НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут телекомунікаційних систем
Кафедра Інформаційно-телекомунікаційних мереж

До захисту допущено:
в.о. завідувача кафедри
_____ Лариса ГЛОБА
«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційно-комунікаційні технології»
спеціальності 172 «Телекомунікації та радіотехніка»
на тему: «Віртуалізація серверів за допомогою хостів контейнерів»

Виконала:
студентка IV курсу, групи ТІ-72
Кузнецова Анастасія Андріївна _____

Керівник:
Професор кафедри ІТМ, д.т.н, с.н.с.
Скулиш Марія Анатоліївна _____

Рецензент:
Доцент кафедри телекомунікацій с.н.с. к.т.н
Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

Кафедра Інформаційно-телекомунікаційних мереж

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інформаційно-комунікаційні технології»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Лариса ГЛОБА

«___»_____2021 р.

ЗАВДАННЯ

Кузнецової Анастасії Андріївни

1. Тема роботи «Віртуалізація серверів за допомогою хостів контейнерів», керівник роботи Скулиш Марія Анатоліївна, професор кафедри інформаційно-телекомунікаційних мереж ІТС, професор, д.т.н., с.н.с,, затверджені наказом по університету від «14» квітня 2021 р. № 1007-с

2. Термін подання студенткою роботи 7 червня 2021 р.

3. Вихідні дані до роботи: використання програми CPU, Coreos і Centos, запити SQL та HTTP, тестова система SQL-insert, SQL-Select, Linux, наукові твори та статті про віртуалізацію серверів.

4. Зміст роботи:

1. Провести аналіз технологій віртуалізації.
2. Провести аналіз високопродуктивних методів віртуалізації серверів.
3. Створити тимчасову машину.
4. Дослідити емпіричний метод збору даних для виміру продуктивності віртуалізованих контейнерах хостах.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

- титульний слайд;
- актуальність;
- мета, завдання дипломної роботи;
- виконані завдання;
- аналіз технологій віртуалізації;
- опис методу контейнеризації автоматизованої структури;
- опис технологій, які використовуються у дипломній роботі;
- створення тимчасової машини;
- новизна дипломної роботи;
- загальні висновки.

7. Дата видачі завдання 29 вересня 2020 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Підбір літератури з теми дослідження	01.10.2020-21.10.2020	Виконано
2	Пошук інформації про існуючі стандарти, їх можливості та недоліки, ознайомлення з документацією роботи.	05.11.2020-30.11.2020	Виконано
3	Написання розділу : Аналіз технологій віртуалізації.	01.12.2020-31.12.2020	Виконано
4	Ознайомлення з віртуалізацією рівня операційної системи.	14.01.2021-10.02.2021	Виконано
5	Написання розділу: Аналіз високопродуктивних методів віртуалізації серверів з хостами контейнерів.	15.02.2021-17.03.2021	Виконано
6	Написання розділу: Загальний метод контейнеризації автоматизованої структири.	20.03.2021-10.04.2021	Виконано
7	Написання розділу: Емпіричний метод збору даних для виміру продуктивності у віртуалізованих контейнерах хостів.	10.04.2021-25.04.2021	Виконано
8	Оформлення дипломної роботи.	25.04.2021-15.05.2021	Виконано

Студентка Анастасія КУЗНЕЦОВА

Керівник Марія СКУЛИШ

РЕФЕРАТ

Робота містить 65 сторінок, 28 рисунків. Було використано 29 джерел.

Мета роботи: покращити продуктивність обробки даних у віртуальних контейнерах, дослідити, як за допомогою хостів контейнерів відбувається віртуалізація устаткування, проаналізувати та дослідити технології для кращої підтримки віртуального контейнера.

Об'єкт дослідження: програми CPU, Coreos і Centos, запити SQL та HTTP, тестова система SQL-insert, SQL-Select, Linux.

Предмет дослідження: віртуалізація серверів, збір даних, знаходження їх, вирішення проблеми з віртуалізованим устаткуванням. Крім того досліджуються деякі широко використовувані методи віртуалізації серверівтазбіру даних для вимірювання продуктивності у віртуалізованих контейнерах хостів.

В представленій роботі розглянуті та наведені деякі методи та основні операційні системи для їх віртуалізованого тестування.

Ключові слова: віртуалізація, контейнеризація, хмарні обчислення, контейнер, контейнер Linux, продуктивність, віртуальна машина, WordPress, Linux, Apache, MySQL, PHP, Coreos, Photon OS, VMware, контейнер для програмного забезпечення, Docker, апаратна віртуалізація.

ABSTRACT

The work contains 65 pages, 28 figures. 29 sources were used.

Purpose: to improve the performance of data processing in virtual containers, to investigate how equipment is virtualized with the help of container hosts, to analyze and explore technologies for better support of the virtual container.

Object of research: CPU, Coreos and Centos programs, SQL and HTTP queries, SQL-insert test system, SQL-Select, Linux.

Subject of research: virtualization of servers, data collection, finding them, solving the problem with virtualized equipment. In addition, some widely used methods of server virtualization data collection to measure performance in virtualized host containers.

Some methods and basic operating systems for their virtualized testing are considered in the presented work.

Keywords: Virtualization, Containerization, Cloud Computing, Container, Container Linux, Performance, Virtual Machine, WordPress, Linux, Apache, MySQL, PHP, Coreos, Photon OS, VMWare, Software Container, Docker, Hardware Virtualization.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ..... 8

ВСТУП……9

РОЗДІЛ 1 АНАЛІЗ ТЕХНОЛОГІЙ ВІРТУАЛІЗАЦІЇ …..……..11

1.1. Технологія віртуалізації.…….……11

1.2. Віртуалізація рівня операційної системи…….……..12

1.3. Огляд контейнерів в середовищі Linux…….……13

1.4. Особливості ядра…….……..…….……14

1.5. Переваги контейнерів…….……15

1.6. Контейнерні технології……18

РОЗДІЛ 2 АНАЛІЗ ВИСОКОПРОДУКТИВНИХ МЕТОДІВ ВІРТУАЛІЗАЦІЇ СЕРВЕРІВ.....21

2.1. Модернізація традиційних додатків Docker (MTA) …….....21

2.1.2.App2Cloud…….....23

2.1.3.Інструменти з відкритим вихідним кодом…….....23

2.1.4. VM2 Docker……..……..23

2.1.5.Image2 Docker…….……25

2.2. Хмарні обчислення…….....28

2.2.1. Віртуалізація…….……29

2.2.2. Віртуалізація апаратного забезпечення…….....30

2.3. Контейнеризація…….....33

2.4.VMware Vsphere…….....35

2.5. Опис існуючих систем……..….....36

РОЗДІЛ 3 ЗАГАЛЬНИЙ МЕТОД КОНТЕЙНЕРИЗАЦІЇ АВТОМАТИЗАВАНОЇ СТРУКТРИ.....39

3.1.Збір інформації……..….....39

3.1.1.Створення тимчасової машини…….....41

3.1.2.Порівняння машин…….….....40

3.1.3.Вирішення залежностей пакету…….....41

3.1.4.Управління пакетами……..…...42
3.1.5.Файли та конфігурація додатків…….…....42
3.1.6.Створення контейнерного зображення…….....43
3.2.Автоматизація Framework…….....44
3.2.1.Драйвер……...45
3.2.2.Агрегатор ……...45
3.2.3.Пошук…….……45
3.2.4.Конструктор зображень…….....46
РОЗДІЛ 4 ЕМПІРИЧНИЙ МЕТОД ЗБОРУ ДАНИХ ДЛЯ ВИМІРУ ПРОДУКТИВНОСТІ У ВІРТУАЛІЗОВАНИХ КОНТЕЙНЕРАХ ХОСТІВ...47
4.1.Емпіричний метод…….....……..47
4.2.Опис методу……..……47
4.2.1.Топологія……48
4.2.2.Обладнання для апаратного забезпечення ……49
4.2.3.Запити HTTP ……51
4.2.4.SQL Queries……....53
4.3.Надійність та дійсність……..…55
4.4.Результати та аналіз…….…56
4.4.1.Запити HTTP ……..56
4.4.2.SQL Queries……..…58
ЗАГАЛЬНІ ВИСНОВОКИ ПО РОБОТІ……62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……...63

ПЕРЕЛІК СКОРОЧЕНЬ

VM (Virtual Machine) – віртуальна машина

LXC (Linux Container Technology) топологія контейнерів операційної системи Linux

KVM (Kernel Virtual Machine) – віртуальна машина ядра

OS (Operating System) – операційна система

AWS (Amazon Web Services) – Amazon веб-сервіс

GCP (Google Cloud Platform) – хмарна платформа

CPU (Central Processing Unit) – центральний процесор

PID (Process Identifier) – індифікатор процесу

IPC (Inter Process Communication) – між процесорна взаємодія

TCP (Transmission Control Protocol) – протокол управління передачею

SSH (Secure Shell) – базова оболонка

ВСТУП

Загалом віртуалізація - це техніка, яка використовується для абстрактної операційної системи з апаратного забезпечення. Інша технологія це може бути використане для досягнення подібних цілей - тобто контейнеризація. Контейнеризація – це техніка віртуалізації рівня операційної системи, яка дозволяє запускати програми счстково ізольовані на тому ж апаратному забезпеченні. Контейнеризовані програми поділяють те ж сааме ядро Linux, але запускається з упакованих контейнерів, що включає в себе достатньо бінарів і бібліотек для функціонування програми. Останнім часом він став більш поширеним щоб побачити апаратну віртуалізацію під діючими операцій-контейнером.

Віртуалізація домінує за рахунок того, що пропонує різні послуги, які використовуються та управляється через Інтернет. Як правило, програми запускаються всередині віртуальних машин в ізольованому середовищі. Однак завжди існує апаратна віртуалізація. Для запуску таких віртуальних машин необхідно багато затрат, тому у своїй роботі пропоную рішення даної проблеми. Метою моєї роботи є покращити продуктивність обробки даних у віртуальних контейнерах, дослідити, як за допомогою хостів контейнерів відбувається віртуалізація устаткування, проаналізувати та дослідити технології для кращої підтримки віртуального контейнера.

Об’єкт дослідження: програми CPU, Coreos і Centos, запити SQL та HTTP, тестова система SQL-insert, SQL-Select, Linux.

Предметом дослідження є віртуалізація серверів, збір даних, знаходження їх, вирішення проблеми з віртуалізованим устаткуванням. Крім того досліджуються деякі широко використовувані методи віртуалізації серверівтазбіру даних для вимірювання продуктивності у віртуалізованих.

Для досягнення поставленої мети були вирішені наступні задачі:

1. Проаналізувати технології віртуалізації;
2. Проаналізувати високопродуктивні методи віртуалізації серверів;
3. Дослідити та розробити загальний метод контейниризації автоматизованої структури;
4. Створити тимчасову машину;
5. За допомогою емпіричного методу збору даних виміряти продуктивність у віртуалізованих контейнерах хостах, дослідити та розробити більш точний, але простий процес.

У моїй роботі показана доцільність та ефективність підходу створення прототипу автоматизованої структури, яка використовує підхід з людською взаємодією.

Технологія для подальшого розвитку є інтегрованими контейнерами VMware, які спрямовані на інтеграцію управління контейнерами Linux з Vsphere (апаратна платформа віртуалізації за допомогою VMware) інтерфейс управління. Виконуємо на трьох різних операційних системах (Centos, CoreOS та Photon OS), щоб побачити, чи вибір контейнерний хост впливає на продуктивність.

Наші результати показують зменшення продуктивності при порівнянні апаратного віртуалізованого хосту контейнерів до контейнерних хостів безпосередньо на апаратному забезпеченні.

Тому важливо розглянути три пункти, а саме : фактичне застосування, вибір операційної системи, тип операції.

