

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

實作一個基於 Docker 具監控及自動部署的容器管理
平台

The Implementation of a Container Management Platform
with Monitoring and Self-Deployment on Docker

研究生：江峻毅

中華民國一零六年七月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 江 峻 毅 所提之論文

實作一個基於 Docker 具監控及自動部署的容器
管理平台

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人

許慶賢 簽章

委 員

呂榮輝

賴冠川

顏信村

指 導 教 授

楊朝棟 簽章

中華民國 106 年 6 月 27 日

摘要

近年來，虛擬化成為新一代資料中心最為關鍵的技術之一。不過虛擬化技術帶來的麻煩是每個實例都需要運行客戶端作業系統以及其中的大量應用程式，因此會產生沉重的負載，也會影響工作效率及性能表現。為了瞭解容器虛擬化是否能夠解決傳統虛擬化碰到的問題，本論文研究跟評估了一些環境的效能表現(裸機, Docker 容器以及虛擬機器)來瞭解關於每一個環境的特點之間的比較。詳細來說，我們透過工具來測量跟分析系統在每個環境中的性能。這些結果可以幫助我們探討 Docker 會帶來多少影響與差異。另外，我們利用 Docker 將應用服務容器化，並且結合 OpenStack 實做一個管理 Docker 容器的監控平台，在過程中碰到 OpenStack 虛擬化發動時間過長的問題，因此我們利用 Nova-Docker 來解決這個問題。Hadoop 是解決大數據儲存和計算的高效工具。它帶來的高效擴展、高容錯等優勢的同時，也增加了部署及維護的難度。Docker 以一處封裝，隨處運行作為出發點，大幅降低複雜平台的部署和維護難度。利用 Docker 來部署 Hadoop，可以大幅度降低部署花費的時間，並且提高部署效率。然後透過比較 Hadoop 有無結合 Docker 來驗證 Docker 是否能夠解決部署難度及花費時間。

關鍵字: 虛擬化、雲端運算、容器虛擬化技術、Docker、Hadoop.

Abstract

In recent years, virtualization is one of the key to next generation of data center. However, the problem of virtualization technology is each instance needs to run a client operating system and a lot of applications. So, it will have a heavy load and also affect the work efficiency and performance. In order to understand whether container-based virtualization can solve the problems of traditional virtualization. In this work, we study and evaluate the performance of these environments(bare-metal, Docker container, and virtual machines) to understand the differences between the characteristics of each environment with another one. For more detail, we measured and analyzed the system performances in each environment. These results can help us explore how much impact of docker performance. In addition, we could build application containerization via Docker, and we combined with OpenStack to implement a container management platform with monitoring based on Docker. We used Nova-Docker as an integration platform because of OpenStack virtualization is running too long. Besides that, Hadoop is an efficient tool to solve Big Data platform, it provided an efficient expansion for high fault tolerance and other advantages and also increased the difficulty of subordinates and maintenance. We used Docker to deploy Hadoop, it can reduce the spend time on deployment significantly and improve more effectively. Then, compare between Hadoop integrates with Docker and without Docker to verify that Docker is able to solve depoly difficulty and it's time spend.

Keywords: Virtualization, Cloud Computing, Container-based virtualization, Docker, Hadoop.

致謝詞

經過兩年在東海研究所的磨練，還有學習研究所的課程過程中，讓我學習到很多，在資訊工程領域上更加精進，研究過程中碰到許多困難跟挑戰，從這中間不斷克服並且解決問題，也讓我增加許多過去大學沒有過的經驗。

能完成這篇論文必須感謝很多人，首先，最感謝我的指導教授楊朝棟教授，從大學專題就一路跟著楊老師直到研究所的論文研究，過程中老師不斷傳授給我及其他同學各種最新科技，讓我接觸各項新的事務，擴展我的視野及國際觀，除了研究方面的教導，楊老師更教會我很多平常做人處事的態度以及應對，在每週會報不斷的提點我、督促我，讓我有動力能夠繼續完成這篇論文，謝謝老師在大學及研究所生涯的指導，雖然常在各方面上碰到瓶頸，但都是因為有老師的鼓勵與幫忙，才能讓我在研究所生活順利進行下去。之後，也會努力遵循老師給我的教誨。

特別感謝口試委員許慶賢教授、呂芳懌教授、賴冠州教授以及薛念林教授百忙之中撥空前來參加我的論文口試，在論文口試時提出很多論文的盲點和非常多寶貴的意見，讓我從各教授意見得到更多啟發，將論文變得更加完整，學生衷心感謝。也還要感謝我的好朋友人豪、培倫、宗岳、學弟妹以及最重要的同學們，這兩年的生活一路走來，在我有問題時，毫不吝嗇的幫助我、給我意見，最後解決問題。

最後要感謝我的家人，在我的求學路上，讓我心無旁騖，可以專心在學業上，有了你們的叮嚀跟關心，讓我可以更堅定的走在這條人生道路上，並且讓我的研究得以成功，由衷感謝一路陪伴的所有人。

東海大學資訊工程學系 高效能實驗室 江峻毅 106 年 07 月

Table of Contents

摘要	i
Abstract	ii
致謝詞	iii
Table of Contents	iv
List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Motivation	2
1.2 Thesis Goal and Contributions	3
1.3 Thesis Organization	3
2 Background Review and Related Works	4
2.1 Virtualization and Hypervisor	4
2.1.1 Docker	5
2.1.2 Hypervisor	7
2.2 Hadoop Ecosystem	9
2.2.1 Hadoop	9
2.2.2 Apache Spark	11
2.3 OpenStack	11
2.3.1 OpenStack Componet	12
2.3.2 OpenStack Conceptual Architecture	14
2.4 Power Distribution Units(PDU)	16
2.5 Related Works	17
3 System Design and Implementation	20
3.1 System Design Architecture	20
3.2 System Implementation	21
3.2.1 Status Monitoring	23
3.2.2 Dockerize Hadoop	24
3.2.3 Dockerize Spark	26

3.2.4	Assign Tasks	26
3.2.5	User Services	28
4	Experimental Results	30
4.1	Experimental Environments	30
4.2	Boot-Time on OpenStack	31
4.3	Docker and Virtual Machine Performance Comparison	32
4.3.1	CPU Utilization of Virtual Machine	32
4.3.2	File I/O Performance Comparison between Docker and KVM	33
4.4	Performance of Hadoop and Spark Comparison between Docker and Virtual Machine	36
4.4.1	Execution Time of Hadoop and Spark on Docker and Vir- tual Machine	37
4.4.2	Deploying Hadoop in Different Environments	38
4.5	Container Management Platform	38
5	Conclusions and Future Work	41
5.1	Concluding Remarks	41
5.2	Future Work	42
	References	43
	Appendix	47
A	Hadoop Installation	47
B	Spark Installation	51
C	Docker Installation	53
D	Nova-Docker Installation	55
E	PDU Information program	57

List of Figures

2.1	Different between traditional architecture and virtual architecture . . .	5
2.2	Different between virtual architecture and Docker architecture . . .	6
2.3	Hosted hypervisor	8
2.4	Bare-metal hypervisor	9
2.5	The architecture of Hadoop	10
2.6	The conceptual architecture of OpenStack	15
2.7	Raritan's PDU	17
3.1	The overview architecture of system	21
3.2	Web service	22
3.3	OpenStack on Docker	23
3.4	docker api1	23
3.5	docker api2	24
3.6	The process of Hadoop on Docker	25
3.7	Packaging Hadoop into image	25
3.8	Spark Dockerfile	26
3.9	The process of launch	27
3.10	Assign tasks	27
3.11	Send information back to web	28
4.1	Boot-time on OpenStack	31
4.2	Average server boot time between KVM	32
4.3	Average server reboot time between KVM	33
4.4	Average server delete time between KVM	33
4.5	CPU utilization of virtual machine	34
4.6	Read speed comparison between Docker and KVM	34
4.7	Write speed comparison between Docker and KVM	35
4.8	Read and Write speed comparison between Docker and KVM	35
4.9	Power usage during the experiment	36
4.10	Execution time of Hadoop in different workloads	37
4.11	Execution time of Spark in different workloads	37
4.12	Hadoop start-up time	38
4.13	Container monitoring on the web site	39
4.14	Do actions on web site directly	39
4.15	Remove images on web site directly	40
4.16	Remove volumes on web site directly	40

4.17 Users information 40



List of Tables

2.1	KVM size	18
2.2	Docker size	18
3.1	Software & language Specification	28
4.1	Hardware specification	30
4.2	Software specification	31
4.3	The data of power usage	36

Chapter 1

Introduction

Cloud computing has a huge change for industry development after Internet. Not only the IT industry which provide cloud computing technology, but also the general using way of the government, enterprise and personal are getting some change with the born of cloud computing. In the IT industry, cloud computing is undoubtedly caused a comprehensive impact. From the most basic computer components - processors, servers, storage devices, network equipment, information security equipment, software, data centers, information services, even though smart phone, tablet computer and other emerging mobile devices are unable to break off relations from cloud computing. In recent years, cloud computing has become one of the hottest topics. Cloud computing mainly combines virtualization, service management automation and standardized technology. It provides flexible computing ability and data analysis method with high performance. Companies can run many kinds of service on the cloud platform without constructing data center. This innovative computing and business model has attracted widespread attention in industry and academia.

Cloud computing has become the most popular topic today; OpenStack and Docker are the most popular undoubtedly. Easy management of the virtual machine of OpenStack combined with Docker that is light and fast becoming we want to try, if combined with OpenStack and Docker can be spread, for cloud computing certainly is a great leap forward. Under the popularity of cloud computing, we

began to research for energy efficiency, how to achieve the highest resource utilization and lowest energy is the issue they face of enterprise, schools, and other places which need a lot of computing.

Docker is an open platform for developing, shipping, and running various applications in a faster way. Container technology is available through the operating system. A container packages the application service or function with all of the libraries, configuration files, dependencies and other necessary parts to operate. Each container shares the services of one underlying operating system. Docker has emerged as a de facto standard platform that allows users to quickly compose, create, deploy, scale and oversee containers across Docker hosts. Docker allows a high degree of portability so that users can register and share containers over various hosts in private and public environments. Docker benefits include efficient application development, lower resource use and faster deployment compared to VMs.

1.1 Motivation

In recent years, many companies using the traditional way of one project and one test environment, for different projects to establish a separate environment. The advantage of this way is that the environment is built for projects and is used in a single project environment where the cost is low and suitable for situations where the number of test items are small, but as the business grows, the business increases and the number of test items increase, the disadvantages of this kind of way are gradually being apparent. So we think if we can improve the problem with changing hypervisor will be effective, so we expect Docker that very popular recently. If Docker as hypervisor, using the characteristics of Docker that light, fast, and high utilization of resources, to explore the energy consumption, resource utilization, and performance whether be improved during computing, and the speed of deploy cluster whether be faster than KVM hypervisor.

1.2 Thesis Goal and Contributions

This work will implement a container management platform on Docker. Users will see some information about container, such as how many containers are running and when the user created it. There are also many features, you can pause, stop and delete containers. We also compared the performance of some environments to understand that about comparison between the characteristics of each environment with one another. First, we will compare the time of turning on virtual machines of Docker hypervisor and KVM hypervisor, it will turn on five virtual machines at the same time to understand that the time of turning on multiple virtual machines. Next, establish a Hadoop cluster on these five virtual machines of Docker hypervisor and KVM hypervisor, allocating the same amount of resources, measuring the power consumption and resources utilization. According to the data we obtained, we want to know if Docker is faster than general virtual environment. Finally, we provide a platform to manage containers.

1.3 Thesis Organization

In Chapter 2, we will describe some background information, including Virtualization, Hypervisor, Docker, Mesos, Hadoop, OpenStack and related work. In Chapter 3, we will introduce our experimental environment and the overall architecture of our design. Chapter 4 shows the experimental results and analysis. Chapter 5 provides conclusions and future work of this work.

Chapter 2

Background Review and Related Works

2.1 Virtualization and Hypervisor

In computing, virtualization [1–4] refers to the act of creating a virtual (rather than actual) version of something, including (but not limited to) a virtual computer hardware platform, operating system, storage device, or computer network resources. With virtualization, the computer's physical resources, such as servers, network, memory, and storage, are abstractly presented after conversion, so that users can apply those resources in a better way than the original configuration. Simply put, virtualization is a technology that allows the user to transform hardware into software, and it allows the user to run multiple operating systems simultaneously on a single computer. Virtual architecture is different from traditional architecture, as shown in Figure 2.1. Traditional architecture can run single operating system on a single computer, but the virtual architecture can run multiple operating systems on a single computer.

There are many benefits of virtualization, such as:

- Encapsulation - VMs can be described in a file.

- Enables running multiple operating systems.
- Resource management.
- High availability and disaster recovery.
- Create “Base Environment” .
- Safe testing of new software.
- Easy Management.

