
    usage
    exit 1
fi

# Remove any existing symlinks which may cause issues
dropins-clean(){
    mkdir -p $DOCKER_DROPIN_DIR
    rm -f $DOCKER_DROPIN_DIR/*.conf
}

dropins-enable(){
    systemctl daemon-reload
    systemctl restart docker
}

# Make a symlink for the overlay config file required by docker
dropins-enable-overlay(){
    dropins-clean
    ln -s $KUBERNETES_DIR/dropins/docker-overlay.conf $DOCKER_DROPIN_DIR/
    dropins-enable
}

# Make a symlink for our flannel network file
dropins-enable-flannel(){
    dropins-clean
    ln -s $KUBERNETES_DIR/dropins/docker-flannel.conf $DOCKER_DROPIN_DIR/

```

```

    dropins-enable
}

require-images(){
    local FAIL=0

    # Check to see if our required images are present
    for IMAGE in "$@"; do
        if [[ -z $(docker images | grep "$(echo $IMAGE | grep -o "[^:]*" | head -1)") ]]; then

            # If it's not exist, try to pull
            docker pull $IMAGE

            if [[ -z $(docker images | grep "$(echo $IMAGE | grep -o "[^:]*" | head -1)") ]]; then

                echo "Pull failed: $IMAGE"
                FAIL=1
            fi
        fi
    done
}

# Load an image to system-docker
load-to-system-docker(){
    # If they don't already exist in docker images, load them
    if [[ -z $(docker -H unix:///var/run/system-docker.sock images | grep "$1") ]]; then
        docker save $1 | docker -H unix:///var/run/system-docker.sock load
    fi
}

get-node-type(){
    local workerstate=$(systemctl is-active k8s-worker)
    local masterstate=$(systemctl is-active k8s-master)
    if [[ $workerstate == "active" ]]; then
        echo "worker";
    elif [[ $masterstate == "active" ]]; then
        echo "master";
    else
        echo "";
    fi
}

# check to see if kubernetes is active?
is-active(){
    if [[ $(get-node-type) != "" ]]; then
        echo 1;
    else

```



```

        echo 0;
    fi
}

checkformaster(){
    if [[ $(curl -m 5 -sSLik http://$1:8080 2>&1 | head -1) == *"OK"* ]]; then
        echo "OK"
    fi
}

# Update a variable in our config file
# Needs variable and new value: updateconfig K8S_MASTER_IP [new value]
updateconfig(){
    updateline $KUBERNETES_CONFIG $1 "$1=$2"
}

updateline(){
    if [[ -z $(cat $1 | grep "$2") ]]; then
        echo "$3" >> $1
    else
        sed -i "$2/c\\$3" $1
    fi
}

wait_for_system_docker(){
    # Wait for system-docker to start by "docker ps"-ing every second
    local SYSTEM_DOCKER_SECONDS=0
    while [[ $(docker -H unix:///var/run/system-docker.sock ps 2>&1 1>/dev/null; echo $?) !=
0 ]]; do
        ((SYSTEM_DOCKER_SECONDS++))
        if [[ ${SYSTEM_DOCKER_SECONDS} == ${TIMEOUT_FOR_SERVICES} ]]; then
            echo "system-docker failed to start. Exiting..." 2>&1
            exit
        fi
        sleep 1
    done
}

wait_for_etcd(){
    # Wait for the etcd to answer
    local ETCD_SECONDS=0
    while [[ $(curl -fs http://localhost:4001/v2/machines 2>&1 1>/dev/null; echo $?) != 0 ]]; do
        ((ETCD_SECONDS++))
        if [[ ${ETCD_SECONDS} == ${TIMEOUT_FOR_SERVICES} ]]; then
            echo "Error: etcd failed to start." 2>&1
            exit
        fi
        sleep 1
    done
}

```

```

done
}

wait_for_flannel(){
    # Wait for the flannel subnet.env file to be created
    local FLANNEL_SECONDS=0
    while [[ ! -f /var/lib/kubernetes/flannel/subnet.env ]]; do
        ((FLANNEL_SECONDS++))
        if [[ ${FLANNEL_SECONDS} == ${TIMEOUT_FOR_SERVICES} ]]; then
            echo "Error: flannel failed to start." 2>&1
            exit
        fi
        sleep 1
    done
}

wait_for_docker(){
    # Wait for docker to start
    local DOCKER_SECONDS=0
    while [[ $(docker ps 2>&1 1>/dev/null; echo $? ) != 0 ]]; do
        ((DOCKER_SECONDS++))
        if [[ ${DOCKER_SECONDS} == ${TIMEOUT_FOR_SERVICES} ]]; then
            echo "docker failed to start. Exiting..." 2>&1
            exit
        fi
        sleep 1
    done
}

start-master(){

    # Disable any running Kubernetes services to avoid conflict
    disable >/dev/null
    sleep 1

    # Check to see if images are present, else pull images from Docker Hub
    require-images ${MASTER_IMAGES[@]}

    # Update the config to show this Pi is the master node
    updateconfig K8S_MASTER_IP 127.0.0.1

    # Load required images to docker
    load-to-system-docker $K8S_PREFIX/etcd
    load-to-system-docker $K8S_PREFIX/flannel