РОЗДІЛ 1
АНАЛІЗ ТЕХНОЛОГІЙ ВІРТУАЛІЗАЦІЇ

1.1. Технологія віртуалізації

Технологія віртуалізації відноситься до абстракції обчислювальних ресурсів, таких як CPU, зберігання, мережа, пам'ять, стек застосування та база даних з додатків та звісно кінцеві користувачі, які користуються цими послуги. Технологія віртуалізації спирається на компоненти програмного забезпечення (ПО) для моделювання апаратного функціонування шляхом створення віртуальних ресурсів. У сучасному обчислювальному світі віртуалізація в основному використовується в апаратному та в рівнях операційної системи (ОС).

Віртуалізація рівня апаратного забезпечення широко використовувалася протягом останнього десятиліття для реалізації віртуалізації. Вона використовує гіпервізор, щоб створити абстрактний шар між програмним забезпеченням з основним обладнанням. Гіпервізор - це шматок програмного забезпечення, що створює віртуальні ресурси, які працюють безпосередньо на апаратному забезпеченні з обов'язковою системою головного адміністратора. Гіпервізор класифікується на два типи:

Тип 1 - рідний гіпервізор, тобто запускаються безпосередньо на обладнанні-хазяїна без необхідності ОС. Він обробляє доступ до апаратних ресурсів, зроблених гостьовими операційними системами, що діють як посередник.

Тип 2 - розміщений гіпервізор, тобто працює всередині приймаючої ОС, тим самим забезпечуючи доступ до апаратного забезпечення для гостей, та створює повну ізоляцію та абстракцію з апаратного забезпечення господаря. Цей тип віртуалізації також відомий як повна віртуалізація.

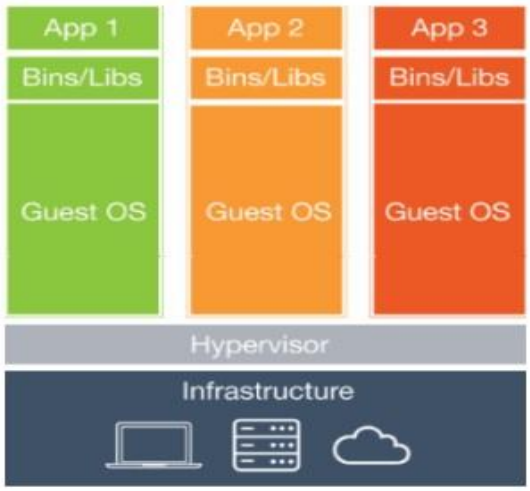

Рис.1.1 Тип 1: рідний гіпервізор

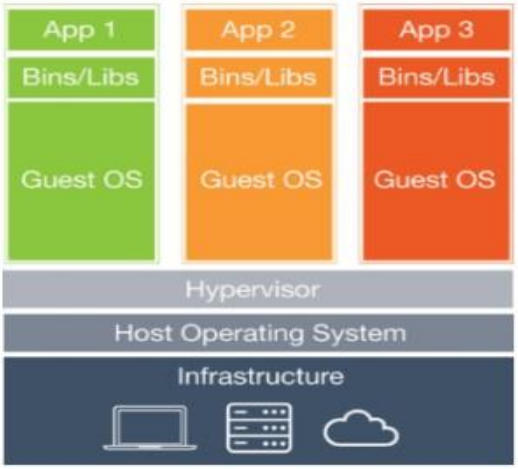

Рис. 1.2 Тип 2: розміщений гіпервізор

1.2. Віртуалізація рівня операційної системи

Віртуалізація рівня операційної системи набула значних змін за останні кілька років. Віртуалізація апаратного рівня вважається важкою, оскільки вона спирається на апаратну емуляцію. Альтернативно, у віртуалізації рівня ОС-рівня є контейнерний двигун, який використовує функції ядра, такі як CGroups, простір імен, CHROOT та ін. для створення ізольованих користувацьких хмарних екземплярів, відомих як контейнери, на вершині машини хоста, як показано на рисунку 3. Контейнери поділяють хост-ядро машин за допомогою контейнерного двигуна замість того, щоб працювати на

повну операційну систему, як це робить у віртуалізації на основі гіпервізора, тим самим зменшуючи його.

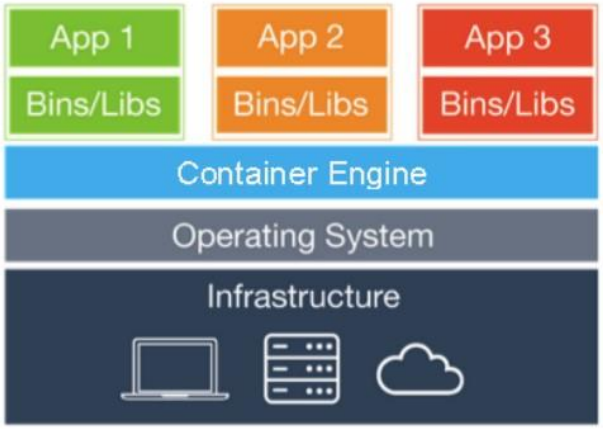

Рис. 1.3 ОС- рівень віртуалізації

Контейнеризація була реалізована в операційній системі Unix ще в 1979 році. Пізніше, оскільки технологія контейнеризації розвиваються, були більш суттєві риси, які реалізовані для ізоляції файлової системи, користувачів та мереж. Перша повна реалізація менеджера контейнера Linux є LXC1. Пізніше, Docker. Повна система для управління контейнерами Rkt3. Rkt3 був створений coreOS для подолання недоліків Docker.

1.3. Огляд контейнерів в середовищі Linux

Контейнери використовуються для забезпечення ізоляції та низької накладності в середовищі Linux. Інтерфейс управління або контейнерний двигун взаємодіє з компонентами ядра та забезпечує інструменти для створення та управління контейнерами, як показано на рисунок 5. Оскільки контейнери легкі, ми можемо досягти більшої щільності та кращої продуктивності у порівнянні з віртуальними машинами.

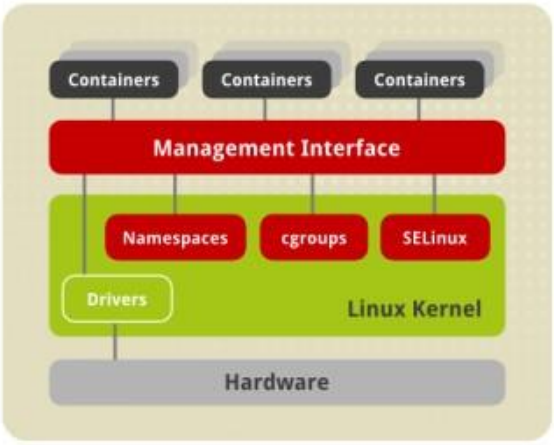

Рис.1.4 Архітектура контейнерів Linux в Red Hat Enterprise Linux

Запущені контейнери на ОС в основі Non-Linux технічно неможливо, як це вимагає доступ до ядра Linux. Невелика, легка віртуальна машина створюється на ОС на базі Nonlinux, як Mac або Windows. Функція віртуалізації апаратного забезпечення повинна бути ввімкненою на хост-машині, щоб підтримати спінінг віртуальної машини. Один раз Linux OS працює, контейнери створюються за допомогою ядра віртуальної машини.

1.4. Особливості ядра

CGroups або Control групи - це функції в ядрі Linux, які забезпечують вимірювання та обмеження ресурсів, як пам'ять, процесор, група процесів та пропускна спроможність диска.

Для управління ресурсами більша частина технології контейнерів використовує CGroups як основний механізм. Дану функцію можна контролювати, налаштовано, відхилено в доступі до конкретних ресурсів, і навіть переконфігуровані динамічно на робочу систему.

Простір імен Linux забезпечує кожен процес з власним поглядом системи, тобто обмежує те, що можна побачити процес. Простір імен Linux створює абстракцію ресурсів та робе його як окремий екземпляр всередині контейнера. Таким чином, різні контейнери можуть одночасно використовувати ті ж самі ресурси.

Security-Enhanced Linux або SELINUX - це реалізація багаторівневої безпеки (MLS), механізм керування доступом (MAC) та безпека декількох категорій (MCS) у ядрі Linux. Застосовуючи політику SELinux, вона забезпечує безпечне відокремлення контейнерів.

CHROOT або CHANGE ROOT використовується для зміни кореневого каталогу процесу та його до нового каталогу, що видно лише для даного процесу. Ідея використання CHROOT контейнерів, є ізольованим та ділеться файловою системою через кілька контейнерів.

1.5. Переваги контейнерів

Щільність

Щільність - це максимальна кількість контейнерів або віртуальна машина, яку хост машина може підтримувати без будь-яких необхідних для неї ресурсів. Контейнери легкі, ніж віртуальні машини, тобто у них є менша утилізація пам'яті. Таким чином, більше контейнерів можуть працювати на хост-машині порівняно з віртуальною машиною, як показано на рисунку 6.

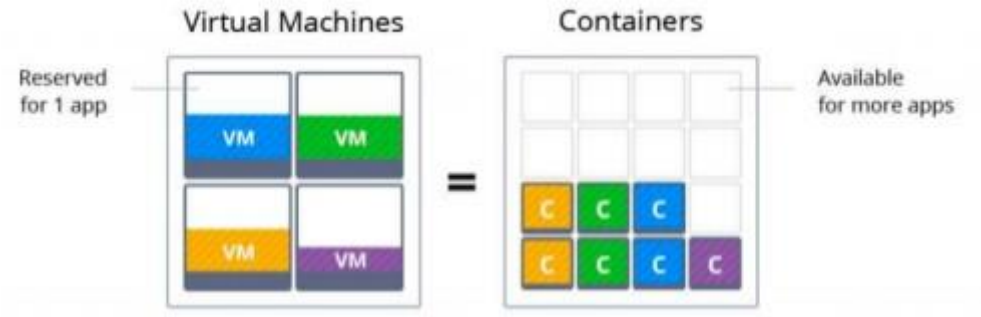

Рис.1.5 Щільність віртуальних машин та контейнерів

Утилізація ресурсів

Контейнери забезпечують ефективні утилізації ресурсів у порівнянні з віртуальною машиною, коли вони поділяють ресурси між ними. У випадку віртуальних машин, ОС завантажується, навіть якщо одна програма буде працювати всередині неї. Як видно на рисунку 7, необхідні ресурси з хост-машини зарезервовані для кожної віртуальної машини.

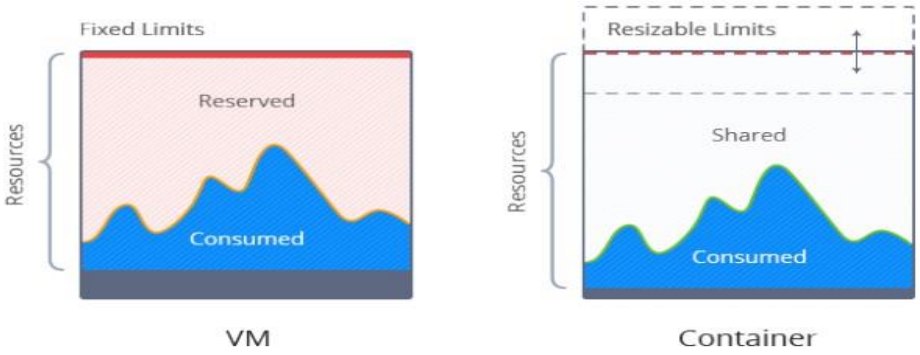

Рис.1.6 Утилізація ресурсів віртуальних машин та контейнерів

Масштабування

Зміна розміру межі ресурсу (наприклад, процесор або оперативної пам'яті) у контейнері може бути зроблений на льоту, без перезавантаження за допомогою функції ядра CGROUP. Однак, в віртуальних машинах, ми повинні спочатку зупинити їх, перенаправити, а потім перезапустити.

Згодом, як показано на рисунку 8, простота під час масштабування віртуальні машини.