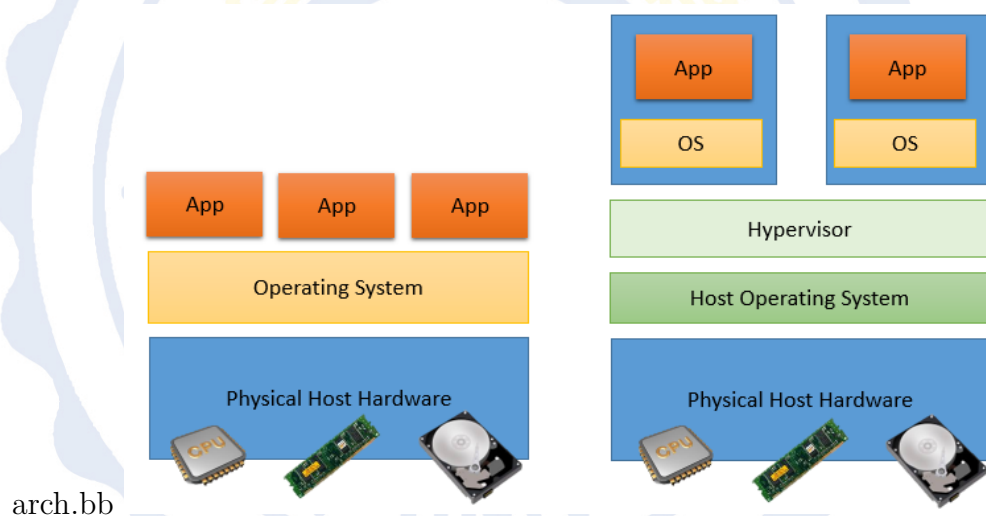
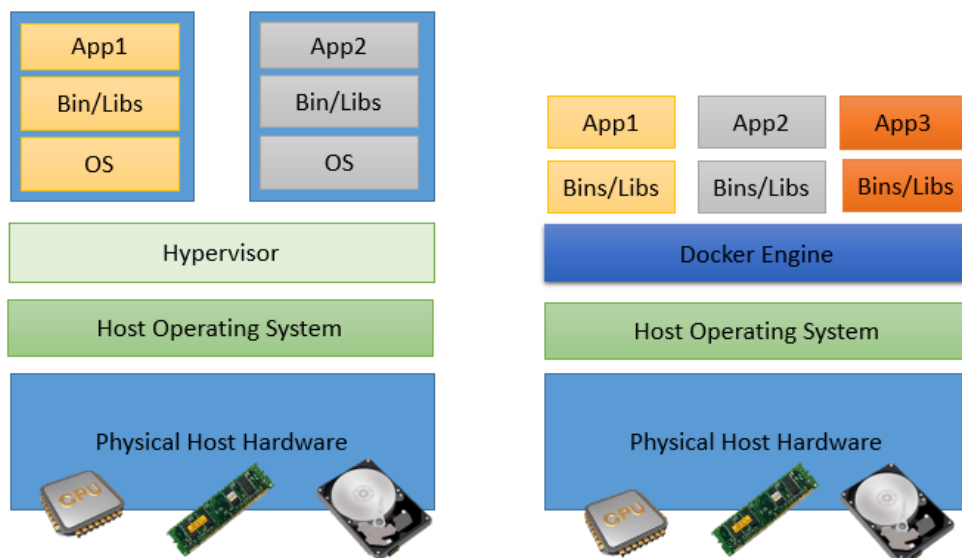


FIGURE 2.1: Different between traditional architecture and virtual architecture

2.1.1 Docker

Docker [5–8] is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable filesystem such as aufs and others to allow independent “containersto” run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. The Linux kernel’s support for namespaces mostly isolates an application’s view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel’s cgroups

provide resource limiting, including the CPU, memory, block I/O and network. Since version 0.9, Docker includes the libcontainer library as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC [9–11] (Linux Containers) and systemd-nspawn. By using containers, resources can be isolated, services restricted, and processes provisioned to have an almost completely private view of the operating system with their own process ID space, file system structure, and network interfaces. Multiple containers share the same kernel, but each container can be constrained to only use a defined amount of resources such as CPU, memory and I/O. Using Docker to create and manage containers may simplify the creation of highly distributed systems by allowing multiple applications, worker tasks and other processes to run autonomously on a single physical machine or across multiple virtual machines. This allows the deployment of nodes to be performed as the resources become available or when more nodes are needed, allowing a platform as a service (PaaS)-style of deployment and scaling for systems like Apache Cassandra, MongoDB or Riak. Docker also simplifies the creation and operation of task or workload queues and other distributed systems. Docker architecture is different from virtual architecture, as shown in Figure 2.2.



arch.bb

FIGURE 2.2: Different between virtual architecture and Docker architecture

2.1.2 Hypervisor

A hypervisor [12, 13] or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs VMs. A computer on which a hypervisor is running one or more VMs is defined as a host machine. Each VM is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources. Simply stated, a hypervisor creates a layer of abstraction that isolates an OS and its associated applications from the underlying computing hardware. The isolation effectively mitigates software from its traditional reliance on hardware devices and their drivers. The implications of this behavior are profound. A hypervisor allows OSes and their application workloads to run on a broader array of hardware. Similarly, multiple OSes and workloads, each a unique VM or VM instance, can reside on the same system to simultaneously share computing resources. Each VM can be migrated between computing platforms on demand with little (if any) processing disruption. The result is better use of computing platforms with seamless workload migration and backup capabilities. Hypervisors generally fall into two categories: hosted and bare-metal. Both offer distinct benefits and drawbacks.

A hosted hypervisor, as shown in Figure 2.3, runs within the OS and allows additional OS and application instances to run on top of it. VMware Server and Microsoft Virtual Server, as well as numerous endpoint-based virtualization platforms like VMware Workstation, Microsoft Virtual PC and Parallels Workstation are hosted hypervisors.

There are advantages of Hosted Hypervisor:

- Virtualization installs like application rather than like OS.
- Can run alongside conventional applications.
- Avoid code duplication—OS already has process scheduler, memory management, device support etc.

- More suitable for personal users.

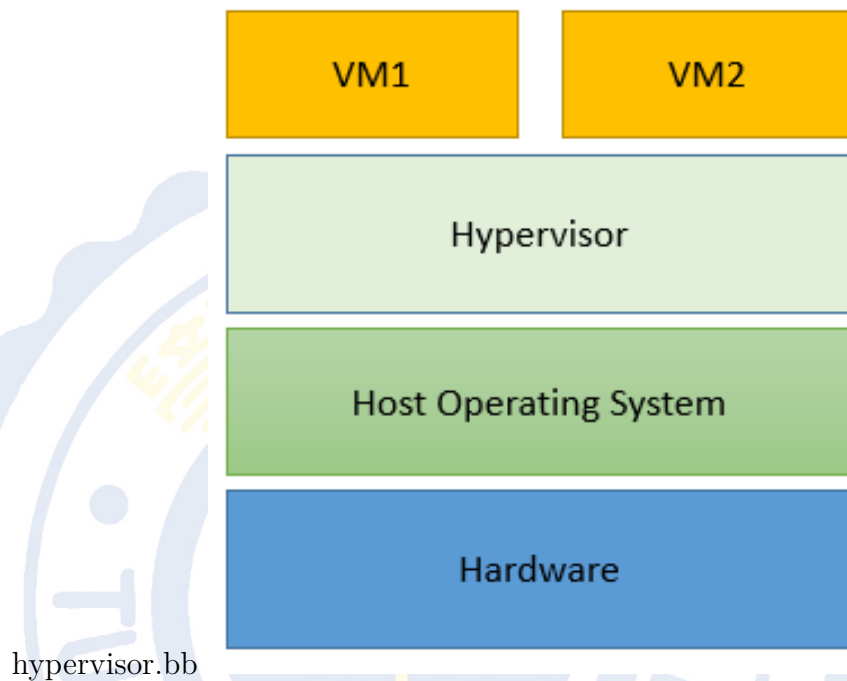
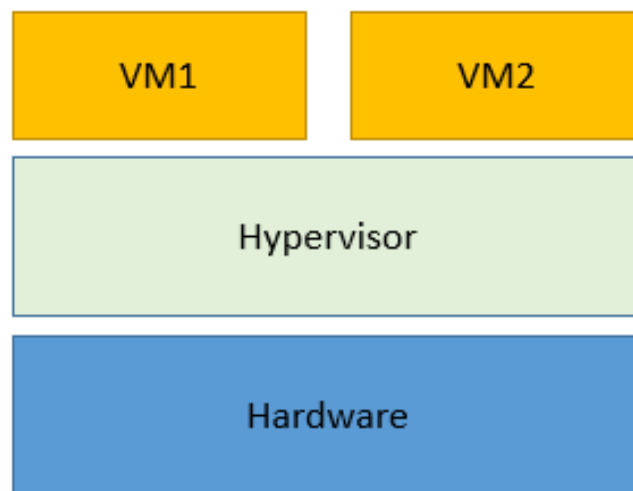


FIGURE 2.3: Hosted hypervisor

Bare-metal hypervisor, as shown in Figure 2.4, is the most commonly deployed type, and it can be installed directly onto the computing hardware. Its OS installs and runs above the hypervisor. Major virtualization products that can be termed as bare-metal hypervisors include Oracle VM, VMware ESXi, Microsoft Hyper-V and Citrix XenServer.

There are advantages of Bare-Metal Hypervisor:

- Better performance with lower overhead.
- Highly efficient direct I/O pass-through architecture for network and disk.
- Complete control over hardware.
- Advanced features like live migration available.
- Suitable for production environments.



hypervisor.bb

FIGURE 2.4: Bare-metal hypervisor

2.2 Hadoop Ecosystem

2.2.1 Hadoop

Apache Hadoop [14–18], the most popular solutions for big data processing now, is Apache Software Foundation open source framework. Hadoop implementation is constructed in accordance with published Google MapReduce and Google File System papers. The Apache Hadoop framework is built on top of the Hadoop Distributed File System (HDFS), and it supports a stable and automatic distributed processing system. Hadoop implements the MapReduce programming framework, which divides files into smaller file fragments of the same size, and allows file fragments executed on any node in the cluster. Hadoop is designed to scale up from a single server to thousands of machines, and provides parallel computing. The architecture shown in Figure 2.5.

The Apache Hadoop project consists of the following:

The project includes these modules:

- Hadoop Common: The Hadoop common contains the libraries and modules of Hadoop.

- HDFS: HDFS is designed to provide high throughput access to very large datasets.
- Hadoop YARN: A framework for cluster resource management and task scheduling.
- Hadoop MapReduce: MapReduce is composed of the map and reduce, of which the input is divided into a plurality of blocks to be executed on each node.

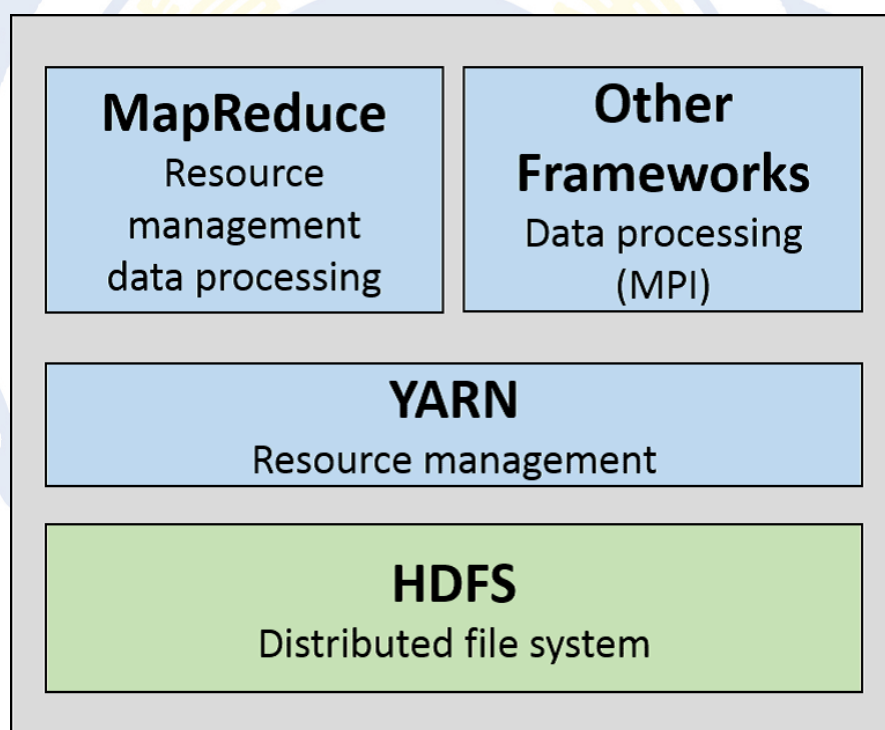


FIGURE 2.5: The architecture of Hadoop

Hadoop ecosystem has very diverse tools to make Hadoop useful in many applications, for example, tools such as Spark and Storm are useful for processing real-time streaming data; in addition, the in-memory technology of Spark makes it a good solution for machine learning; HBase is useful for NoSQL data storage, and Sqoop is useful for data conversion between relational database and Apache Hadoop.

2.2.2 Apache Spark

Apache Spark [19] is an open-source cluster computing framework originally developed in the AMPLab at UC Berkeley. Compared to the two-stage disk-based MapReduce paradigm of Hadoop, Spark's in-memory primitives provide performance up to 100 times faster for certain applications. By permitting user programs to load data into memory of a cluster and repeatedly query it, Spark is well suited for machine learning algorithms. Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety of systems, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, and Amazon S3. Spark also supports a pseudo distributed local mode, usually used only for developing or testing purposes, where distributed storage is not required and the local file system can be used instead; in this scenario, Spark is running on a single machine with one executor per CPU core. In 2014, Spark has more than 465 contributors, making it the most vigorous project in the Apache Software Foundation and Big Data open source projects.