    # Restart docker and wait for it to be available
    systemctl restart system-docker
    wait_for_system_docker

```

```

# Enable etcd service
systemctl enable etcd
systemctl start etcd
wait_for_etcd

# Enable flannel service
systemctl enable flannel
systemctl start flannel
wait_for_flannel

# Point docker to use flannel
dropins-enable-flannel

# Wait for docker to come up
wait_for_docker

# Enable the master service
systemctl enable k8s-master
systemctl start k8s-master

echo "The Kubernetes master is now running"
}

start-worker(){

# Disable any running Kubernetes services to avoid conflict
disable >/dev/null
sleep 1

# Get and update the master IP in our config file
IP=${1:-$K8S_MASTER_IP}
updateconfig K8S_MASTER_IP $IP

# Check if the master is running
if [[ $(checkformaster $IP) != "OK" ]]; then
    echo "Error: The master not running on $IP."
    exit
fi

# Get our required images from docker hub if necessary
require-images ${WORKER_IMAGES[@]}

# Load the flannel to docker
load-to-system-docker $K8S_PREFIX/flannel

# Enable docker
systemctl restart system-docker

```

```

wait_for_system_docker

# Enable and start flannel
systemctl enable flannel
systemctl start flannel

# Wait for flannel
wait_for_flannel

# Point docker to use flannel
dropins-enable-flannel

# Wait for docker to come up
wait_for_docker

# Enable our worker service
systemctl enable k8s-worker
systemctl start k8s-worker

echo "The Kubernetes worker is now running"
}

# Bring down this Pi node safely by stopping all services
disable(){
    systemctl daemon-reload

    systemctl stop flannel etcd k8s-master k8s-worker
    systemctl disable flannel etcd k8s-master k8s-worker

    dropins-enable-overlay
}

# Commands available
case $1 in
    'start-master')
        start-master;;

    'start-worker')
        start-worker $2;;

    'stop-node')
        disable;;

    esac

```

8.2.9 Upgrade Go

[This is the script for section 5.4.1 of the report]

```
#!/bin/bash
cd $HOME
curl http://dave.cheney.net/paste/go-linux-arm-bootstrap-c788a8e.tbz | tar xj
curl https://storage.googleapis.com/golang/go1.6.src.tar.gz | tar xz
ulimit -s 1024 # set the thread stack limit to 1mb
ulimit -s      # check that it worked
cd $HOME/go/src
env GO_TEST_TIMEOUT_SCALE=10 GOROOT_BOOTSTRAP=$HOME/go-linux-arm-
bootstrap ./all.bash
```

8.2.10 Dashboard

[This is the .yaml code for section 4.4.3 of the report]

```
kind: List
apiVersion: v1
items:
- kind: ReplicationController
  apiVersion: v1
  metadata:
    labels:
      app: kubernetes-dashboard
      version: v1.0.1
    name: kubernetes-dashboard
    namespace: kube-system
  spec:
    replicas: 1
    selector:
      app: kubernetes-dashboard
    template:
      metadata:
        labels:
          app: kubernetes-dashboard
      spec:
        containers:
        - name: kubernetes-dashboard
          image: gcr.io/google_containers/kubernetes-dashboard-arm:v1.0.1
          imagePullPolicy: Always
          ports:
          - containerPort: 9090
            protocol: TCP
          args:
          livenessProbe:
            httpGet:
              path: /
              port: 9090
            initialDelaySeconds: 30
            timeoutSeconds: 30
- kind: Service
  apiVersion: v1
```

```
metadata:
  labels:
    app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 9090
  selector:
    app: kubernetes-dashboard
```

8.2.11 getCriuDeps

```
rm -rf criu
rm -rf p.haul
```

```
pacman -Sy --noconfirm wget unzip autoconf automake libbsd git make gcc python python2
pkg-config asciidoc xmlto protobuf protobuf-c python2-setuptools python2-pip
```

echo Thanks! Now go back to standard user and run getCriu

8.2.12 getCriu

```
git clone https://github.com/xemul/criu
git clone https://github.com/xemul/p.haul
```

```
pip2.7 install ipaddr requests flask protobuf google psutil
```

```
cd criu
make install -j5
```

```
cp -R pycriu /usr/lib/python2.7/site-packages/
```

```
cd ../p.haul
python2 setup.py install
```

echo Done! Now edit p-haul/service/wrap to python2