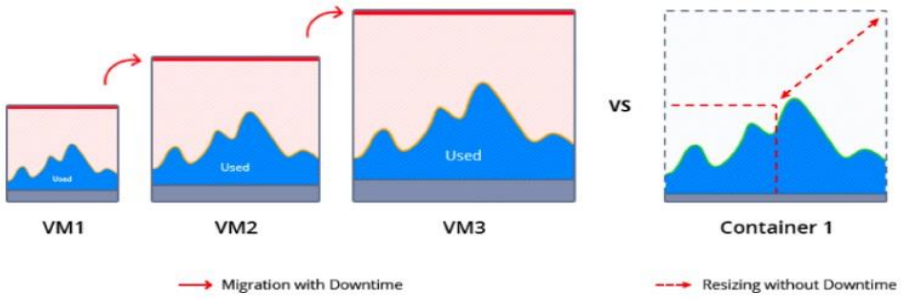

Рис. 1.7 Масштабування віртуальних машин та контейнерів

Швидке забезпечення

Контейнери поділяють ядро хоста і, таким чином, вимагається значно менше часу для початку. Вони діють як процес, що працює в ізольованому просторі. У разі віртуальної машини, повна операційна система завантажується і, таким чином, вимагає більше часу для обертання у порівнянні з контейнерами.

Портативність

Портативність рухає застосування з одного середовища до іншого. Контейнери надають середовище для запуску програми разом з необхідними файлами та конфігураціями. Таким чином, запустіть їх на будь-якій операційній системі або апаратному засобі впливають на застосування всередині нього.

Відкритий контейнер

З кожнем днем все більш стає технологій, що призвело до власної специфікації та методу запуску. Різні лідери в контейнерній промисловості, такі як CoreOS, Docker створики проект з відкритого упорядкування, який називається відкритою ініціативою контейнера (OCI) під Linux. OCI допомагає забезпечити відкриту промисловість стандартів навколо контейнерних форматів та виконання, тобто, він визначає технічні характеристики та створює зображення OCI (Image-Spec) та OCI Runtime (Runtime-Spec), щоб запустити OCI зображення.

Специфікація зображень

Специфікація зображень (Image-Spec) визначає спосіб створення контейнера на основі OCI зображення. Зображення OCI - це колекція зображень метаданих про вміст, конфігурації зображень (інформація, пов'язана з застосуванням та навколишнім середовищем) і шарувата файлова система, що містить файли програми та налаштування.

Специфікація виконання

Специфікація виконання (Runtime-Spec) визначає спосіб запуску зображення OCI, розпаковуючи зображення OCI у файлові файли файлової системи OCI. Зображення OCI містить усю інформацію, яка необхідна для запуску програми (наприклад, команду, аргументи, користувачі, змінні середовища). Будь-який контейнерний двигун, як Docker, RKT може запустити зображення OCI без додаткових аргументів.

• Docker Run My_AP: V1.0.0

• rkt run my_app, версія = v1.0.0

Специфікація розподілу

В даний час OCI також працює над створенням специфікації розподілу, щоб стандартизувати

Спосіб розповсюдження зображення. Ця нова специфікація підтримуватиме сумісність у контейнерній екосистемі.

1.6. Контейнерні технології

Зазвичай контейнери - це додаткові центри, які містять одну програму, тобто контейнер і відомі як контейнери для застосування. Альтернативно, є система контейнерів, які працюють по-різному. Вони поводяться як повна операційна система, але поділіться ядром керівника. Всі компоненти програми запускаються всередині, тобто єдина контейнер система. Залежно від вимоги, будь-який з них можна вибрати види контейнерів. Обидві технології контейнерів мають свої плюси і мінуси, які обговорюватимуться далі.

Контейнер застосування

Контейнер застосування, працює один процес всередині контейнера і, таким чином, створює окремий контейнер для кожного компонента програми. Тобто містить все, що вимагається кожним компонентом програми для його виконання. Тому він працює аналогічно в будь-якому середовищі, незалежно від його операційної системи або основне обладнання. Більш доцільно створювати проекти, які дотримуються архітектури Microservice. Таким чином, за допомогою контейнерів додатків це досить легко масштабні один або кілька компонентів застосування.

Docker - це контейнерна технологія з відкритим кодом, яка забезпечує платформу для будівництва, запуск та доставка додатків разом з їх залежностями всередині віртуального середовища з використанням контейнерів. Французька компанія під назвою Dotcloud створила власний

Docker. DotCloud забезпечив PaaS (платформу як послугу) хостинг для веб-додатків. Пізніше він був перейменований до Docker Inc. Спочатку, Docker використовував LXC в середені. Однак,з версією 0.9, Docker створив свій власний драйвер під назвою Libcontainer5, щоб отримати доступ до Linux Kernel. Пізніше, з появою OCI, вони покращилися до runC.

CoreOS - це відкрите джерело, легка операційна система, розроблена спеціально для підтримка контейнеризації. CoreOS та Docker мали угоду, щоб створити простий контейнерний двигун. Однак, Docker почав додавати нові функції, такі як кластеризація, мережа та ін, що робить його більш складною системою. З іншого боку, Coreos контейнерний двигун, RKT, зберігає простоту і забезпечує найбільш жорстку безпечну модель. RKT також слідує стандартам OCI, щоб створити зображення контейнерів OCI, відомі як зображення контейнера App (ACI) та запускає їх за допомогою Runtime OCI середовища.

Системний контейнер, діє більше як повна операційна система. Тобто може запускати повнофункціональні системи init як sysvinit, openrc та systemd, що дозволяє кілька процесів для запуску всередині одного контейнера Linux. Цей тип контейнера є кращим для багатьох спадкоємців або монолітних додатків, на основі архітектури, оскільки це дозволяє повторювати архітектуру та конфігурації програми. Системні контейнери можуть мати повільний час початку, оскільки вони важче, ніж у порівнянні з застосуванням контейнерів внаслідок додаткових послуг, необхідних для виконання декількох процесів.

LXC або контейнер Linux

Контейнер Linux або LXC - це перша повна реалізація контейнера Linux менеджера. LXC - це щось між CHROOT та віртуальною машиною. Головною метою LXC - створити середовище, яке максимально близьке до стандарту установки Linux без потреби апаратного емуляції та окремого ядра. LXC використовує профіль безпеки Apparmor для безпеки. Він також використовує

CGroups для обмеження доступу до контейнерів. Таким чином, контейнер бачить лише власну файлову систему та процес простір. В даний час LXC використовує такі особливості ядра:

• Простір імен ядра (наприклад, користувач, IPC, Mount, UTS, NET та PID)

• Профілі Selinux та Apparmor

• CHROOT

• Групи CGroups або Control

• Seccomp політика

• Можливості ядра

Основною перевагою LXC є її легка реалізація, допомагаючи йому виконати на своїх швидкостях і забезпечуючи кращу файлову систему та ізоляцію мережі.

Отже, в даному розділі було розглянуто технологію віртуалізації та її типи. Пояснено, як виник контейнер Linux, основні технології , різні типи та переваги Linux. А також розглянуто, чому стандарти створюються навколо Linux контейнерів.

РОЗДІЛ 2
АНАЛІЗ ВИСОКОПРОДУКТИВНИХ МЕТОДІВ ВІРТУАЛІЗАЦІЇ СЕРВЕРІВ З ХОСТАМИ КОНТЕЙНЕРІВ

2.1. Модернізація традиційних додатків Docker (MTA)

Ліфт і зсув - це підхід, за допомогою якого ми переміщуємо запущену заявку від одного середовища до іншого. Цей підхід спочатку був використаний для переміщення додатків від простих до віртуальних машин у хмарі. Тепер використовується аналогічна стратегія для уточнення застосувань. Програма, що працює на машині, може бути переміщена в системні контейнери або контейнери для застосування на основі їх архітектури. Якщо застосування можна розкласти на менші частини, вони переноситься на додаток контейнерів з використанням Docker, RKT та ін. Рисунок 9 показує, як кожну частину програми можна перемістити в різні контейнери для застосування.

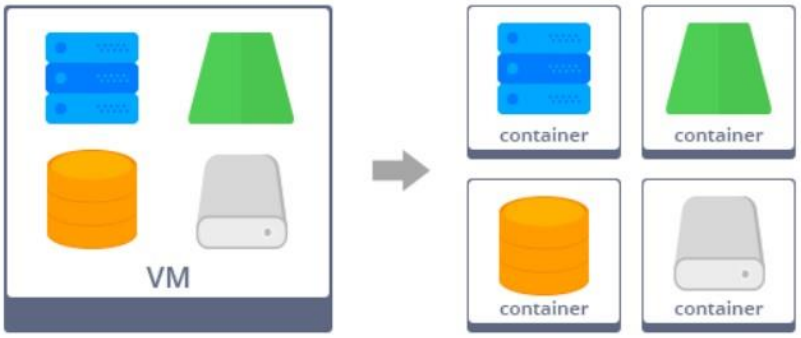

Рис. 2.1 Розкладка додатків на невеликі частини для переміщення їх у контейнери для застосування.

Якщо застосування щільно поєднується, бажано перемістити його в системний контейнер використання OpenVZ, LXC. Рисунок 10 показує, як цілий додаток переходить у системний контейнер.

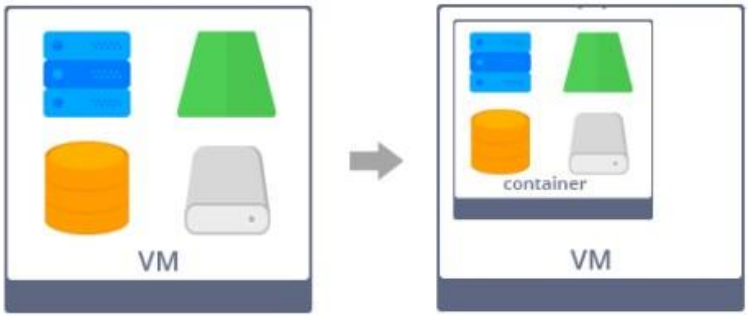

Рис. 2.2 Переміщення цілого додатку в системні контейнери.

Комерційні інструменти

Багато компаній ведуть шляхи у технологіях віртуалізації. Ці компанії також надають рішення, щоб мірувати фізичну машину до віртуальної машини або контейнери. Ці рішення зазвичай пропонуються як преміальні послуги разом з додатковими витратами. Розглянемо міграційні рішення, надані різними компаніями.

Модернізація традиційної програми Docker (MTA), дає інформацію про перетворення спадщини на архітектуру, керовану мікросервісом. Docker підтримує цю послугу всім своїм користувачам, які мають Enterprise Edition Docker. Рисунок показує крок за кроком процес встановлення додатків за допомогою процесу MTA Docker.

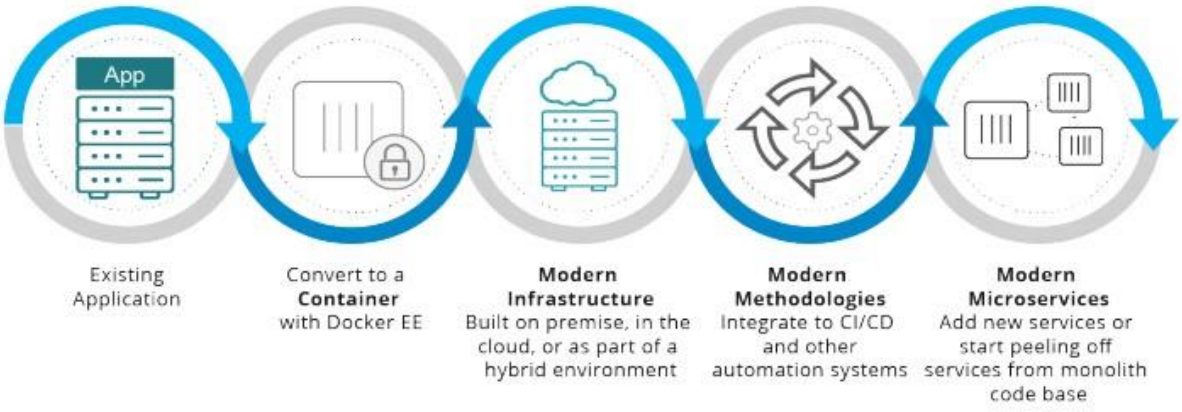

Рис. 2.3 Шлях перетворення існуючої програми до сучасного мікросервісу

Існуюча програма перетворюється на Docker File за допомогою інструмента "Open Source Image2 Docker". Після того, як Docker file готове, рішення про рефактор, або запровадження CI / CD автоматизацію або

розгортання зображення Docker до нової інфраструктури на основі індивідуальної вимоги до проекту.