2.3 OpenStack

OpenStack [20–25] is a free and open-source cloud computing software platform. It began in 2010 as a joint project of Rackspace Hosting and NASA. Currently, it is managed by the OpenStack Foundation, a non-profit which oversees both development and community-building around the project. And OpenStack.org released it under the terms of the Apache License. Users primarily deploy it as an IaaS solution. The technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a data center which users manage through a web-based dashboard, command-line tools, or a RESTful API.

2.3.1 OpenStack Component

OpenStack has a modular architecture with various code names for its components.

- **Compute (Nova):** OpenStack Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations. KVM, VMware, and Xen are available choices for hypervisor technology, together with Hyper-V and Linux container technology such as LXC.
- **Object Storage (Swift):** OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used. The Total Cost of Ownership (TCO) can be higher than using enterprise-class storage because many copies require high availability.
- **Block Storage (Cinder):** Cinder is a block storage service for OpenStack. It is designed to allow the use of either a reference implementation (LVM) to present storage resources to end users that can be consumed by the OpenStack Compute Project (Nova). The short description of Cinder is that it virtualizes pools of block storage devices and provides end users with a self-service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device.

- Networking (Neutron): OpenStack Networking (Neutron, formerly Quantum) is a system for managing networks and IP addresses. OpenStack Networking ensures the network is not a bottleneck or limiting factor in a cloud deployment, and gives users self-service ability, even over network configurations.
- Dashboard (Horizon): OpenStack Dashboard (Horizon) provides administrators and users a graphical interface to access, provision, and automate cloud-based resources. The design accommodates third party products and services, such as billing, monitoring, and additional management tools. The dashboard can also be branded for service providers and other commercial vendors who want to make use of it. The dashboard is one of several ways users can interact with OpenStack resources. Developers can automate access or build tools to manage resources using the native OpenStack API or the EC2 compatibility API.
- Identity Service (Keystone): OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. Additionally, the catalog provides a list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can determine which resources they can access by programs.
- Image Service (Glance): OpenStack Image Service (Glance) provides discovery, registration, and delivery services for disk and server images. Stored images can be used as a template. It can also be used to store and catalog an unlimited number of backups. The Image Service can store disk and server images in a variety of back-ends, including OpenStack Object Storage. The Image Service API provides a standard REST interface for

querying information about disk images and lets clients stream the images to new servers.

- **Telemetry (Ceilometer):** OpenStack Telemetry Service (Ceilometer) provides a single point of contact for billing systems, providing all the counters they need to establish customer billing, across all current and future OpenStack components. The delivery of counters is traceable and auditable, the counters must be easily extensible to support new projects, and agents doing data collections should be independent of the overall system.
- **Orchestration (Heat):** Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat also attempts to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack. Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API.
- **Database (Trove):** Trove is Database as a Service for OpenStack. It is designed to run entirely on OpenStack, with the goal of letting users to quickly and easily utilize the features of a relational or non-relational database without the burden of handling complex administrative tasks. Cloud users and database administrators can offer and manage multiple database instances as needed. Initially, the service will focus on providing resource isolation at high performance while automating complex administrative tasks such as deployment, configuration, patching, backups, restores, and monitoring.

2.3.2 OpenStack Conceptual Architecture

Launching a VM or instance involves many interactions among several services. Figure 2.6 provides the conceptual architecture of a typical OpenStack environment. In this work, we use version IceHouse. We just use Nova, Glance, Keystone

and Horizon in our model.

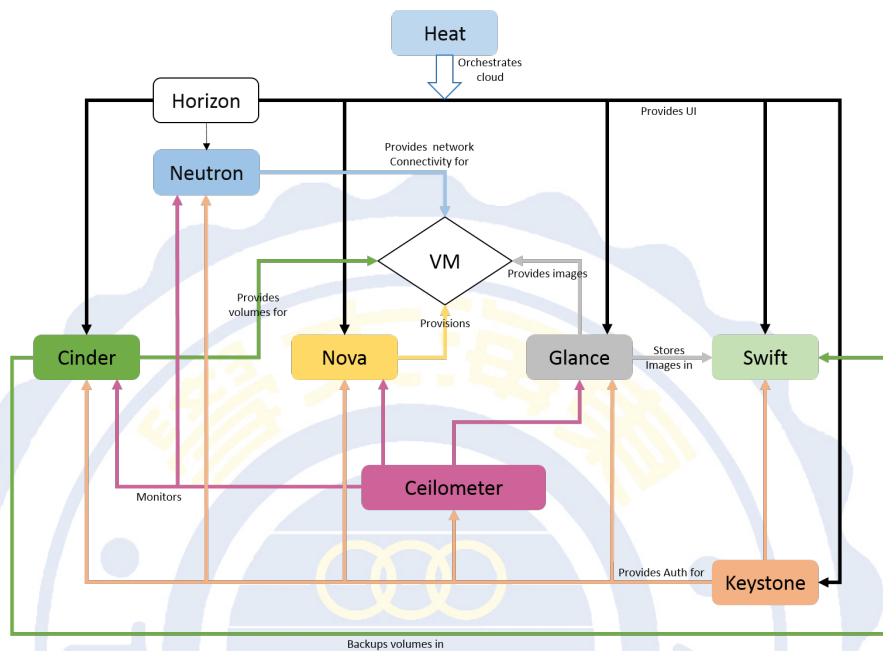


FIGURE 2.6: The conceptual architecture of OpenStack

2.4 Power Distribution Units(PDU)

A Power Distribution Unit (PDU) [26,27], as shown in Figure 2.7, is a device used in datacenters to distribute AC power to multiple servers and other equipment. Power distribution units (PDUs) range from simple 120v power strips to units that break out 120 volts from 240v and three-phase power. Advanced units are managed remotely via the SNMP management protocol or from a Web browser or other management console, causing outlets to be turned on and off at prescribed times and in a proper sequence for shutting down and powering up equipment. The growing complexity of IT environments, from wiring closets and server rooms to data centers of all sizes, has increased the need for reliable power distribution to the rack level. Eliminating power management issues is essential for IT and Facilities managers to maintain system availability of increasing higher density equipment. Power Distribution Units are an essential element in managing power capacity and functionality for critical network, server and data center equipment.

- **Basic PDU:** The most basic PDU is a large power strip without surge protection. It is designed to provide standard electrical outlets for data center equipment and has no monitoring or remote access capabilities. The floor-mounted and rack-mounted PDUs can be more sophisticated, providing data that can be used for power usage effectiveness (PUE) calculations.
- **Floor-mounted PDU:** A floor-mounted PDU, sometimes called a main distribution unit (MDU), provides an important management bridge between a building's primary power and various equipment racks within a data center or network operations center (NOC). Each PDU can handle larger amounts of energy than an ordinary power strip (300 kilovolt-amps and higher depending on the manufacturer and model) and typically provides power to multiple equipment racks.
- **Rack-mountable PDU:** A rack-mountable PDU mounts directly to an equipment rack so it can control and monitor power to specific servers, switches

and other data center devices and assist in balancing power loads. Rack-mountable PDAs are known by several different names, including smart PDUs and intelligent PDUs. Such PDUs include three-phase displays for devices sharing power well as remote management tools that use the Simple Network Management Protocol (SNMP) to provide administrators with the ability to adjust and monitor power demands from offsite locations.



FIGURE 2.7: Raritan's PDU

2.5 Related Works

In the recent years, there are many research about Docker with OpenStack, container, power consumption. We choose some research about them to discussion.

Preeth E N et al. [28] in 2015, evaluated the performance of these Docker containers based on their system performance. That is based on system resource utilization. Different benchmarking tools are used for this. Performance based on file system is evaluated using Bonnie++. Detail results obtained from all these tests are also included in this paper. The results include CPU utilization, memory utilization, CPU count, CPU times, Disk partition, network I/O counter etc.

Kyoung-Taek Seo et al. [29] in 2014, tested the average boot-time of Docker and KVM, as shown in Table 2.1 and Table 2.2, VM of KVM uses Full-Virtualization, and specifies the size when it is generated. They need resources more than 8GB

when they use Ubuntu-Desktop, so it is hard to generate more than 50 virtual machines on 500GB Hard-Disk. Docker containers do not contain OS, only installed software resources, so their size are smaller than VM' s. Because of Ubuntu 14.04 image of Docker only have basic software, it just uses half of the same 500GB HDD and 177MB of resources to generate more than 100 containers.

TABLE 2.1: KVM size

Scale	10GB	20GB	40GB
Number of VMs	45	22	11

TABLE 2.2: Docker size

Number of containers	100+
----------------------	------

P. China Venkanna Varma et al. [30] in 2016, studied the working of Docker networks, various factors of CPU context switch latency and how network IO throughput will be impacted with the number of live Docker containers. A Hadoop cluster environment built and executed benchmarks such as TestDFSIO-write and TestDFSIO-read against varying number of the live containers. They observed that Hadoop throughput is not linear with increasing number of live container nodes sharing the same system CPU.

Javier Conejero et al. [31] in 2016, measured the power consumption of Hadoop. In order to understand the power consumption of the Cloud, they devised some experiments that consider three aspects:

- **Basic Power Consumption:** monitoring the power consumption of the running cluster. It involves analyzing the power demand when turning on and off the Cardiff Cloud testbed system without any workload. The power consumption profile is particularly high when the server is switched on and off and stable once it has booted. There is a peak in power consumption during the server starting up, which stabilizes after the operating system finishes loading all services (at 105W). To stop the server requires power to stop all the services. When the server is stopped, a power consumption of 10W is observed because the standby state of the server.

- **power Consumption Range:** this aspect is focus on measuring the maximum and minimum power consumption of a cluster. In order to measure the maximum power consumption, it has to choose a workload that fully stresses all the physical hardware (CPU, memory, and disk) available on the server. They use the Message-Digest Algorithm (MD5)[29] to fully the physical hardware. The power consumption is proportional with the CPU utilization, and they reach the top with 16 threads. The maximum power consumption is 268W.
- **Virtualization and Power Consumption:** this aspect involves monitoring the power consumption with different virtualized workloads executed on a single node Cloud environment. It focuses on analyzing the behavior of the system under a realistic Hadoop workload, performed across different VCs.

The number of worker nodes and their characteristics are not the same, and keeping constant the number of resources allocated to all the VCs. With more worker nodes, the power consumption is increase when the Cluster is idle state. So the size of cluster also impact on power consumption, even if not performing any Hadoop application.

Chapter 3

System Design and Implementation

The main goal of our thesis is to figure out that how to combine Docker with OpenStack. Then we build a platform contains a variety of functions, Users can see many information about containers on this platform, In this section, we will introduce our system architecture. Finally, we will show our user interface.

3.1 System Design Architecture

In the proposed system architecture shown in Figure 3.1. OpenStack had adopted as the basis in order to achieve storage virtualization and unified management. The architecture consists of a controller node and two computing nodes. Several major OpenStack services are running on the Controller node, such as Identity service, Image service, Networking service, Nova service, and Dashboard. Two compute nodes had connected to the PDU for monitoring and recording their energy consumption.

of system.bb

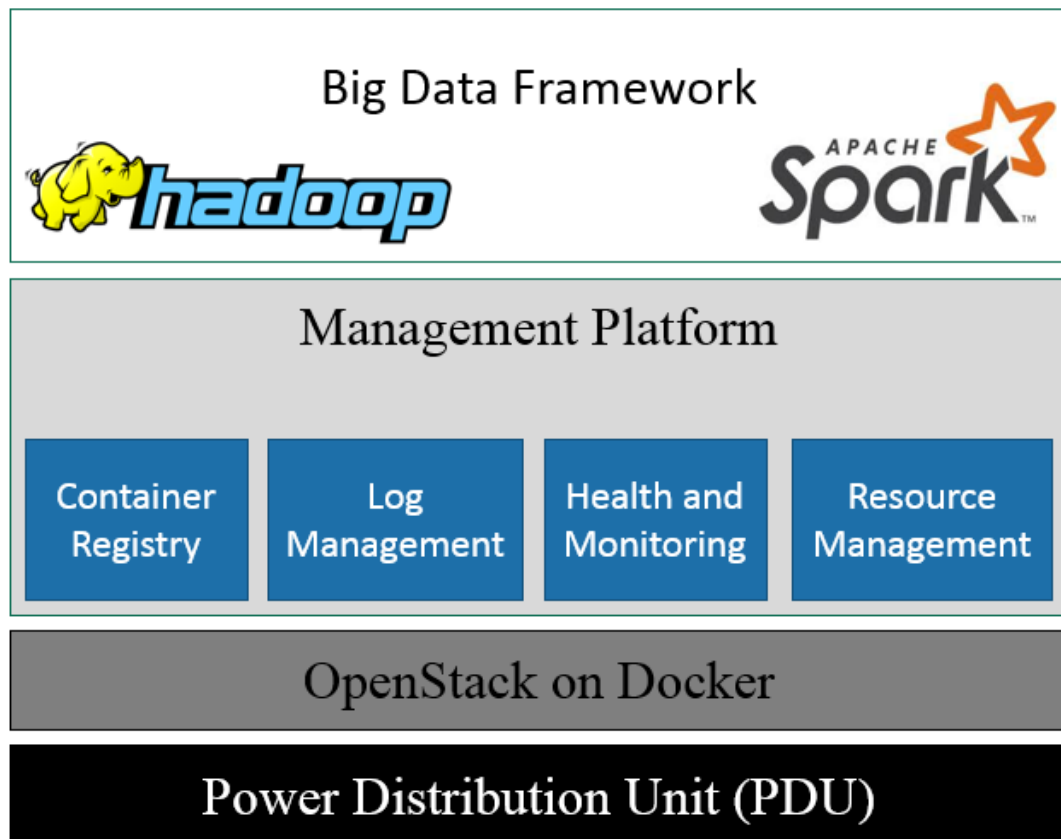


FIGURE 3.1: The overview architecture of system

3.2 System Implementation

In this thesis, we will use the Docker api and PHP to write some automatic program, including the state monitoring program which can monitor the information of containers, energy consumption record program which can monitor the power consumption. The following is a detailed description of the programs.