2.1.2. App2Cloud

App2Cloud4 - використовують свої інструменти автоматизації та команди висококваліфікованих розробників, щоб перетворити традиційні програми, на програми, щоб відповідали внутрішньому Docker контейнеру. Клієнти можуть використовувати веб-платформу на вимогу, щоб завантажити додаток. Однак це нереально для компанії або особи, щоб поділитися своїми вихідниими кодами для міграції.

2.1.3. Інструменти з відкритим вихідним кодом

Docker набирає популярності та розвиток контейнеризації, а інструменти обертаються навколо нього. Однак існуюча робота в області контейнеризації є досить обмеженою. Ми пройдемо через роботу різних інструментів з відкритим вихідним кодом поряд з плюсами і мінусами. Інструменти мають неадекватну підтримку, неповну документацію та багато відкритих питань. Використання цих інструментів з відкритим вихідним кодом може включати величезну криву. Також, інструменти не дозволяють користувачам втручатися в постійний процес.

2.1.4. VM2 Docker

Віртуальні машини до набору контейнерів Docker, як показано на рисунку 12. Ця структура вимагає кожного VM для запуску агента (частину програмного забезпечення). Після того, як агент запускає на VM то він проходить інформацію про VM, таку як операційна система, запуск процесів, встановлених пакетів, відкритих портів до основної програми. Агент записується в систему, і він встановлює роз'єм TCP через який робить дзвінки віддаленої процедурою. Після чого ініціює процес перетворення та створює додатковий файл, використовуючи всю інформацію, зібрану про VM. Використовує Utility RSYNC Linux, щоб переконатися, що файлова система VM знаходиться в синхронізації з файловою системою Docker. Всі процеси,

що працюють всередині віртальної машини (ВМ), починаються всередині контейнера Docker. Таким чином, контейнер для Docker виступає, як контейнер системи і виконує декілька процесів всередині.

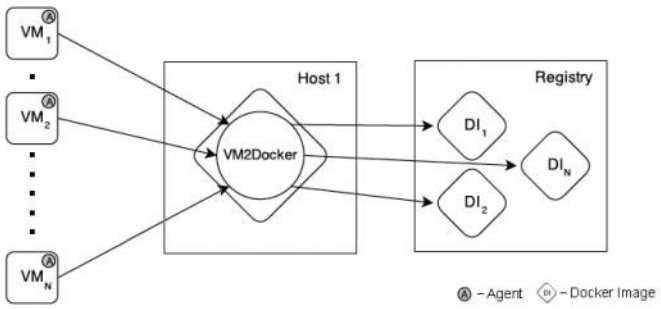

Рис. 2.4 Огляд структури VM2Docker

Плюси:

• Вся необхідна інформація щодо цільової VM збирається програмно

• Використовується такою ж базою

• Зображення для кожного контейнера.

• Приготування конверсії у невеликому контейнері розміром у порівнянні з VM.

• Він підтримує великі розподіли Linux.

Мінуси:

• Це не підтримує багатозначну структуру.

Немає підтримки мережевих контейнерів.

• Вона має проблеми безпеки, оскільки порт зберігається відкритим на цільовому VMS.

• Це не збирає інформацію, як користувачі, SSH-конфігурація та пароль.

• Цей підхід прив'язаний до Docker.

• Це не дає жодної документації для встановлення та тестування прототипу.

2.1.5 Image2 Docker

Image2 Docker - інструмент, заснований у грудні 2016 року під патрантажем Docker Inc. image2docker дозволяє користувачеві конвертувати вікна і Linux на основі віртуальної машини диска в Dockerfile. Підтримка Windows і на основі додатків Linux, існує дві різні версії image2docker. Приведемо приклад загальних кроків, які пов'язані з даним процесом на рисунку 13.

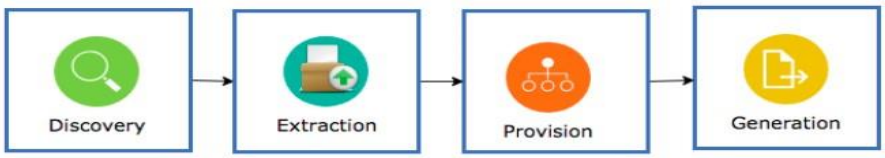

Рис. 2.5 Узагальнений підхід Image2Docker

Image2docker (Windows)

Image2Docker (Windows) - це модуль PowerShell, який порти існуючих додатків до Docker. В основному він підтримує IIS (Інтернет-інформаційні послуги) та ASP.NET на основі додатків. Він виявляє різні запущені програми, такі як asp.net Веб-додатки, екземпляри сервера Microsoft SQL, веб-сервер Apache та ін.. кожний додаток, який розглядається як реликт, а окремий код записується для виявлення.

Основних компоненти image2docker:

• Відкрийте функцію: виявляє бажані реликти з віртуального жорсткого диска.

• Створити функцію: генерує вміст DockerFile на основі виявлених 25реліктів.

Image2docker (Linux)

Image2docker (Linux) - це програма Golang, яка використовується для перетворення віртуального на основі Linux жорсткий диск у зображення docker.

Додаток ділиться на чотири частини:

• CLI і робочий процес архітектури це CRUX програми. Розуміє всі плагіни і контролює потік даних. Як тільки всі установи завершили завдання та актуально зібрані дані, архітектура створює остаточний DockerFile.

• Пакувальні плагіни Packager, плагін працює перед будь-яким іншим плагіном і переконується, що вміст віртуального диска доступний для інших плагінів через об'єм. Точний зміст віртуального диска доступний лише для плагіна Packer та у форматі Readonly. Він витягує весь матеріал, який вимагається іншим плагіном і надсилає цю інформацію до інших плагінів.

• Детективні плагіни, перевіряє вихідну систему та збирає будь-яку інформацію, яка вимагатиме зауваження, щоб створити зображення Docker. Приклад плагіна: індентифіккація операційної системи.

• Плагіни надання, після того, як детективний плагін виявляє необхідну інформацію, відповідний виїмка виконується. Далі сприяє остаточному DockerFile з відповідною інформацією. Наприклад, якщо детективний плагін ідентифікує Ubuntu ОС версія 16.10, відповідний виїмка буде сприяти dockerfile, як "з Ubuntu: 16.10".

Плюси:

• Інструмент має модульну архітектуру. Таким чином, додавання нового плагіна є легким.

• Інструмент працює безпосередньо на віртуальному диску. Це не вимагає VM.

• Він підтримує Windows, а також великі розподіли Linux.

Мінуси:

• Оскільки жоден загальний плагін працює для кожного типу програми, користувачі повинні кодити плагіни відповідно до їх потреб.

• Цей інструмент вибирає всі частини, встановлені всередині VM і додає їх до зображення Docker.

• Не має реальної підтримки.

Image2docker виступає за стандартами OCI для створення контейнерного зображення, відомого як зображення docker. Зображення Docker можуть бути створені вручну або можна завантажити з реєстру Docker. Docker Hub9 це реєстр docker, які використовується для зберігання та розподілу docker. Docker зображення, як вони оптимізовані для розміру зображення, забезпечують кращу документацію та дотримання практики.

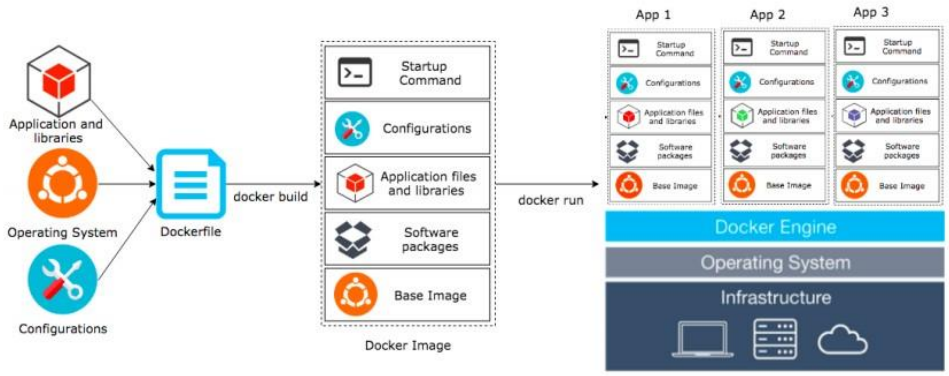

Рис. 2.6 Docker закінчується потоком.

На рисунку 2.6 показано, що для обробки контейнерів необхідні зображення. Ці зображення є шаблонами лише для читання, які можуть включати основне зображення ОС або попереднє застосування разом з усіма вимогами. На рисунку 14 ми показали приклад попередньої побудови програми. При побудові зображень, кожна команда виконується, утворює новий шар вершини попередньої. Можемо запустити ці команди вручну або використовувати DockerFile автоматично. Dockerfile - це простий текстовий документ, який містить декларативні вказівки, які використовуються для визначення того, як зміст зображення буде складатися разом. Докер-файл інтерпретується двигуном в перетворені ображення докера.

Розуміння Dockerfile

Інструкції в Dockerfiles допомогли відповісти на різні питання, такі як найпоширеніші питання якості, найпоширеніші основні зображення, що використовується, тип використання мови програмування тощо. Надамо пояснення деяких інструкції, що використовуються в реалізації прототипу.

1. FROM команда, яка вказує основне зображення, яке буде використовуватися.

2. RUN команда виконання, яка виконує вказану команду. Тут він оновлює apt кеш і встановлює сервер NGNix.

3. ENTRYPOINT команда для запису повідомляє двигуну Docker, щоб розпочати це зображення з вказаною командою / файл. Тут розпочався сервер NGNix.

4. EXPOSE команда підказка, якавказує на те, який порт контейнер буде слухати.

2.2. Хмарні обчиисления

З розвитком телекомунікаційних технологій мережі та розширенням інтернету можливо для централізації обчислювальних ресурсів назвати концепцію хмарними обчисленнями. Національний інститут стандартів та технологій (NIST) розробив документ, що визначає хмарні обчислення. Вони визначають хмарні обчислення як "модель для забезпечення повсюдного, зручного доступу до мережі та спільного пулу налаштовуваних обчислювальних ресурсів" і описує свої істотні характеристики. Основні характеристики, які NIST описує: здатність споживача надати комп'ютерні ресурси з постачальником послуг та можливість досягнення ресурсів через доступ до мережі, можливість використання ресурсів мульти-орендаря, масштабованість та моніторинг використання ресурсів.

Cloud Computing Can може бути доступними через кілька різних моделей обслуговування. Служба програмного забезпечення-AS-A-SERVICE полягає в тому, щоб доставити доступ до доступу користувачів, не надаючи їм розуміння базової платформи, на якій запущено програму. Інша модель послуг - це платформа-ASA-сервіс, який дозволяє споживачам розгортати свої послуги на централізованій платформі, але без розуміння основної інфраструктури, на якій працюють послуги. Точна остання сервісна модель - це інфраструктура-AS-A-A-A-A-Service, яка дозволяє споживачам створювати

власний Віртуальний центр обробки даних з контролем операційних систем та розгортання сервісу. Споживач не має контролю над основною хмарною інфраструктурою. Крім необхідності технологія мережі, що дозволяє отримати доступ до цих ресурсів, має бути увімкнення ізоляції та управління різними ресурсами, що працюють у хмарі.Одна така технологія, яка широко використовується, - це віртуалізація.