We implement a website for users to connect with OpenStack api. First, users can run what their need by our web service. The process shown is Figure 3.2.

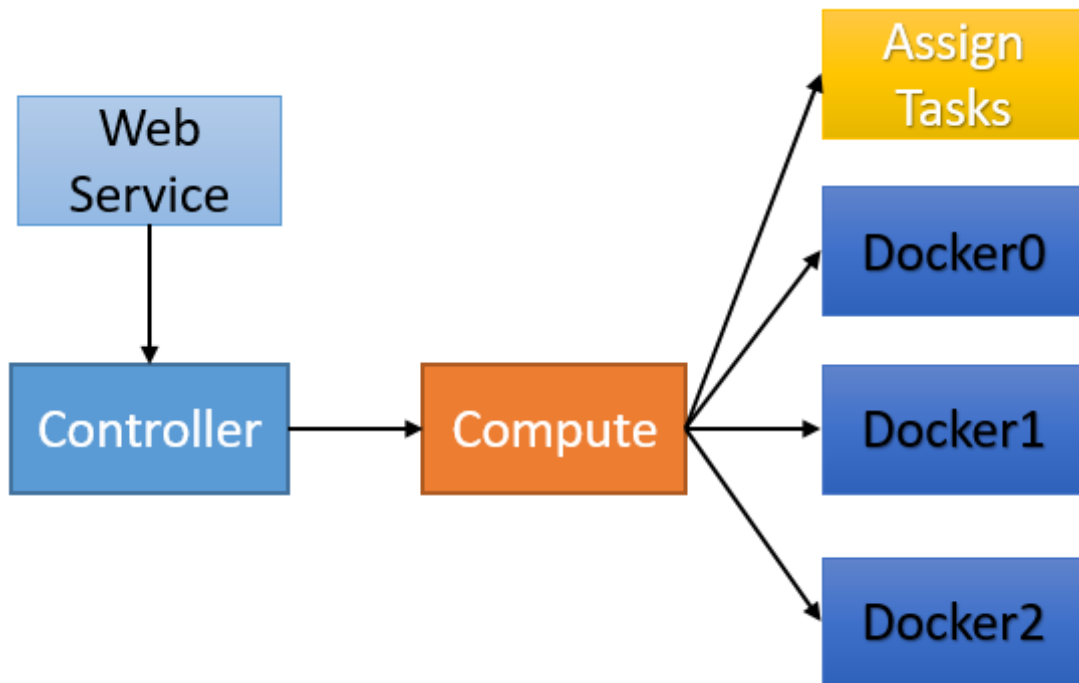


FIGURE 3.2: Web service

- First, users enter the requirements they needs from web service.
- Then, controller node will arranges tasks
- Finally, compute node will assigns tasks, then build up virtual machine.

In order to make the service faster, reduce the time to launch the virtual machine can increase the speed rate. KVM and QEMU on OpenStack because of the complete virtualization led to launch time too long, docker based on Linux container technology can solve the problem effectively. Applications and virtualization are more lightweight and faster.

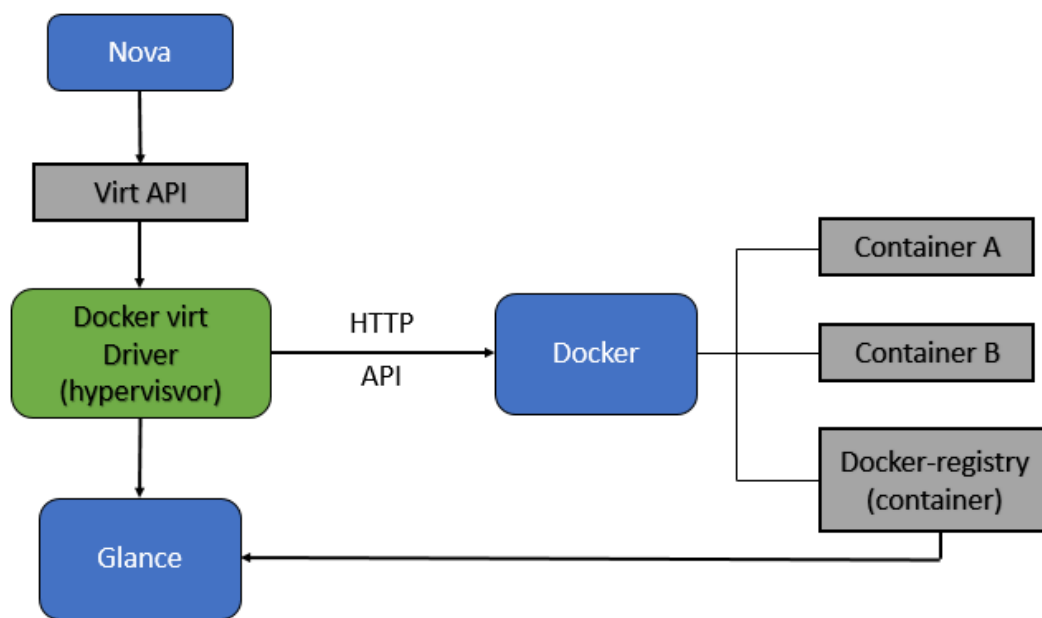


FIGURE 3.3: OpenStack on Docker

3.2.1 Status Monitoring

We monitored the status of each container via web user interface. If we want to present information about containers on the site, we need to get it through Docker official api shown in Figure 3.4 and 3.5, and then sort out the data we need, we information visualization on the user interface finally. By the web user interface we can see which container is running, when the container been created. Through these monitoring data, we can also observe the status of the images on each container. The status monitoring function was developed by the Docker api to obtain status data. The monitoring data includes the utilization of CPU and memory.

```

class ContainersController < ApplicationController
  def index
    @dockers = Docker::Container.all
  end
end
end
  
```

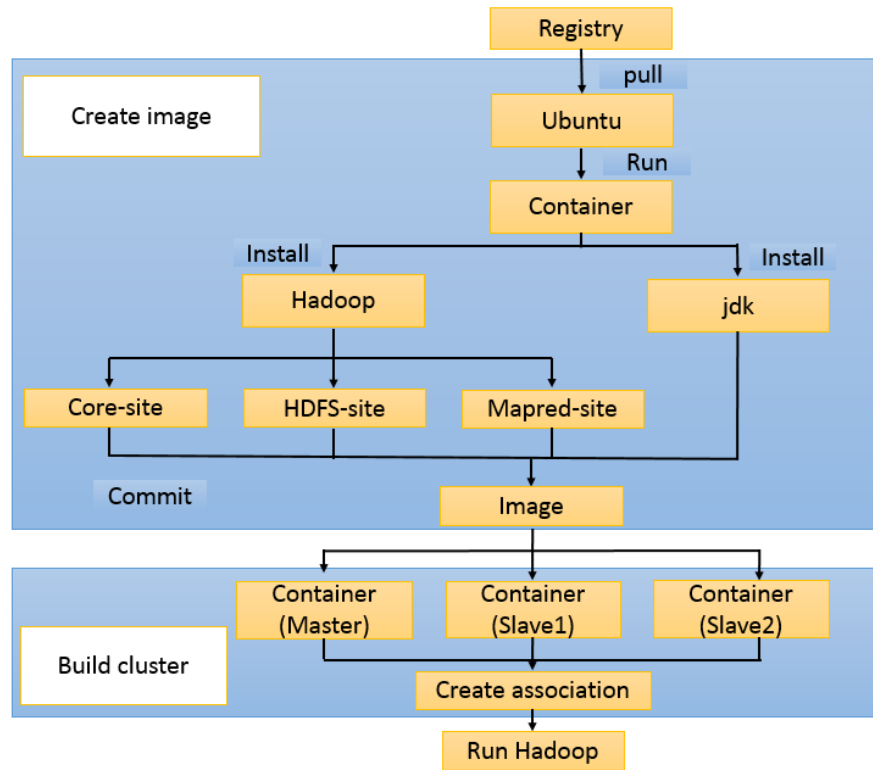
FIGURE 3.4: docker api1

```
<!-- row -->
<div class="row">
  <div class="col-xs-12">
    <table class="table table-hover table-striped table-bordered">
      <tbody>
        <tr>
          <th>VM ID</th>
          <th>Status</th>
          <th>Image</th>
        </tr>
        <%=@dockers.each do |docker|%=>
          <tr>
            <td><%=docker.id[0,6]%=></td>
            <td><%=docker.info['Status']%=></td>
            <td><%=docker.info['Image']%=></td>
          </tr>
        <%=end%=>
      </tbody>
    </table>
  </div>
</div>
</section>
```

FIGURE 3.5: docker api2

3.2.2 Dockerize Hadoop

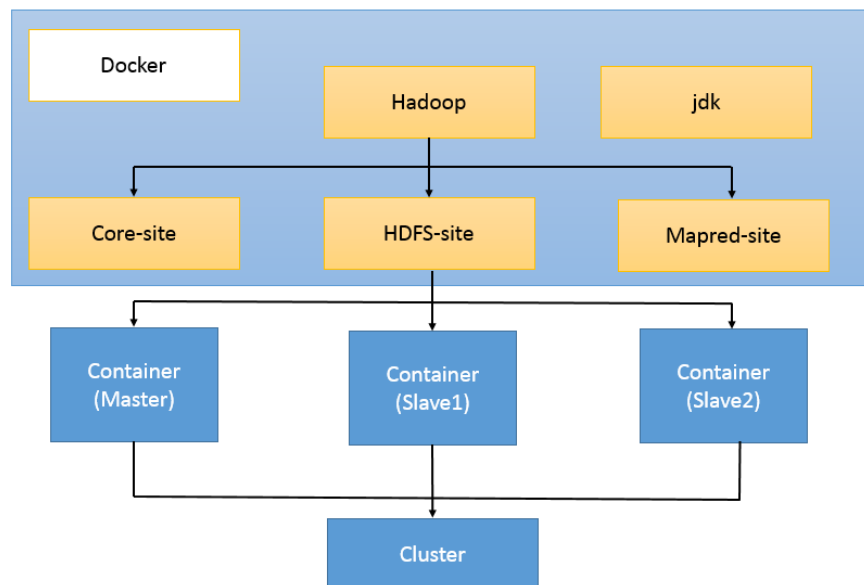
Figure 3.6 shows the process of Hadoop on Docker. At first, client download Ubuntu from the Registry and then create container. Second, run the container, they will install jdk and Hadoop, then package the information in the container into an image. The client creates the container for the master and slave, and then creates an association between master and slave, after all, run Hadoop.



on docker.bb

FIGURE 3.6: The process of Hadoop on Docker

Hadoop on Docker is mainly package Hadoop and jdk into an image, when the client needs to build or expand Hadoop, just pulling the image, and do some simple configuration.



docker.bb

FIGURE 3.7: Packaging Hadoop into image

3.2.3 Dockerize Spark

First deploy Spark cluster as Docker container spanning multiple physical host, then automate Spark service configuration inside containers to facilitate cluster cloning. Finally, container storage is always ephemeral. Persistent storage is external.

- Base.
- Master container runs all Spark services(master, worker, jupyter, zeppelin)
- Worker container runs Spark worker

And we use Dockerfile to build up Spark cluster, Figure 3.8 shown in below.

```
FROM sequenceiq/hadoop-docker:2.6.0
MAINTAINER SequenceIQ

#support for Hadoop 2.6.0
RUN curl -s http://d3kbcqa49mib13.cloudfront.net/spark-1.6.1-bin-hadoop2.6.tgz | tar -xz -C /usr/local/
RUN cd /usr/local && ln -s spark-1.6.1-bin-hadoop2.6 spark
ENV SPARK_HOME /usr/local/spark
RUN mkdir $SPARK_HOME/yarn-remote-client
ADD yarn-remote-client $SPARK_HOME/yarn-remote-client

RUN $BOOTSTRAP && $HADOOP_PREFIX/bin/hadoop dfsadmin -safemode leave && $HADOOP_PREFIX/bin/hdfs dfs -put $SPARK_HOME-1.6.1-bin-hadoop2.6/lib /spark

ENV YARN_CONF_DIR $HADOOP_PREFIX/etc/hadoop
ENV PATH $PATH:$SPARK_HOME/bin:$HADOOP_PREFIX/bin
# update boot script
COPY bootstrap.sh /etc/bootstrap.sh
RUN chown root.root /etc/bootstrap.sh
RUN chmod 700 /etc/bootstrap.sh

#install R
RUN rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
RUN yum -y install R

ENTRYPOINT ["/etc/bootstrap.sh"]
```

FIGURE 3.8: Spark Dockerfile

3.2.4 Assign Tasks

We can launch a container with Hadoop environment directly on the platform to provide us the experiment environment. Here is the process of launch, after received command from the web user interface, system will use OpenStack api to deploy the basic environment.

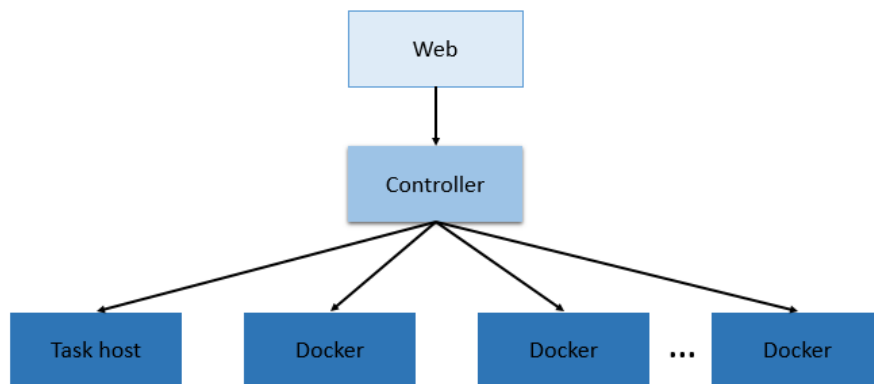


FIGURE 3.9: The process of launch

When the basic environment is completed, the host of task will start to assigning tasks to Docker vm. Then we can do some experiment by this way.

tasks.bb

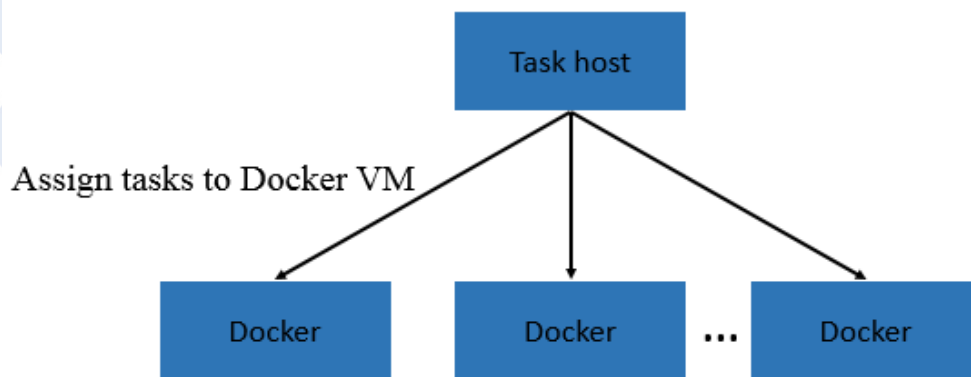


FIGURE 3.10: Assign tasks

When Docker vm finished tasks, tasks will be send back to the host of task, after the host of task sorts out the information, it will send to web service to present information to users. The process is shown in Figure 3.11.

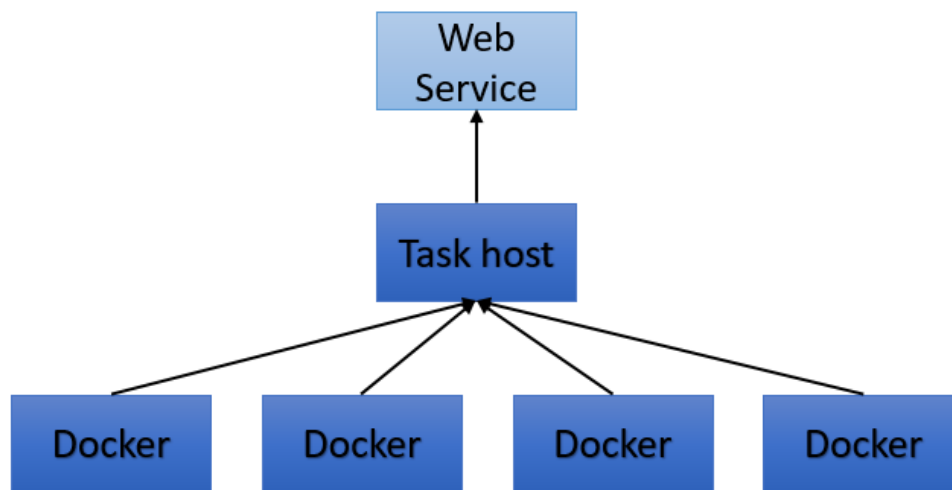


FIGURE 3.11: Send information back to web

3.2.5 User Services

In order to present status about containers, we designed a web site as our management platform. Because the web services is compatible with a variety of platform such as PC, mobile and tablet. We adopted Nginx, Nginx is free and open source software, it's a web server, which can also be used as a reverse proxy, load balancer and HTTP cache. Also, we used several techniques to help us build our platform, shown in Table 3.1.

TABLE 3.1: Software & language Specification

Software & language	Version
Nginx	1.13.1
Golang	1.8.3
Bootstrap	3.3.6
Python	2.7.6

In order to make site more beautiful, we use Bootstrap as our website framework. Bootstrap is a powerful front-end framework for faster and easier web development. It includes HTML and CSS based on design templates for common user interface components.



Chapter 4

Experimental Results

In this chapter, we show the experimental environment and the experimental results. In section 4.1, we describe our experimental environment including hardware specification and software specification. The experimental results are shown in section 4.2.

4.1 Experimental Environments

The experimental environment consists of three computers and their hardware specifications had listed in Table 4.1. The first physical machine consist of 12-core CPU, 30 GB memory, 2 TB disk and with Ubuntu 14.04 as the operating system. The hardware specification of No.2 and No.3 physical machine is the same: 32-core CPU, 64 GB memory, 4 TB disk and with Ubuntu 14.04 as the operating system. All experiments were measured on No.2 and No.3.

TABLE 4.1: Hardware specification

No.	CPU	RAM	HDD	OS
1	Intel(R) Core(TM) i7 CPU X 990	30GB	2TB	Ubuntu 14.04
2	AMD Opteron(TM) Processor 6274	64GB	2TB	Ubuntu 14.04
3	AMD Opteron(TM) Processor 6274	64GB	2TB	Ubuntu 14.04

Software specifications had listed in Table 4.2 . The Docker version is 1.13 The KVM version is 2.6.20.

TABLE 4.2: Software specification

Software	Docker	Sysbench	KVM	Hadoop	Spark
version	1.13	0.4.12	2.6.20	2.7.1	1.6.0

4.2 Boot-Time on OpenStack

We test the time of opening five VMs of KVM and Docker, the hypervisor of compute1 is Docker, the hypervisor of compute2 is KVM, OS is ubuntu14.04. Shown as Figure 4.1, it spent 55.54 seconds to turn on five VMs, but Docker only spent 9.7 seconds, it shows that time of deploy VMs of Docker is much less than KVM. If Docker use in the environment that need more amount of VMs, it definitely saved more time.

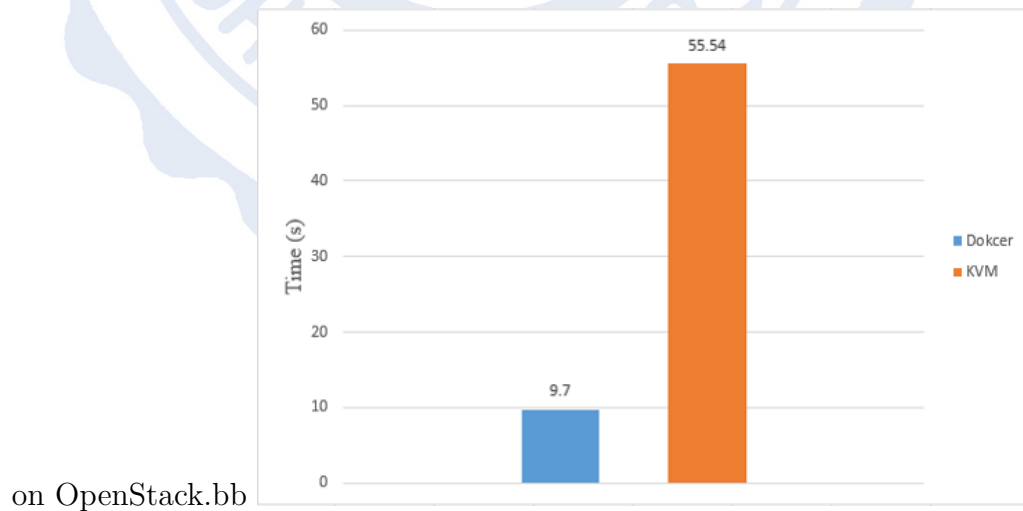


FIGURE 4.1: Boot-time on OpenStack

4.3 Docker and Virtual Machine Performance Comparison

Here we measured Docker and KVM performance on server boot time, reboot time and delete time. First of all is average server boot time. First we boot VM, then wait for VM to become active and repeat the above steps for a total of 15 VMs. Finally we delete all VMs. The result is shown in Figure 4.2 .

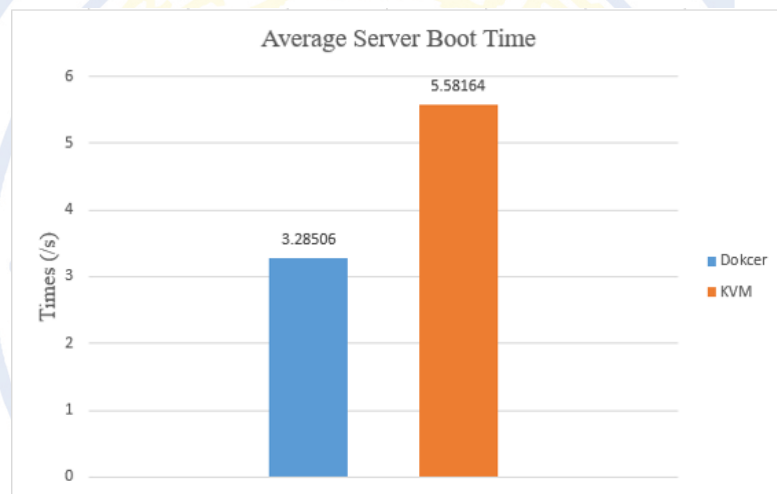


FIGURE 4.2: Average server boot time between KVM

About reboot time and delete time, we boot a VM and wait it to become active, then reboot the VM and wait it to become active, we repeat at reboot a total of 5 times. Next we delete VM and repeat the above for a total of 5 VMs. The result is shown in Figure 4.3 4.4.

4.3.1 CPU Utilization of Virtual Machine

In the VM performance experiment, we tried to find at what setting of the VMs vCPU utilization. In this experiment, we execute VMs by the High Performance Linpack (HPL). HPL has characteristic of a distribution system and use MPI to compute some data and finally it will produce a score. The resource of these ten VMs are all the same, 4-core CPU, 4GB memory, and 20GB disk. When the five VMs of KVM running HPL, the CPU utilization of compute2 is 61.4.

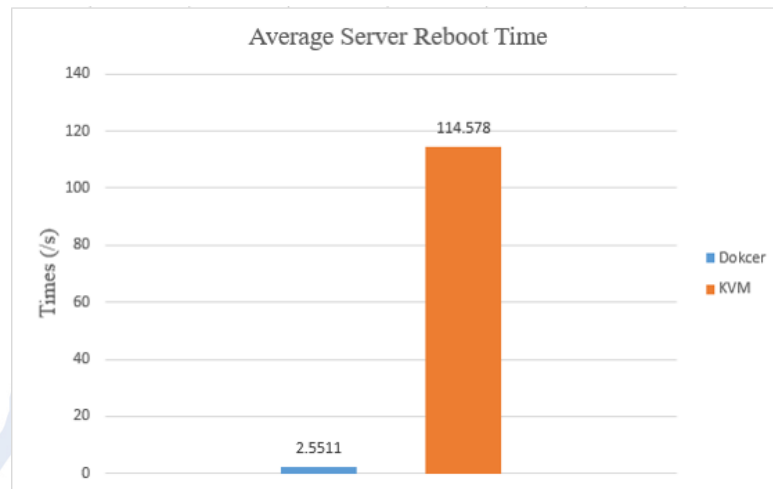


FIGURE 4.3: Average server reboot time between KVM

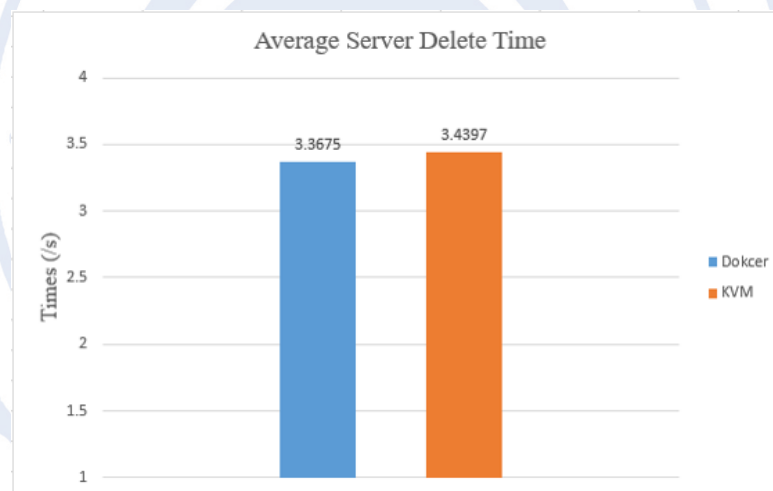


FIGURE 4.4: Average server delete time between KVM

4.3.2 File I/O Performance Comparison between Docker and KVM

When we deploying Hadoop cluster on Docker, we should make sure that read and write performance isn't less than the existing virtual machine technology, because Hadoop clusters need to storing and processing big data, it have high expectations for read and write performance. In order to verify the performance of the container and KVM, we did some experiment about it.