2.2.1. Віртуалізація

Вважається, що концепція віртуалізації походить з початку 1970-х років. Перше проходцями були бізнес-корпорації, типу IBM, які витрачали значні зусилля, намагаючись розробити ефективну технологію спільного використання часу для своїх основних рамок. TimeSharing дозволяє обчислюванню потужності мейнфрейму який ділитися на акції, які можуть бути розподілено до різні групи користувачів. Пропускаючи вперед до кінця 1990-х років, технологічна компанія VMware розробила першу продукцію віртуалізації, які могли б віртуалізувати архітектуру X86. З X86 віртуалізація обладнання та операційна система була розділена в дві абстракційні шари. Абстрактний шар дозволяє операційним системам та програмам, що працюють на фізичні машині стати апаратним агностиком і з тим, що приходить підвищена спритність і безперервність бізнесу. Послуги більше не повинні бути зняті для технічного обслуговування обладнання або резервні копії, оскільки додатки можуть бути легко перенесені на інші фізичні хости або скопійованими на них. З 2007 року VMware заявив, що він мав клієнтів з виробництвом серверів, які працювали протягом трьох років без простоїв.

2.2.2. Віртуалізація апаратного забезпечення

Віртуалізація апаратного забезпечення - це технологія, яка створює абстрактний шар між обладненням та операційною системою. Це робиться за допомогою програмного забезпечення, що називається гіпервізором. Гіпервізори приходять у різних формах, але є два первинні типи; Тип-1 гіпервізори та тип-2 гіпервізори. Гіпервізори типу-1 також називаються

гіпервізорами металів, оскільки вони працюють безпосередньо на апаратному забезпеченні. Приклади гіпервізорів типу-1 - Microsoft Hyperv, VMWare ESXI та Citrix Xenserver. Тип-2 гіпервізори запускаються на операційній системі хазяїна і, отже, поставляється з підвищеними накладними витратами для віртуальних машин. Приклади Тип-2 гіпервізорів - це Workstation WorkStation та Oracle VM Virtualbox.

Основною функціональністю віртуалізації апаратного забезпечення полягає в тому, щоб зробити кілька операційних систем та додатків, що виконуються паралельно за тим самим апаратним забезпеченням, створюючи віртуальні екземпляри обладнання для кожної віртуальної машини, і тим самим робить економію витрат за рахунок збільшення утилізації обладнання. Найдавніший продукт сервера, який зумів зробити це був VMware ESX. Він використовував спеціальне ядро, яке називається VMKernel, який був розроблений для запуску та керування робочими навантаженнями віртуальних машин. Сам VMKernel запускає монітор віртуальної машини, VMM, для кожної віртуальної машини в системі. VMM є відповідальний за реалізацію віртуального обладнання та виконання віртуальної машини. Для того, щоб мати кілька віртуальних машин бігаючи на тому ж хості, вони повинні бути ізольовані один від одного і фактично обладнений. Щоб пояснити, чому це важливо уявити собі віртуальну машину, що надсилає привілейовану інструкцію для того, щоб повернути себе. Для того, щоб ця інструкція вимикала віртуальну машину і не всю систему, виклик повинен бути інтерпретером у правильному шляху. Для цього використовується методика, що називається двійковим перекладом. Техніка привілейована інструкції з віртуальних машин та перекладає їх у те, що означає інструкція в контексті джерела, що є віртуальною машиною. Рисунок 2.7 , показує як повна віртуалізація з двійковою трансляцією роботи по відношенню до виконання інструкцій кільця архітектури X86.

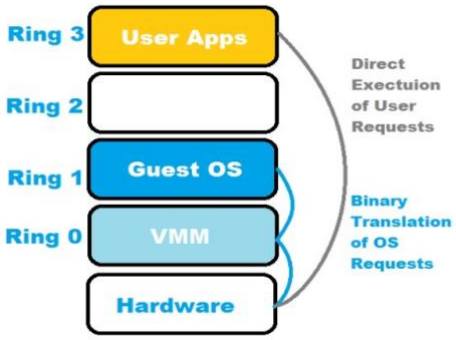

Рис. 2.7 Віртуалізація з двійковою трансляцією

Іншим раннім підходом до віртуалізації апаратного забезпечення була-паравіртуалізація. Використання паравіртуалізації вимагає модифікованої гостьової операційної системи з програмами керування віртуалізацією та драйверами пристроїв. Перевага цього полягає в тому, щоб видалити необхідність двійкового перекладу, зробивши гостьову операційну систему усвідомлюючи, що вона є віртуалізованою.

З цим додатком Операційна система замінює невірні привілейовані інструкції з гіпердзвінками. Гіпердзвінок є спеціальною інструкцією, яка надсилається до гіпервізора. Гіпервізор потім пересилає привілейовані інструкції до апаратного забезпечення. Передбачається, щоб зменшити накладні витрати віртуалізації, видаляючи необхідність бінарного перекладу, але VMWare претендує на Advantion Advantage, який змінюється залежно від робочого навантаження сервера та натискає на факт, що паравіртуалізації викликає більшу складність, коли справа доходить до управління і підтримки через необхідність змінити ядра гостьової операційної системи.

В останні десятиліття третій підхід до апаратної віртуалізації було використано, як віртуалізацію апаратного забезпечення. Вона вперше з'явилася в 2006 році, коли Корпорація Intel та передові мікро-пристрої випустили свої функції підтримки для апаратної віртуалізації, Intel-VT та Amd-v. Нові доповнення додавали нові функції виконання процесора, що означало, що VMM може працювати нижче кільця 0, в новому режимі, який називається root (іноді називається кільцем - 1). Привілейовані дзвінки

автоматично зафіксовані до гіпервізора і, таким чином, видаляючи необхідність двійкового перекладу, а також видаляє необхідність запускати модифіковані операційні системи, які в свою чергу використовували підхід до паравіртуалізації. Рисунок 2.8 описує, як апаратна допомога віртуалізації працює по відношенню до оновленої інструкції виконання архітектури кільця x86.

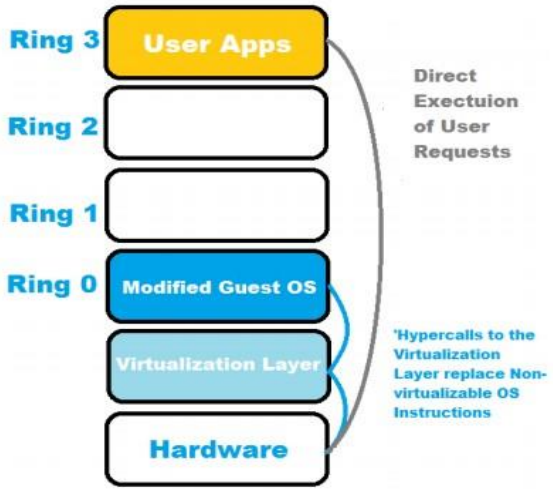

Рис. 2.8 Паравіртуалізація для гіпердзвінків

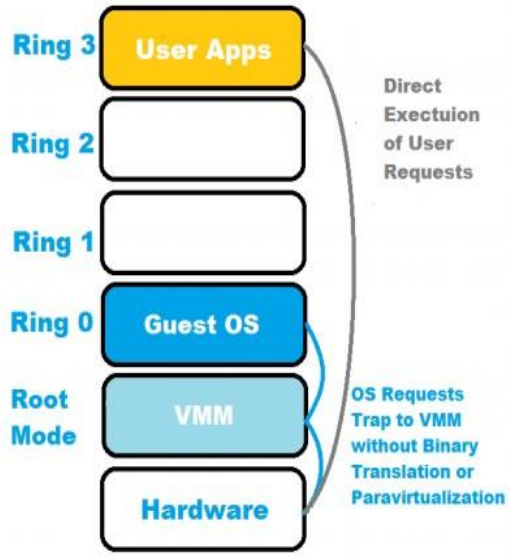

Рис. 2.9 Апаратна допомога віртуалізації

2.3. Контейнеризація

Контейнеризація - це віртуалізація рівня експлуатації, яка використовується для забезпечення ізоляції та управління ресурсами, в першу

чергу в середовищі Linux. Назва контейнеризації походить від способу судноплавних контейнерів, а в контексті додатків він відноситься до активного способу упаковки у ізольованому середовищі виконання. Ізоляція створюється трьома основними компонентами; CHROOT, CGROPS та ядром простору імен. CHROOT - це команда в Linux, яка дозволяє процесу змінювати кореневий каталог для створення контейнерів конкретних файлових систем. Cgroups - це підсистема ядра, за допомогою якої процеси можуть бути призначені квоти ресурсів. Простір імен ядра дозволяють кожному контейнеру отримувати власну конфігурацію мережі та міжтехнологічну комунікацію, IPC, Простір імен. Рисунок 2.10 показує огляд контейнеризації архітектури.

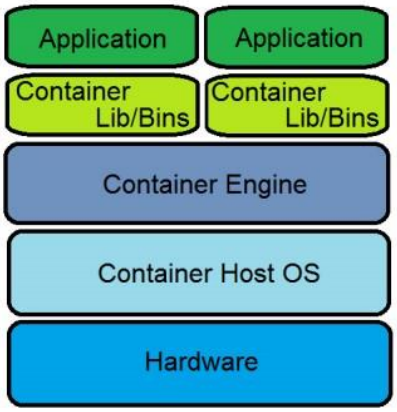

Рис. 2.10 Контейнеризація архітектури

Приклади контейнерних хостів ОС, як видно на рисунку, є Coreos, що є специфічно розроблені для цього, але більшість великих розподілів Linux мають підтримку запуску контейнерних двигунів. Контейнерний двигун, наприклад, Docker або RKT, є програмним забезпеченням який запускає контейнери.

Контейнеризація дозволяє створювати декілька прикладів, які виділяються один з одного, і це ці сегментовані екземпляри, які називаються контейнерами. В Linux, де декілька розподілів Linux використовує те ж саме ядро, що дозволяє кожному контейнеру мати окремий розподіл з контейнера-хоста. Програми, що працюють всередині можуть мати свої власні бібліотеки,

а контейнери можуть бути адаптовані, щоб відповідати цьому конкретному застосуванню. Частини операційної системи господарів контейнера, які поділяються контейнерами чи є тільки для контейнерів, коли вони мають свою частку, в якій вони можуть писати. Обмін ресурсами ядра робить контейнери набагато легшими, ніж його віртуальна машина. Всередині контейнерів достатньо, щоб запустити кожну програму, зберігаючи його розмір до мінімуму, управляющі зберігаються всередині спільної ОС як частину ядра. Тому контейнери дозволяють декілька додатків для запуску, ізольованих від кожного чи інакше одну спільну ОС. Для запуску декількох ізольованих додатків на тому ж хості в традиційній віртуалізації сервера одна віртуальна машина (VM) повинна бути створена для кожного.

2.4. VMware Vsphere

VMware Vsphere - це фірмова платформа віртуалізації сервера з VMware. Продукція ядра - це гіпервізор esxi. Vsphere ESXI - гіпервізор типу-1, який означає, що він встановлюється безпосередньо на апаратне забезпечення та працює як мініатюрна ОС. Він використовує VMKernel у своєму ядрі з інсталяційним відбитком 150 Мб, що означає, що він має мінімальний вплив на наявні ресурси зберігання та його невеликий розмір також зменшує ймовірність порушення через мінімальну поверхню атаки. Інші компоненти Vsphere є сервер VCENTER, клієнт VSPHERE та Vspere Web клієнт. Вони використовуються для управління одним або декількома хостами. Сам ESXI має тільки зрізаний вниз термінальний інтерфейс для лише основної конфігурації, наприклад, встановлення імені хосту та створення конфігурації мережі. Під час керування єдиним ходом адміністратор журналів у віддаленому використанні клієнту Vsphere, який працює на локальному комп'ютері. Коли власники об'єднуються у кластер, цей кластер управляється сервером VCENTER, який контролює всю індивідуальність власників. Сервер VCENTER може бути запущений зовні або як частина кластера. При

адміністративному кластері адміністратори підключаються до сервера VCENTER, використовуючи Vsphere Клієнт або веб-клієнт Vsphere, пізніше, що пропонує додаткові функції та інструменти керування.