Sysbench is a scriptable multi-threaded benchmark tool based on LuaJIT. It is most frequently used for file I/o benchmarks, because the file is random read

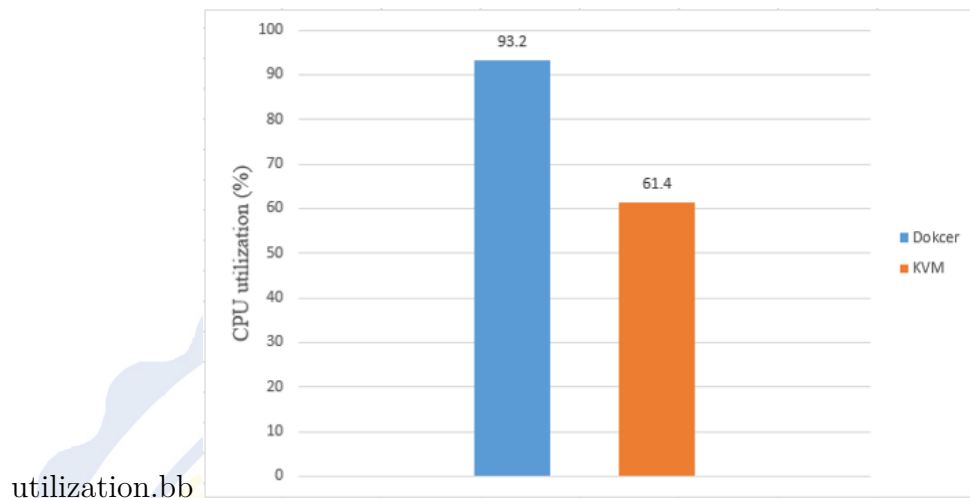


FIGURE 4.5: CPU utilization of virtual machine

and write, in order to achieve better results, the value of `/sys/block/sda/queue/scheduler` is set as `deadline` because of its minimum time-consuming. We randomly read and write 10GB file. Size is 10GB*1, 1.25GB*8, 0.625GB*16, 0.312GB*32. The result is shown in Figure 4.6 4.7 4.8. We also measured power usage during this experiment and the result is shown in Figure 4.9 and Table 4.3.

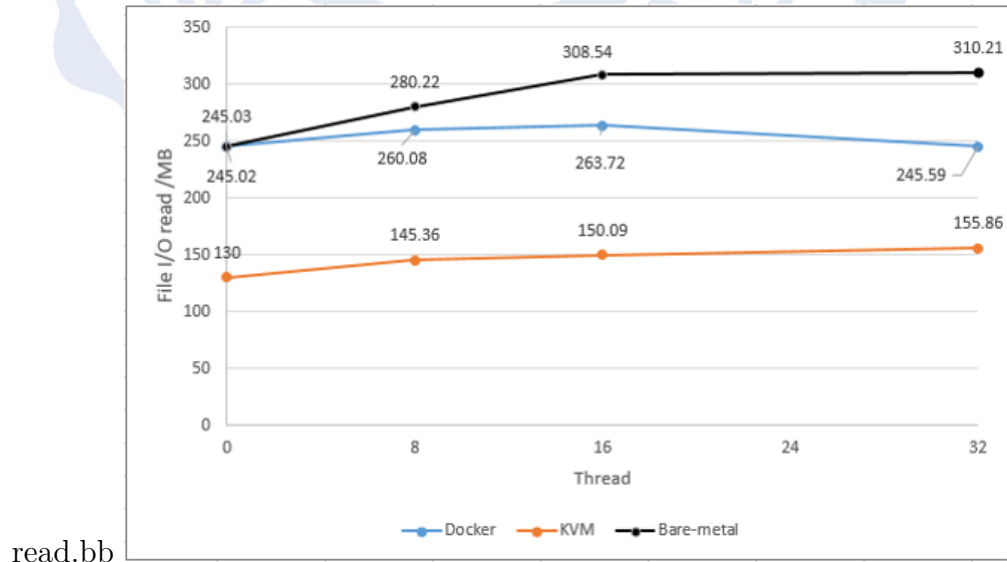


FIGURE 4.6: Read speed comparison between Docker and KVM

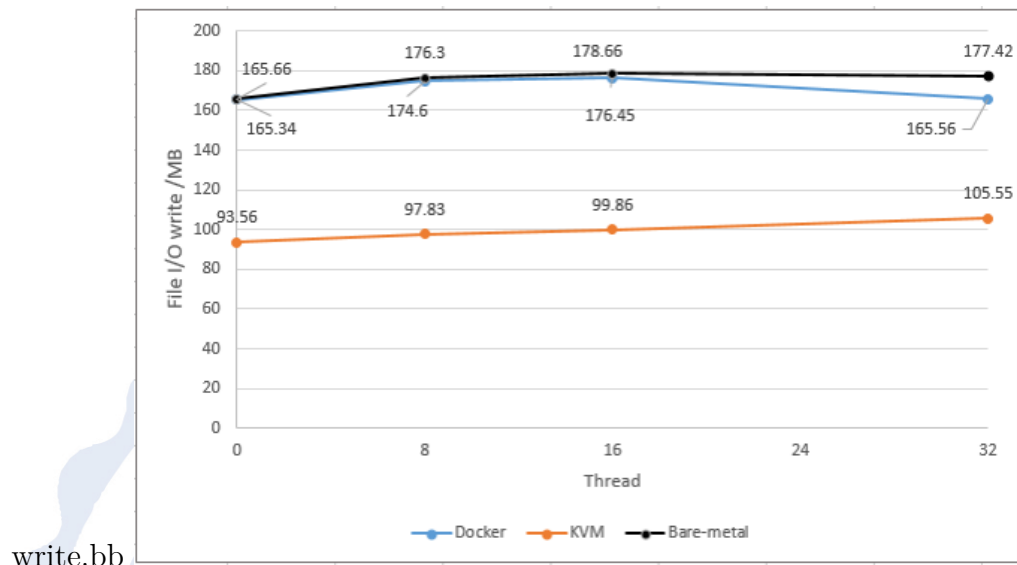


FIGURE 4.7: Write speed comparison between Docker and KVM

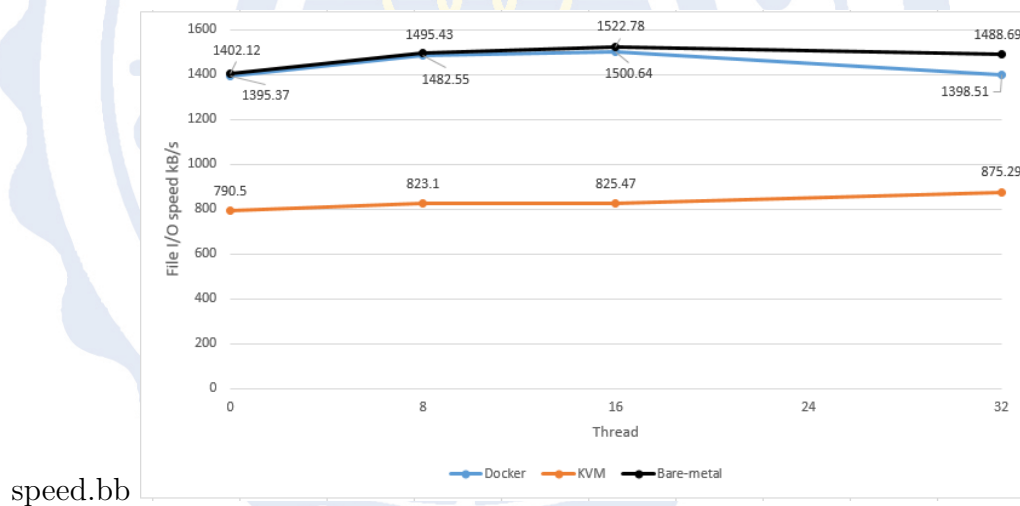


FIGURE 4.8: Read and Write speed comparison between Docker and KVM

From the experimental results we can see that Docker in I/O read and write performance is better than KVM. Therefore, in the same experiment, Docker performance will be close to virtual machine or even better.

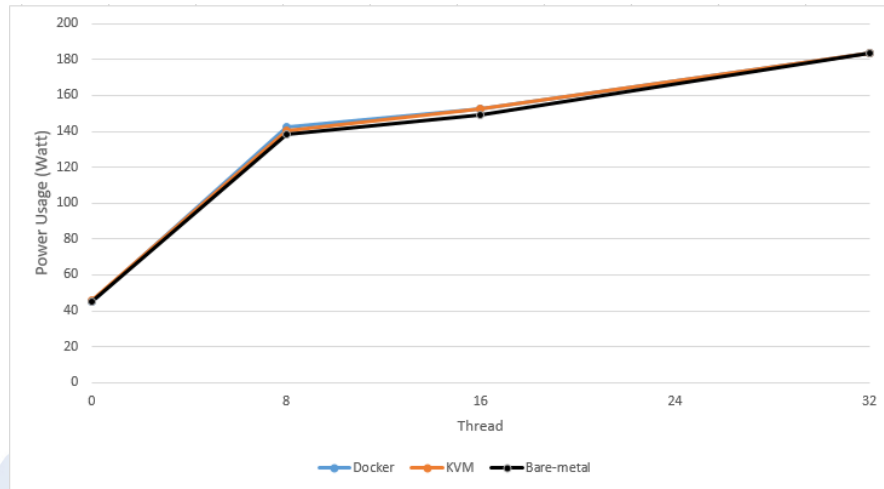


FIGURE 4.9: Power usage during the experiment

TABLE 4.3: The data of power usage

Thread	Bare-metal	KVM	Docker
0	45.31w	45.71w	45.35w
8	138.22w	140.09w	142.11w
16	149.37w	152.54w	152.47w
32	183.51w	183.53w	183.6w

4.4 Performance of Hadoop and Spark Comparison between Docker and Virtual Machine

In this test, we use HiBench to evaluate the performance of typical distributed data processing systems, We select two benchmarks, including WordCount and TeraSort as our workloads.

We introduce these two benchmarks as follows:

- **WordCount:** WordCount is a classical MapReduce workload, which counts the number of occurrences for each word in input text. In our test, the input data is 10GB and generated by RandomWriter and RandomTextWriter in Hadoop distribution.

- TeraSort: TeraSort is a classical workload also, which sorts massive data as fast as possible. In our test, the input data is 10GB and generated by TeraGen in Hadoop distribution.

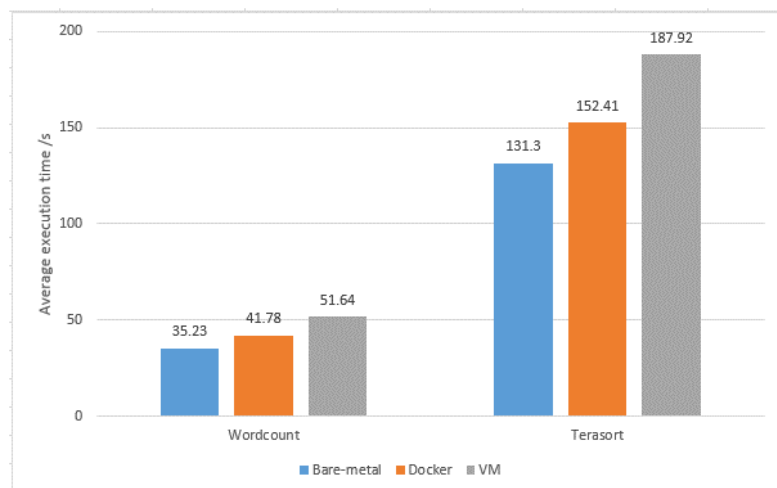
4.4.1 Execution Time of Hadoop and Spark on Docker and Virtual Machine

Figure 4.10 and Figure 4.11 shows the results of Hadoop and Spark system in different environment respectively. As you can see, running Hadoop directly on Docker could gain more efficacy than running in VM.



exe time.bb

FIGURE 4.10: Execution time of Hadoop in different workloads



exe time.bb

FIGURE 4.11: Execution time of Spark in different workloads

4.4.2 Deploying Hadoop in Different Environments

Deploy Hadoop in docker container and VM, then record each the time they took. The result is shown in Figure 4.12.