VMware Vsphere має високу функцію доступності, яка забезпечує відмову від захисту від віртуальні машини. Функція може відслідковувати обидва ласника віртуалізації, так і віртуальні машини якщо виникає апаратна помилка або відсутність операційної системи гостей віртуальна машина автоматично починається в іншому функціональному хості. Відмова може бути зроблена досить швидко і залежно від операційної системи, яка повинна бути завантажена. Vsphere також пропонує особливість, що називається толерантністю від несправності, яка створює живий примірник віртуальної машини, яка працює на різних віртуалізаціях. Результат полягає в тому, що час простою винищувальної толерантності до віртуальної машини обмежується тим, що потрібно для системи, щоб помітити невдачу та активувати неохідну так би мовити інстанцію.

2.5. Опис існуючих систем

Photon OS - мінімальний головний контейнер Linux у розробці VMware. Будучи операційною системою контейнера, означає, що вона побудована для певної мети забезпечення загального ядра для контейнерів, що працюють на хості. Для запуску контейнерів фотон ОС має підтримку розгортання контейнера з Docker, RKT та Pivotal Garden і тоді як дуже мінімально, що він поставляється з менеджером, сумісним з YUM для циклу управління .

Photon OS є частиною іншого проекту VMware під назвою "Проект Бонневіль" його мета зробити взаємозв'язок між віртуальними машинами та застосуванням контейнерів додатковим, а не конкурентоспроможним. Проект є розширенням VMware Vsphere і буде контейнером для докерів, які будуть працювати як віртуальні машини, і, таким чином, отримати переваги ізоляції апаратної віртуалізації. Він працює, вивантажуючи ядро з бігу інстанція фотонів для створення нового віртуального контейнера-хазяїна для кожного

нового контейнера, який є розгорнутим . Вивантажуючи тільки ядро та кілька допоміжних ресурсів, необхідних для Запуску контейнерів набагато швидше розгортатися, ніж звичайні віртуальні машини.

Coreos - мінімальна операційна система, яка призначена для керування кластерами, щоб прийняти Linux контейнери. У ядрі Coreos є докер, тобто контейнерний двигун, який використовується для запуску контейнерів. Coreos не поставляється з менеджером пакетів для управління циклом, замість цього, Адміністратори кластера Coreos повинні запускати інструменти в контейнерах . Coreos використовує флот, тобто програмне забезпечення, яке робить адміністратора, здатним лікувати велику групу Coreos машини як єдину систему зі спільною системою init-siton. Використання флоту, кластер також здатний для підтримки високої доступності, оскільки, якщо хост кластера не вдається, то контейнери, що працюють на цьому хості автоматично запускаються на іншому головному господарському кластері .

Centos - це дистрибуція Linux, яка є похідною, від керованою Linux (RHEL) операційною системою. На відміну від RHEL, Centos повністю вільний та його мета - забезпечити платформу для відкритих джерел. Робоча система розробляється з 2004 року, і мета полягає в тому, щоб бути функціонально сумісною з RHEL .

Docker - це програма з відкритим кодом, яка дозволяє розгортати програми всередині контейнерів для програмного забезпечення. Всі залежності додатків включені до контейнера, що-небудь з коду та часу роботи з системними інструментами та системними бібліотеками, все, що є необхідно для запуску програми. Докер робить використання зображень для запуску нових контейнерів і вони є шаблонами, з яких можна створити багато контейнерів, потім кожен контейнер екземпляр конкретного зображення. Зображення створюються з шаруватих файлових систем, які вмикають спільне використання загальних файлів, це, у свою чергу, може допомогти зменшити сховище диска та швидкість завантаження зображень. Один з можливих сценаріїв для використання Докер - оптимізація інфраструктури. Докер

створює легкі контейнери, які всі поділіться тим же ядром, тому немає потреби в додаткових гостьових операційних системах для ізоляції програми. Контейнери створюють менш накладні витрати в порівнянні з віртуальними машинами, зменшуючи час обертання додатків і займати менше зберігання .

Apache HTTP-сервер - це веб-сервер, який станом на листопад 2015 року розміщує 37% всіх веб-сайтів , що робить його найпопулярнішим веб-сервером у світі. Apache мав свій початковий випуск у 1995 році і - це безкоштовне та відкрите програмне забезпечення; частина HTTP-сервера Apache проект, який є спільнотою розробника для подальшої розробки та підтримки Apache, все контролюється Фондом програмного забезпечення Apache . Apache можна використовувати на Linux, OS X і вікна серед інших.

Apache Jmeter - це програма Java, яка використовується для тестування веб-додатків. Jmeter працює тільки на рівні протоколу та не робить зображення, як звичайний браузер.

MySQL - це відкрита база даних, розроблена компанією MySQL AB, головний офіс у Купертіно, штат Каліфорнія, Сполучені Штати та Уппсала, Швеція. У 2008 році компанія була викуплена у Sun Microsystems, Inc і з тих пір спільне підприємство. У 2008 році база даних тобто програмне забезпечення було завантажено 100 мільйонів разів з 50 000 завантажень щодня. Mysql - це транзакційна база даних, що означає, що дані обмінюються в операціях. Якщо транзакція не завершується у повному обсязі, обмін даними повертається назад, тим самим підтримуючи узгодження транзакції у випадку потенційної системи .

В даному розділі розглянуто наявні підходи контейнера, використовуючи комерційні та відкриті інструменти. Також додатково оцінено, як працює кожен підхід, який узагальнює свої обмеження. Дані обмеження стали базовими для нашої роботи. Також у даній главі надало опис технологій, що використовуються в цій роботі, для того, щоб дати основне розуміння технологій та концепції.

РОЗДІЛ 3
ЗАГАЛЬНИЙ МЕТОД КОНТЕЙНЕРИЗАЦІЇ АВТОМАТИЗАВАНОЇ СТРУКТРИ

3.1. Збір інформації

По-перше, ми повинні зібрати всю інформацію про цільову машину, що працює. Необхідна інформація: операційна система, його розподіл та версія, встановлені пакети, запущені процеси, конфігурації SSH, користувачі та змінні середовища. Також повинні збирати інформацію про платформу хмари, якщо цільова машина розміщена на ньому. Аналогічним чином, якщо цільова машина працює на місцеві платформі Cloud, ми збираємо інформацію про гіпервізор та використовувані інструменти щоб створити його. Потім ми витягуємо значущі дані зі зібраної інформації і передаємо його на наступний крок.

3.1.1. Створення тимчасової машини

Використання інформації, зібраної з попереднього кроку, ми створюємо тимчасовий клон цільової машини з тією ж операційною системою. Таким чином, клонована машина буде мати таку ж версію операційної системи. Слово "таку ж" використовується для опису оригінальної версії програмного забезпечення без будь-яких налаштувань. Далі використовуємо цю клоновану машину як базову лінію для порівняння з цільовою машиною, щоб дізнатися чи всі налаштування виконуються на цільовій машині.

1. Налаштування може включати встановлення пакетів, створення нових користувачів, запущених процесів, відкритих з'єднань TCP.

2. Рис. 3.1, показує додаткові програми, що працюють на цільовій машині у порівнянні з клонованою машиною. Таким чином, будь-яке налаштування, яке не є на клоні буде додано до зображення контейнера. Ми видаляємо клон машину після збираємо всю необхідну інформацію.

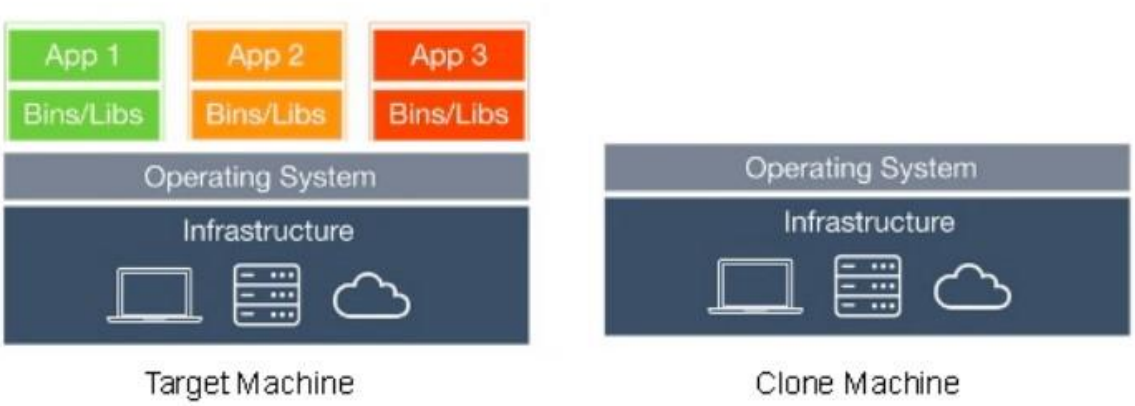

Рис. 3.1 Клонована машина має ту ж ОС, як і на цільові машині але з будь якими налаштуваннями

3.1.2. Порівняння машин

Після того, як ми зібрали всі деталі з цільової (T) та клону (C) машин, ми приймаємо відмінну різницю (T-C) на них. На рисунку 19 показано набір різницю встановлених пакетів на машині цільові та її клоні. Точна виділена частина на рисунку показує додаткові упаковки, які не присутні в клоновані машині і, таким чином, слід додати до зображення контейнера. Аналогічним чином, ми отримуємо іншу інформацію, знайшовши різницю на кожному типі інформації, зібраної як з машин. Вся ця інформація стає критичною частиною контейнера.

Запишемо інший тип похідної інформації:

1. Запуск процесів: кожен процес стане процесом запуску для кожного контейнеру;

2. Відкрити TCP-порти: щоб дозволити підключення та назад з зовнішнім станом, ми прив'язуємо порт контейнера до порту хазяїна;

3. Користувачі та групи: тобто багато додатків починаються з використанням конкретного користувача або групи. Таким чином, ці користувачі та група повинні бути присутніми в контейнері та підтримувати послідовність;

4. Змінні середовища: тобто багато додатків залежать від змінних середовища за їх виконання. Таким чином, похідні змінні середовища додаються до контейнерного зображення;

5. SSH CONFIG: Багато додатків встановлює SSH-зв'язок з різними машинами під час виконання. Щоб дозволити таку ж поведінку, навіть якщо додаток є контейнером, ми копіюємо SSH користувача у контейнер зображення.

3.1.3. Вирішення залежностей пакету

Як тільки з’являється список унікальних пакетів, які будуть встановлені в контейнері, потрібно вирішити залежності між ними. Багато пакетів ніколи не встановлюються безпосередньо. Вони встановлюються, як частина залежностей для іншого пакету. А отже, ми створюємо графік залежності для кожного пакета, як показано на рисунку 3.2.

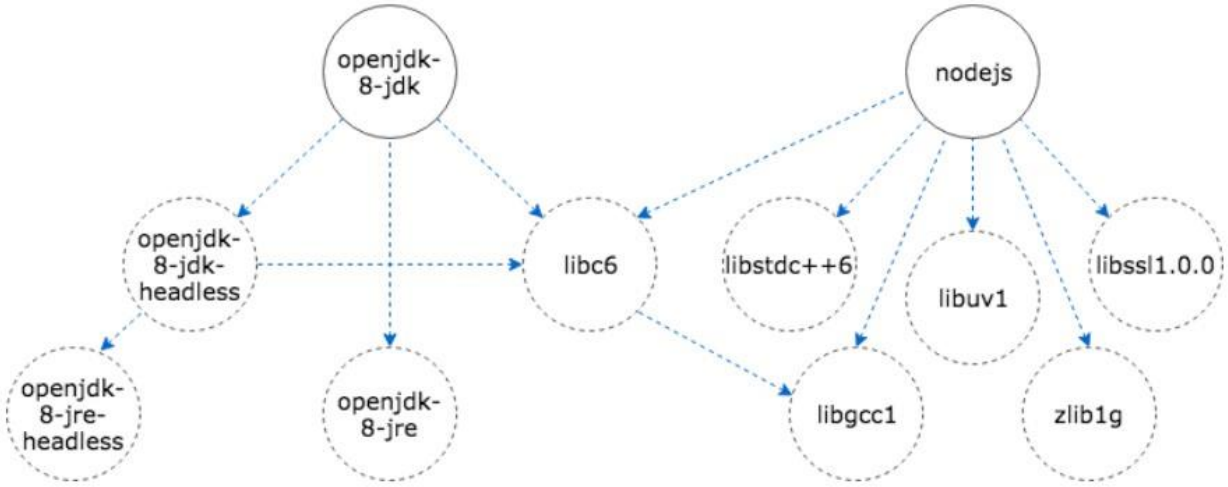

Рис. 3.2 Пакет OpenJDK-8-JDK пакет має три залежності, аналогічним чином і пакет Nodejs.

Пакет має шість залежностей. Як тільки створили графік для кожного пакета, вибираємо лише кореневий вузол (пакет) для установки. Внутрішньо, операційна система знайде залежні упаковки для кореневого вузла та встановить їх. Оскільки після вирішення залежностей буде менше пакетів, буде набагато легше асоціювати кожен пакет з запущеним процесом.

3.1.4. Управління пакетами

Після зменшення кількості встановлених пакетів, ми створюємо співвідношення ProcessActage, тобто відображення між запущеним процесом програми та встановленими пакетами.

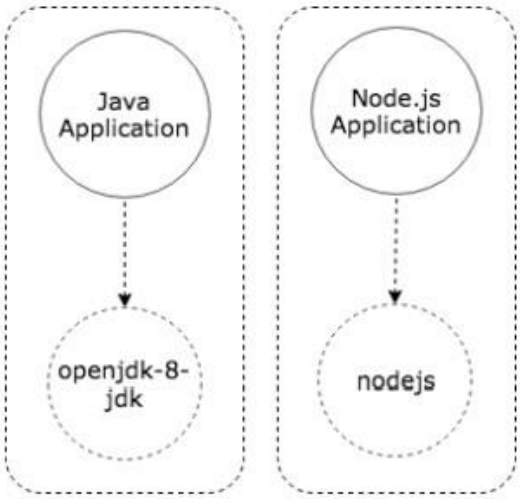

Рис. 3.3 Процес пакетного зв’язку між Java Application та пакетом OpenJDK-8-JDK і Node.js Application та nodejs пакетом

Пакет OpenJDK-8-JDK повинен бути встановлений для запуску Java налаштування. Аналогічним чином, щоб запустити програму Node.js, пакет Nodejs повинен бути встановленим. Якщо хочемо перемістити ці програми в контейнери, то повинні встановити пов'язані з ними пакети у відповідних контейнерах.

3.1.5. Файли та конфігурація додатків

Побудуємо відношення процесу-пакету, ми знаємо, що пов'язані пакети необхідні для запуску програми. Однак деякі програми можуть також вимагати файли і конфігурацію для їх виконання. Таким чином, має бути загальне рішення для пошуку асоційованих файлів конфігурації та папок для кожного процесу програми.

Головним рішенням, є:

Контроль кожного файлу і папки, яка доступна кожним процесом під час виконання. Ця інформація допомагає нам визначити необхідні файли та конфігурацію для кожного процесу застосування. Однак, на цільовій машині, оскільки програма є вже запущеною, ми не зможемо знайти доступ, який був доступним під час його запуску.

Щоб визначити весь файл та конфігурацію, доступ до процесу під час його виконання, ми використовуємо клона, щоб запустити ті ж процеси та контролювати їх. Щоб досягти цього, ми намагаємося зробити клон, як це можливо, максимально схожим на необхідну нашу ціль.

Машина, виконуючи наступні кроки:

1. Переносимо всі додаткові пакети.
2. Створюємо ті ж користувачі та групи.
3. Встановлюємо змінні середовища.
4. Копіюємо всі додаткові файли та папки з цільової машини.

На клоновані машині, подібної до цільової машини, починаємо процеси та контролюємо файли, які вони отримують доступ під час виконання. Таким чином, у нас є список пов'язаних файлів та конфігурацій для кожного процесу.

3.1.6. Створення контейнерного зображення

Маємо всю інформацію, необхідну для створення контейнерного зображення, можемо або створіть системний контейнер або контейнер застосування. Якщо ми можемо розкласти застосування без будь-яких змін у вихідному коді, створюючи додаток контейнер то буде ідеальним варіантом. Однак, якщо застосування щільно з'єднане і розкладання вимагає значного рефактора, створюючи системний контейнер ідеальним рішенням для цього сценарію, це рішення, яке приймається користувачем, який виконує контейнеризацію.

3.2. Автоматизація Framework

Запропонований метод допоможе користувачеві контейнерувати програми всередині пустого заліза або віртуальної машини. Однак цей підхід може стати дуже тяжким та може вимагати значних зусиль тому що розмір програми збільшується. Таким чином, ми вводимо автоматизовану структура, яка використовує наш запропонований метод до виконання контейнеризації застосування. Framework буде працювати автоматично та відображати результати до користувача. Користувач вирішує, що інформація повинна бути передана на наступний крок.

Рисунок 3.4 показує вигляд високо продуктивного рішення. Номер являє собою кроки нашої структури у процесі контейнеризації.

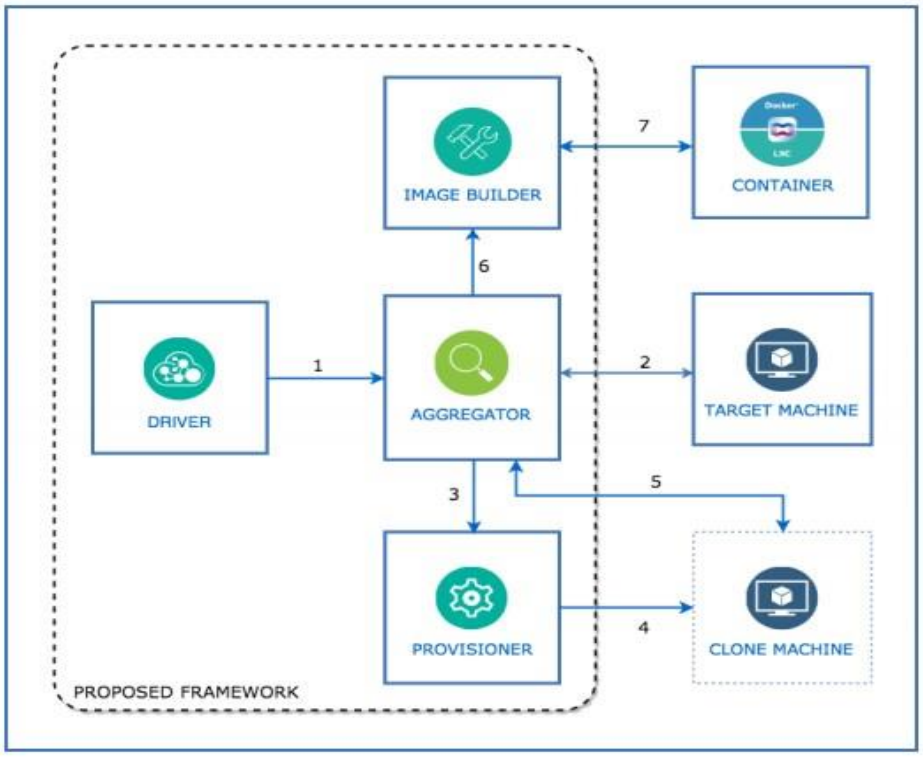

Рис. 3.4 Вид високого рівня компонентів структури

Запропонована структура ділиться на чотири компоненти. Кожен компонент повинен дотримуйтесь вимог. Кожен компонент пояснюється більш детально у наступних розділах.

3.2.1. Драйвер

Компонент драйвера ініціює процес контейнера, отримуючи повноваження користувача та необхідні для встановлення з'єднання з цільовою машиною. Користувацькі повноваження (наприклад, ім'я користувача, пароль або SSH) поділяються з наступним компонентом для подальшої обробки.

3.2.2. Агрегатор

Агрегатор є невід'ємним компонентом нашої основи, оскільки він з'єднується з цільовою машиною для зауваження та зображень. Він спочатку підключається до цільової машини, щоб зібрати всю необхідну інформацію та витягувати значущі дані з нього.

Витягнуті дані поділяються з іншими компонентами, оскільки вони не мають доступу до цільової машини. Агрегатор пересилає інформацію, пов'язану з ОС, щоб створити клон цільової машини. Після того, як клон вже є і працює, то агрегатом збирає таку ж інформацію, що і для цільової машини. Після чого агрегатор приймає зміну відмінності на кожному типі інформації, як пояснюється в розділі 3.1.2. Ця оброблена та обчислена інформація передана до будівельника зображень та створює зображення контейнера.

3.2.3. Пошук

Ресурс створює клон цільової машини за допомогою інформації, що надаються агрегатом. Для генерації клону, як це можливо, до цільової машини, залишок повинен відповідати наступним правилам.

• Для додатків, що працюють на пусті системі: послуги створює клон цільової машини за допомогою будь-якого наявного гіпервізора.

• Для додатків, що працюють на віртуальній машині: видання ідентифікує якщо цільова машина розміщена на хмарі або на локальній платформі. Точний Supplier, яка використовує таку ж платформу для створення клону цільової машини так що клон подібний до цільової машини.

3.2.4. Конструктор ображень

Конструктор зображень (Image Builder) використовує інформацію, отриману від агрегата, щоб створити контейнерне зображення. Це дозволяє користувачеві створювати систему або контейнер застосування. Він також підтримує створення незалежного постачальника оскільки вся інформація, необхідна для створення зображення контейнера, є доступною. Образ Image Builder, далі конструктора також випливає з кращих практик, зрештою, зображення, створене за допомогою конструктора зображення, використовується для керування контейнером Linux.

У даному розділі описано запропонований метод контейнеризації автоматизованої структури, який допоможе в тому, що структура дозволить користувачам переміщати програми, які працюють всередині заліза або віртуальні машині в контейнери Linux.

Це дозволяє забезпечувати інтуїтивний спосіб визначення запущеної заявки разом із його залежностями, необхідними для контейнеризації.

РОЗДІЛ 4
ЕМПІРИЧНИЙ МЕТОД ЗБІРУ ДАНИХ ДЛЯ ВИМІРУ ПРОДУКТИВНОСТІ У ВІРТУАЛІЗОВАНИХ КОНТЕЙНЕРАХ ХОСТІВ

4.1. Емпіричний метод

Новизною є те, що використовується індуктивний підхід, використовуючи емпіричний метод збору кількісних даних. Для того, щоб виміряти продуктивність додатків всередині контейнерів буде проведено ряд експериментів. Дані, зібрані з них стануть часом для завершення запитів на обслуговування та номером завершеної служби запитів.

4.2. Опис методу

Щоб описати метод, було проведено три експерименти: один тест HTTP-Get, другий SQL тест та третій тест SQL-insert. Тест HTTP-Get вимірює накладні витрати на віртуалізацію ЦП. Веб-сервер поклав потрібну сторінку у його кеш-пам'яті, опрацював у наданні сторінки якомога більше паралельних користувачів. Веб-сервер Apache, новий процес був створений для кожного нового користувача. Сторінка була 10 000 байт велика і мережа Gigabit Ethernet може підтримувати загальну пропускну спроможність 1 000000 000/8 = 125 Мб в секунду, що означає, що мережа може підтримувати близько 12 500 операції в секунду не беручи до уваги накладні витрати, введені Ethernet, TCP та http. Накладні витрати, як правило, становить близько 2-9% відповідно до пунктів форуму на стекурологічному потоці, які ще залишають мережу, що підтримує понад 10 000 транзакцій. Якщо результати показують менше транзакцій за секунду, мережа не перевантажена. На максимуму загальної кількості 100 користувачів працювала на веб-сайті, це означало, що якщо час відповіді веб-сервера полягав у прикладі 100 мілісекунд, не більше 1000 транзакцій може перерости мережу на будь-яку дану секунду. Якщо виникло вузьке місце, то це було швидше за все, тому, що процесор не мав обчислювального потенціалу, щоб впоратися з багатьма запитами.