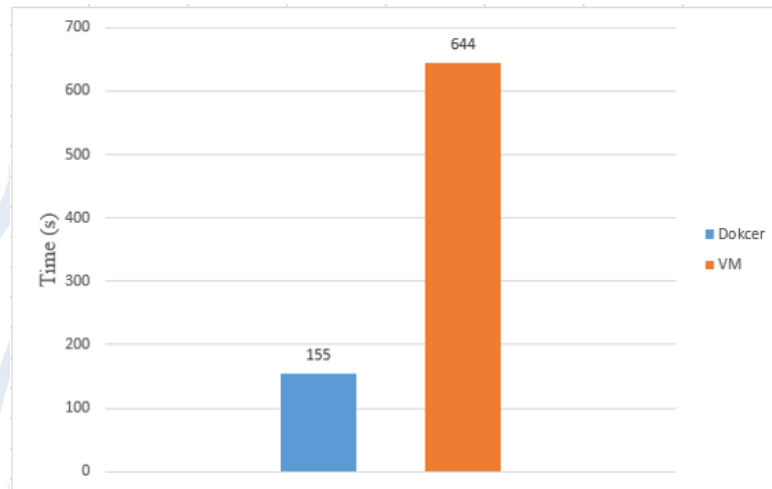


FIGURE 4.12: Hadoop start-up time

After all experiments, we understand that it will reduce deploying time by using Docker, and we also know Docker performance will be close to Bare-metal. According to the experimental result, we found that Docker won't saving power, it cost as much as other environments like bare-metal and virtual machine. So we provide these results to users to know more about Docker.

4.5 Container Management Platform

It's the main page of platform, Figure 4.17 shows the interface of container management platform on web site, it shows many status about containers, such as which container is running and what time you created it.

Users can do some actions to containers on web site directly, some actions like start the container, stop the container, restart it and remove it. More details shown in Figure 4.14.

Also, users can pull images, remove images, remove network, and remove volumes on web site directly.

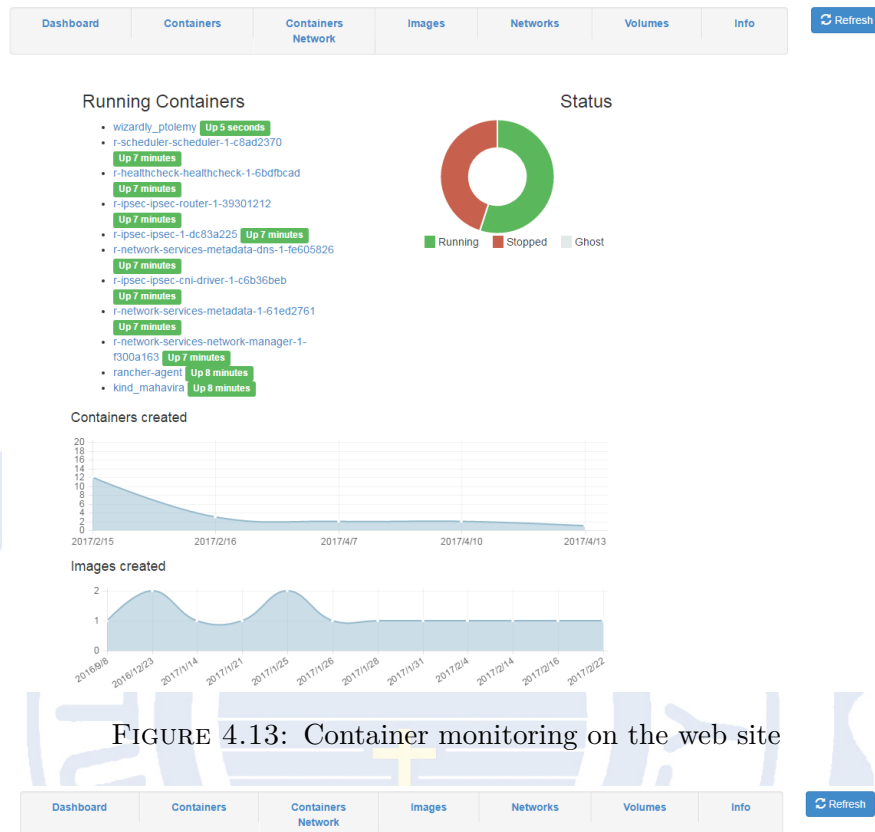


FIGURE 4.13: Container monitoring on the web site

Containers:

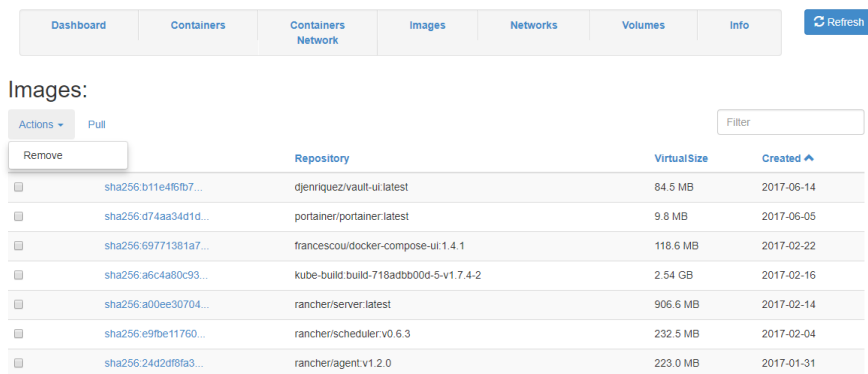
Actions: Start, Stop, Restart, Kill, Pause, Unpause, Remove

Display All Filter

	Image	Command	Created	Status	Log
	<code>uid/ui-for-docker</code>	<code>/ui-for-docker</code>	2017-07-24	Up 5 minutes	<code>stdout/stderr</code>
	<code>djenriquez/vault-ui</code>	<code>yam run serve</code>	2017-06-16	Up 5 weeks	<code>stdout/stderr</code>
	<code>atcol/docker-registry-ui</code>	<code>/bin/sh -c /usr/share/tomcat7/bin/cus...</code>	2017-06-16	Exited (137) 5 weeks ago	<code>stdout/stderr</code>
	<code>elastic_bhaskara</code>	<code>/bin/sh -c /usr/share/tomcat7/bin/cus...</code>	2017-06-16	Created	<code>stdout/stderr</code>
	<code>suspicious_dijkstra</code>	<code>/portainer</code>	2017-06-16	Exited (2) 5 weeks ago	<code>stdout/stderr</code>
	<code>wizardly_ptolemy</code>	<code>/ui-for-docker</code>	2017-04-13	Exited (255) 2 months ago	<code>stdout/stderr</code>
	<code>docker-compose-ui</code>	<code>/env/bin/python /app/main.py</code>	2017-04-10	Exited (255) 3 months ago	<code>stdout/stderr</code>
	<code>determined_galileo</code>	<code>/ui-for-docker</code>	2017-04-10	Exited (255) 3 months ago	<code>stdout/stderr</code>

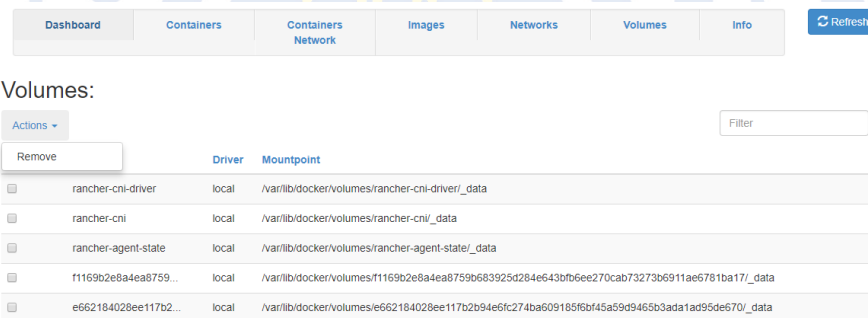
FIGURE 4.14: Do actions on web site directly

Finally, the web site shows information about users machine. For example, it shows how much containers or images did users create and how much memory of these containers or images took up.



Actions	Repository	VirtualSize	Created	
Remove	sha256:b11e4f6b7...	djenriquez/vault-ui:latest	84.5 MB	2017-06-14
	sha256:d74aa34d1d...	portainer/portainer:latest	9.8 MB	2017-06-05
	sha256:69771381a7...	francescoul/docker-compose-ui:1.4.1	118.6 MB	2017-02-22
	sha256:a6c4a80c93...	kube-build-718adb00d-5-v1.7.4-2	2.54 GB	2017-02-16
	sha256:a00ee30704...	rancher/server:latest	906.6 MB	2017-02-14
	sha256:e9fbc11760...	rancher/scheduler:v0.6.3	232.5 MB	2017-02-04
	sha256:24d2d8fa3...	rancher/agent:v1.2.0	223.0 MB	2017-01-31

FIGURE 4.15: Remove images on web site directly



Actions	Driver	Mountpoint
Remove	rancher-cni-driver	local /var/lib/docker/volumes/rancher-cni-driver/_data
	rancher-cni	local /var/lib/docker/volumes/rancher-cni/_data
	rancher-agent-state	local /var/lib/docker/volumes/rancher-agent-state/_data
	11169b2e8a4ea8759...	local /var/lib/docker/volumes/11169b2e8a4ea8759b683925d284e643bfb6ec270cab73273b6911ae6781ba17/_data
	e662184028ee117b2...	local /var/lib/docker/volumes/e662184028ee117b2b94e6fc274ba609185f6bf45a59d9465b3ada1ad95de670/_data

FIGURE 4.16: Remove volumes on web site directly

Containers:	25
Images:	29
Debug:	false
CPUs:	4
Total Memory:	1.94 GB
Operating System:	Ubuntu 16.04.2 LTS
Kernel Version:	4.4.0-75-generic
ID:	UFEO:YIZ7:B7TD:45UA:DCLU:VQCU:LJ2O:OE5B:6X7M:FPWC:OS2W:HMPJ
Labels:	
File Descriptors:	27
Goroutines:	30
Storage Driver:	aufs
Storage Driver Status:	Root Dir: /var/lib/docker/aufs Backing Filesystem: extfs Dirs: 163 Dirperm1 Supported: true

FIGURE 4.17: Users information

Chapter 5

Conclusions and Future Work

This work is stated for the concluding remarks and the future work of this work.

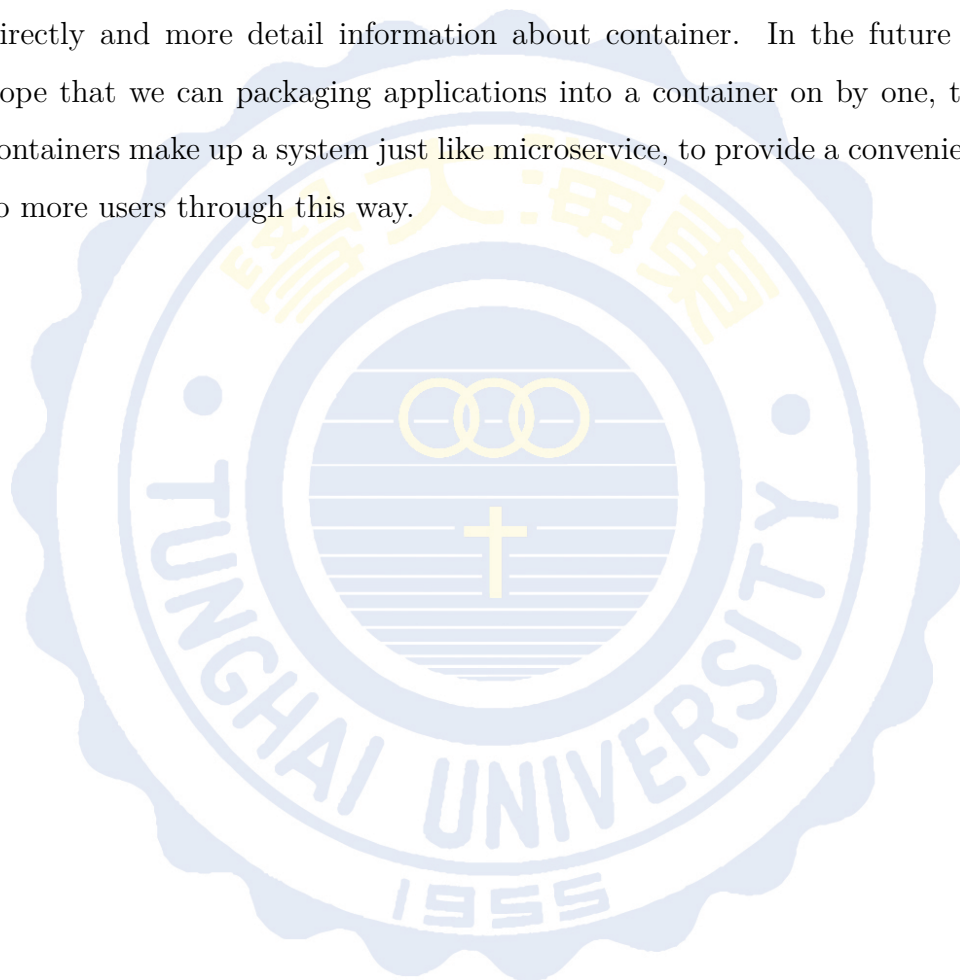
5.1 Concluding Remarks

In this work, we implemented a container management platform based on Docker, including container's status monitoring. Hadoop can take full advantage of distributed clusters, is the way to solve Big data storage and computing. Big data will be related to the distribution of physical nodes, how to deploy and manage these nodes quickly is also need to focus on this issue. Container-based technology reduce the complexity of deployment and improves deployment and maintenance efficiency. In addition, experimental results proves that Docker really can reduce the complexity of the process of its deployment and also save time.

So the combination with Docker must be able to provide a new way for Hadoop, or even other tools related to cloud services. We think containers can replace older simulation techniques, because it's lighter, faster and less hardware required.

5.2 Future Work

The platform still have some work to do. On the website, We plan to have more functions on it, such as deploying more open source technologies on the website directly and more detail information about container. In the future work, we hope that we can packaging applications into a container on by one, then these containers make up a system just like microservice, to provide a convenient service to more users through this way.



References

- [1] Rajkumar Buyya, Christian Vecchiola, and Thamarai Selvi. *Mastering Cloud Computing: Chapter 3 - Virtualization*. MORGAN KAUFMANN, 2013.
- [2] Xiaofei Liao, Hai Jin, Shizhan Yu, and Yu Zhang. A novel memory allocation scheme for memory energy reduction in virtualization environment. *Journal of Computer and System Sciences*, pages 3 – 15, 2015.
- [3] Yaozu Dong, Xiantao Zhang, Jinquan Dai, and Haibing Guan. Hyvi: A hybrid virtualization solution balancing performance and manageability. *Parallel and Distributed Systems*, pages 2332 – 2341, 2014.
- [4] Ben Pfaff, Justin Pettit, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. *Extending Networking into the Virtualization Layer*, 2009.
- [5] František Špaček, Radomír Sohlich, and Tomáš Dulík. Docker as platform for assignments evaluation. *Energy Procedia*, pages 1665–1671, 2015.
- [6] Build, ship and run any app, anywhere, 2015. <https://www.docker.com/>.
- [7] Docker (software), 2015. <http://en.wikipedia.org/wiki/Docker%28software%29>.
- [8] Di Liu and Libin Zhao. The research and implementation of cloud computing platform based on docker. *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on*, pages 475–478, 2014.

- [9] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pages 171–172, 2015.
- [10] Gaku Nakagawa and Shuichi Oikawa. Behavior-based memory resource management for container-based virtualization. *Proceedings - 4th International Conference on Applied Computing and Information Technology, 3rd International Conference on Computational Science/Intelligence and Applied Informatics, 1st International Conference on Big Data, Cloud Computing, Data Science and Engineering, ACIT-CSII-BCD 2016*, pages 213–217, 2017.
- [11] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 275–287, 2007.
- [12] Hypervisor, 2015. <http://en.wikipedia.org/wiki/Hypervisor>.
- [13] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: A performance comparison. pages 386–393, 2015.
- [14] What is apache hadoop?, 2016. <http://hadoop.apache.org/>.
- [15] Apache hadoop, 2016. <http://wiki.apache.org/hadoop/>.
- [16] V. Starikovičius A. and Maknickas M. Big data and hadoop-a study in security perspective. *Procedia Computer Science*, pages 598–601, 2015.
- [17] A. Kačeniauskas, R. Pacevič, V. Starikovičius, A. Maknickas, M. Staškūnienė, and G. Davidavičius. Development of cloud services for patient-specific simulations of blood flows through aortic valves. *Advances in Engineering Software*, pages 57–64, 2017.
- [18] Aymen Jlassi and Patrick Martineau. Benchmarking Hadoop Performance in the Cloud - An in Depth Study of Resource Management and Energy

- Consumption. In *The 6th International Conference on Cloud Computing and Services Science*, ROME, Italy, April 2016.
- [19] Spark, 2015. <http://en.wikipedia.org/w/index.php?title=Spark&oldid=654641608>.
- [20] Openstack open source cloud computing software, 2015. <http://www.openstack.org/>.
- [21] What is openstack?, 2015. <http://opensource.com/resources/what-is-openstack>.
- [22] Openstack, 2015. <http://en.wikipedia.org/wiki/OpenStack>.
- [23] Zhaojun Li, Haijiang Li, Xicheng Wang, and Keqiu Li. A generic cloud platform for engineering optimization based on openstack. *Advances in Engineering Software*, pages 42 – 57, 2014.
- [24] Yoji Yamato, Masahito Muroi, Kentaro Tanaka, and Mitsutomo Uchimura. Development of template management technology for easy deployment of virtual resources on openstack. *Journal of Cloud Computing*, 3(1):1–12, 2014.
- [25] Yoji Yamato, Yukihiisa Nishizawa, Masahito Muroi, and Kentaro Tanaka. Development of resource management server for production iaas services based on openstack. *Journal of Information Processing*, 23(1):58–66, 2015.
- [26] Power distribution unit, 2015. http://en.wikipedia.org/wiki/Power_distribution_unit.
- [27] What is power distribution unit?, 2013. <http://searchdatacenter.techtarget.com/definition/power-distribution-unit-PDU>.
- [28] Preeth E N, Fr. Jaison Paul Mulerickal, Biju Paul, and Yedhu Sastri. Evaluation of docker containers based on hardware utilization. *Control Communication & Computing India (ICCC), 2015 International Conference on*, pages 697–700, 2015.

- [29] Kyoung-Taek Seo, Hyun-Seo Hwang, Il-Young Moon, Oh-Young Kwon, and Byeong-Jun Kim. Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters (ASTL), 2014 Networking and Communication*, pages 105–111, 2014.
- [30] P. China Venkanna Varma, K.V. Kalyan Chakravarthy, V. Valli Kumari, and S. Viswanadha Raju. Analysis of network io performance in hadoop cluster environments based on docker containers. *Advances in Intelligent Systems and Computing*, pages 227–237, 2016.
- [31] Javier Conejero, Omer Rana, Peter Burnap, Jeffrey Morgan, Blanca Caminero, and Camen Carrión. Analyzing hadoop power consumption and impact on application qos. *Future Generation Computer Systems*, pages 213–223, 2016.

Appendix A

Hadoop Installation

I. Modify hosts

```
# sudo vim /etc/hosts
```

II. Modify hostname

```
# sudo vim /etc/hostname  
# sudo service hostname start
```

III. Install Java JDK

```
# sudo apt-get -y install openjdk-7-jdk  
# sudo ln -s /usr/lib/jvm/java-7-openjdk-amd64 /usr/lib/jvm/jdk
```

IV. Add hadoop user

```
# sudo addgroup hadoop  
# sudo adduser --ingroup hadoop hduser  
# sudo adduser hduser sudo
```

V. Creat SSH authentication login

```
# ssh-keygen -t rsa -f \~{}/.ssh/id\_{}rsa -P ""
# cp \~{}/.ssh/id\_{}rsa.pub \~{}/.ssh/authorized\_{}keys
# scp -r \~{}/.ssh hduser:~/
```

VI. Download hadoop

```
# cd ~
# wget http://ftp.twaren.net/Unix/Web/apache/hadoop/common \\\
  /hadoop-2.6.0/hadoop-2.6.0.tar.gz
# tar xzf hadoop-2.6.0.tar.gz
# mv hadoop-2.6.0.tar.gz hadoop
```

VII. Add the environment variable

```
# vim .bashrc

export JAVA_HOME=/usr/lib/jvm/jdk/
export HADOOP_INSTALL=/home/hduser/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
```

VIII. Set hadoop config

```
# cd hadoop/etc/hadoop
# vim hadoop-env.sh

export JAVA_HOME=/usr/lib/jvm/jdk/

# vim core-site.xml

<property>
  <name>fs.default.name</name>
  <value>hdfs://hadoop-master:9000</value>
</property>

# vim yarn-site.xml

<property>
```

```
<name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>hduser</value>
</property>

# cp mapred-site.xml.template mapred-site.xml
# vim mapred-site.xml

<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>

# mkdir -p ~/mydata/hdfs/namenode
# mkdir -p ~/mydata/hdfs/datanode
# vim hdfs-site.xml

<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/home/hduser/mydata/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/home/hduser/mydata/hdfs/datanode</value>
</property>

# vim slaves
hadoop-master
node01
node02
node03
node04
node05
node06
node07
node08
node09
node10
node11
node12
```

IX. Copy hadoop to all nodes

```
# scp -r /home/hduser/hadoop node01:/home/hduser
# scp -r /home/hduser/hadoop node02:/home/hduser
# scp -r /home/hduser/hadoop node03:/home/hduser
# scp -r /home/hduser/hadoop node04:/home/hduser
# scp -r /home/hduser/hadoop node05:/home/hduser
# scp -r /home/hduser/hadoop node06:/home/hduser
# scp -r /home/hduser/hadoop node07:/home/hduser
# scp -r /home/hduser/hadoop node08:/home/hduser
# scp -r /home/hduser/hadoop node09:/home/hduser
# scp -r /home/hduser/hadoop node010:/home/hduser
# scp -r /home/hduser/hadoop node011:/home/hduser
# scp -r /home/hduser/hadoop node012:/home/hduser
```

X. Format HDFS

```
# hdfs namenode -format
```

XI. Start hadoop

```
# start-all.sh
```

XII. Use jps to see java running program

```
# jps
```

XIII. MapReduce JobTracker monitoring website

```
# hadoop-master:50030
```

Appendix B

Spark Installation

I. Download and Unzip Scala

```
#wget \\  
http://ftp.twaren.net/Unix/Web/apache/spark/spark-1.4.1/spark-1.4.1-bin-hadoop2.6.tgz  
#tar zxf spark-1.4.1-bin-hadoop2.6.tgz  
#mv spark-1.4.1-bin-hadoop2.6 spark  
#cd spark/conf
```

IV. Set Spark config

```
#vim spark-env.sh  
export SCALA_HOME=/usr/lib/scala  
export JAVA_HOME=/usr/lib/jvm/jdk  
export SPARK_MASTER=master  
export HADOOP_HOME=/home/hduser/hadoop  
export SPARK_HOME=/home/hduser/spark  
export SPARK_LIBRARY_PATH=.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib:$HADOOP_HOME/lib/native  
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop  
  
#vim slaves  
hadoop-master  
node01  
node02  
node03  
node04  
node05  
node06
```

```
node07
node08
node09
node10
node11
node12
```

III. Copy spark to all nodes

```
# scp -r /home/hduser/spark node01:/home/hduser
# scp -r /home/hduser/spark node02:/home/hduser
# scp -r /home/hduser/spark node03:/home/hduser
# scp -r /home/hduser/spark node04:/home/hduser
# scp -r /home/hduser/spark node05:/home/hduser
# scp -r /home/hduser/spark node06:/home/hduser
# scp -r /home/hduser/spark node07:/home/hduser
# scp -r /home/hduser/spark node08:/home/hduser
# scp -r /home/hduser/spark node09:/home/hduser
# scp -r /home/hduser/spark node10:/home/hduser
# scp -r /home/hduser/spark node11:/home/hduser
# scp -r /home/hduser/spark node12:/home/hduser
# bin/start-hbase.sh
```

Appendix C

Docker Installation

I. Update first

```
# sudo apt-get update
```

II. Then Upgrade

```
# sudo apt-get upgrade
```

III. Download Docker

```
# curl -sSL https://get.docker.com/ubuntu/ | sudo sh
```

IV. Start up Docker

```
# sudo service docker start
```

V. Check out the version of Docker

```
# sudo docker version
```

VI. Verify that Docker is installed successfully

```
# sudo docker info  
# sudo ifconfig docker0
```



Appendix D

Nova-Docker Installation

I. Modify nova-compute.conf on compute node

```
# [DEFAULT]
# /#compute_driver=libvirt.LibvirtDriver
# compute_driver=novadocker.virt.docker.DockerDriver
# [libvirt]
# /#virt_type=kvm
# virt_type=docker
```

II. Install git

```
# sudo apt-get install git
```

III. Get Nova-Docker by git clone

```
# git clone -b stable/kilo https://github.com/stackforge/nova-docker.git
```

IV. Install pip

```
# sudo apt-get install python-pip
```

V. Check out the version of Nova-Docker

```
# sudo pip list | grep nova-docker
```

V. Add docker.filters to rootwrap

```
# sudo cp nova-docker/etc/nova/rootwrap.d/docker.filters  
# /etc/nova/rootwrap.d/
```

VI. Modify docker.sock

```
# chmod 666 /var/run/docker.sock
```

VI. Nova-Docker restart

```
# service nova-compute restart
```

Appendix E

PDU Information program

```
<?php
function get_server_info($host, $community, $objectid) {
$a = snmpget($host, $community, $objectid);

$tmp = explode(":", $a);
if (count($tmp) > 1) {
$a = trim($tmp[1]);
}
return $a;
}

$host="IP";
$community="public";

for($i=1;$i<=8;$i++){
$Power = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.7.".$i);
echo $i."-Power: ".$Power."<br>";

$I = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.4.".$i);
echo $i."-I: ".$I."<br>";

$V = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.6.".$i);
echo $i."-V: ".$V."<br>";

$PF = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.9.".$i);
echo $i."-PF: ".$PF."<br>";
$sql="INSERT INTO `ServerMonitor`.`Power` (`ID`, `TIME`, `SID`, `V`, `C`, `P`, `PF`) \\  
VALUES (NULL, CURRENT_TIMESTAMP, ' ".$i."', ' ".$V."', ' ".$I."', ' ".$Power."', ' ".$PF."') \\  
";
//echo $sql;
mysql_query($sql) or die('MySQL query error');
$sql="UPDATE `ServerMonitor`.`PowerRealTime` SET `TIME` = CURRENT_TIMESTAMP(), \\  

```

```
`V` = '". $V."', `C` = '". $I."', `P` = '". $Power."', \\
`PF` = '". $PF.'" WHERE `PowerRealTime`.`ID` = '". $i.'";
mysql_query($sql) or die('MySQL query error');
}
?>
```