Експерименти SQL-Select вимірювали затримку, яку невеликий SQL-вибирав запит та міг бути виконаний між двома контейнерами для застосування, що працюють на тому ж контейнері хазяїна. Коли сервер бази даних отримав запит, він також повинен був читати з диску. Відмінності між віртуалізованими та не віртуалізованими установками мали намір показати накладні витрати считування операцій і, якщо існує різниця в між технічній латентності зв'язку. Третій тест, тест SQL-вставки, вимірював накладні витрати операції. Мета цих експериментів в тому, щоб виміряти продуктивність застосування контейнерів, які працюють всередині контейнерів, розгорнутої Docer, а також порівняння не-віртуалізованого контейнера хостів до віртуалізованих контейнерів. Відмінності в контейнеризованому застосуванні між самими операційними системами контейнера.

4.2.1. Топологія

Розділ поділяється на: топологію для налаштування та тест-аналіз фізичного обладнання, що підтверджує їх рівну продуктивність вводу / виводу, закінчуючись наданням детального опису окремого експерименту.

Середовище віртуальної лабораторії складалася з шасі Bladecenter IBM, що містить чотири сервери та два вимикачі Cisco Ethernet. Експеримент був виконаний на п'яти різних установках. Кожен з серверів Blade мав одну або три налаштування.

Пять налаштувань (як видно на рис.4.1)

1. CentOS, що працюють на гіпервізорі ESXI
2. Coreos, що працює на гіпервізорі ESXI
3. Photon OS, що працює на гіпервізорі ESXI
4. CentOS працює безпосередньо на апаратному забезпеченні
5. Coreos працює безпосередньо на апаратному забезпеченні

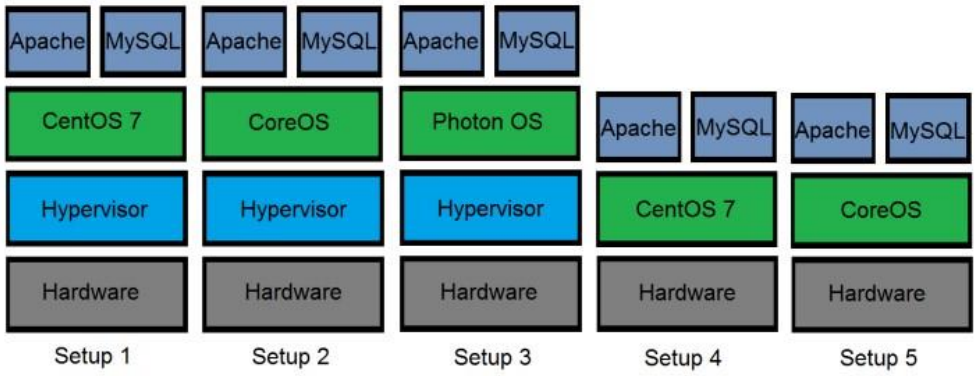

Рис.4.1 Показано п'ять різних налаштувань апаратного забезпечення

З чотирьох серверів, що розміщується 1, 2 та 3, два інші, розміщені у налаштуванні 4 та 5, і останній виступав як тестування клієнта, який перевірив всі інші налаштування за допомогою Apache Jmeter. Кожен з серверів Blade був підключений до двох Cisco Ethernet-вимикачів за допомогою двох мережевих інтерфейсів, один для кожного перемикач. Для відокремлення трафіку керування від тестового трафіку в мережі один інтерфейс був спеціальним інтерфейсом адміністрування, який використовується для обробки керування хостами, віртуальних машин та контейнерами для програмного забезпечення. Інший інтерфейс був використаний для обробки трафіку, створеного Apache Jmeter. Інтернет також був доступний за допомогою інтерфейсу керування, щоб тягнути до низу Docker контейнери з абоненських сховищ.

Кожна установка складалася з двох додатків, що працюють всередині власного програмного забезпечення контейнера; веб-сервер Apache і база даних MySQL.

4.2.2. Обладнання для апаратного забезпечення

Три тестові сервери, які поділені на однакові технічні характеристики, але, щоб мати можливість зробити будь-які дійсні висновки у порівнянні між налаштуваннями на різних фізичних серверах мали бути рівним у виконанні, що стосується наших орієнтирів. Тому у цьому випадку зчитувати та писати

продуктивність дисків було важливим, оскільки використовується апаратне забезпечення, так як за 10 років, як відомо механічний диск зноситься та погіршується з часом. Під час установки Centos був встановлений на всіх трьох серверах, щоб забезпечити невідомі операційні змінні, що впливають на продуктивність серверів. Бенчмаркінг Linux тобто інструмент FIO був використаний для виконання тестів, і метод був заснований на тому, що вважається гарною практикою для моделювання веб-сервера та робочих навантажень бази даних на I / O відповідно до контрольного диску I / O на бінарній смузі веб-сайту. Як правило, краще виміряти продуктивність підсистеми диска в блоці IOPS (операції вводу / виводу в секунду) у трьох різних тестах; випадкові зчитування / запису, випадкові зчитування та випадкові записи, всі з блоком розміром 4 кілобайт. Fio також дозволено для запуску декількох потоків, які було зроблено для моделювання декількох відвідувань веб-сайту, де кілька користувачів можуть одночасно прочитати та записати на диск. Результати I / O тест показав, що всі три сервери запропонували аналогічну продуктивність, як видно на рисунку 4.2, де Blade01-03 являє собою кожен з трьох телевізійних серверів IBM. Тест також випробує життєдіяльність дисків, і результати представлені на рисунку 4.2. Хоча Blade01 має менше половини життєдіяльності blade02 і blade03, найдовша затримка blade03 - це 0.50 мілісекунд, і ми вважаємо, що недостатньо, щоб зробити помітний вплив на експерименти продукту.

апаратне забезпечення орієнтири

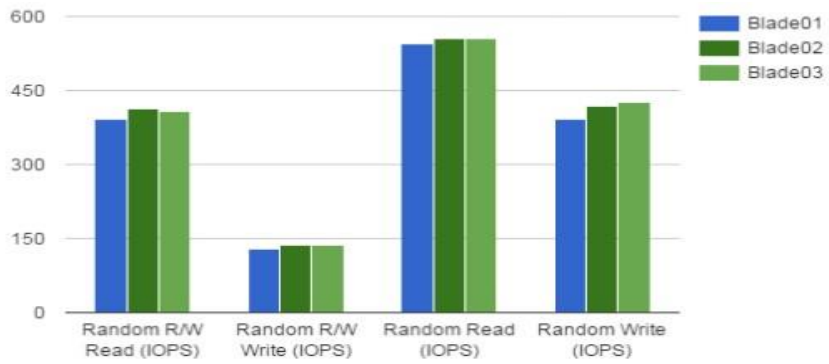

Рис. 4.2 Апаратне забезпечення зчитування та запису IOPS

4.2.3. Запити HTTP

Головний сервер не розміщує будь-яку установку, а вміcто цього виступає як генератор зовнішнього навантаження, що відправив трафік до кожного з п'яти тестових налаштувань. Випробування було зроблено, дозволяючи Apache імітувати зростаючу кількість активних користувачів, починаючи з параллельного очікування апаратного забезпечення прихованого стану

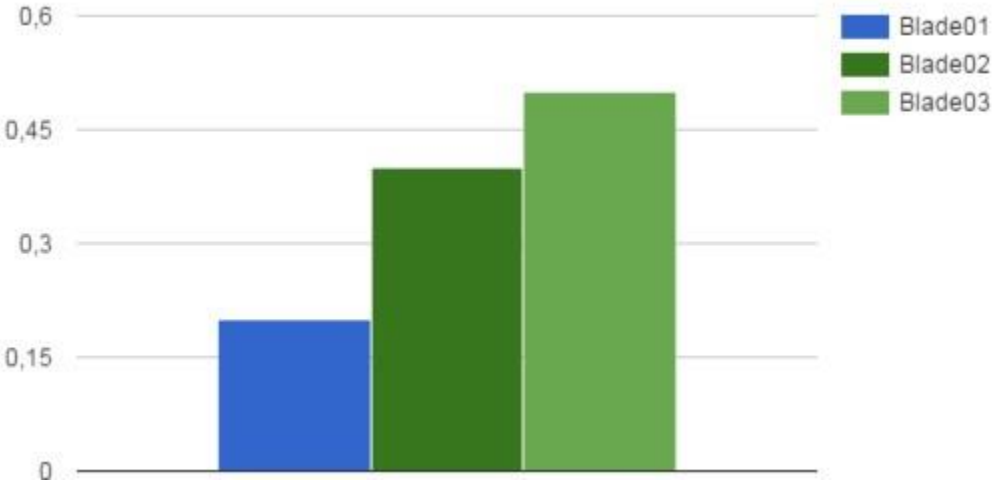

Рис. 4.3 Прихований стан апаратного забезпечення системи

Кількість користувачів 1, які потім збільшують лінійно, як видно на рисунку 4.3, до 100 протягом тривалості 720 секунд. Тести продовжували протягом 80 секунд зі 100 одночасними користувачами проти веб-сайту.

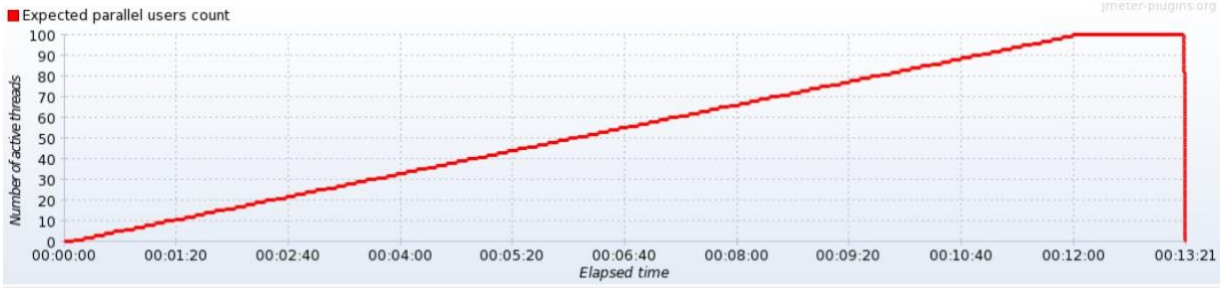

Рис. 4.4 Під час тесту показано навантаження користувачами на веб - сайт

Apache, для кожного користувача, запитав початкову сторінку встановлення WordPress на кожну з чотирьох налаштувань, а час, необхідно для обслуговування запитів зареєстрованих у файлі CSV. Процедура була

можлива, оскільки Apache зареєстрував час, коли імітаційний користувач надсилає запит HTTP та зареєстрований час коли користувач отримав відповідь. Кожна установка була перевірена п'ять разів. Незалежна змінна між тестами була різною, що видно на рисунку 4.4 . Файл CSV містив кожен успішний HTTP-запит на окремому рядку та з файлу, який ми вилучаємо інформацію, використовуючи два різних слухачів в Apache; відповідь час з часом і операціями в секунду. Іншими словами час відгуку з часом показав, наскільки швидко клієнт міг отримати відповідь від сервера на різних рівнях навантаження, а операції на секунду слухача показали, скільки успішних операцій, які були завершені сервером. З файлів CSV також можна було витягнути загальну кількість успішних транзакції завершені під час випробувань, підраховуючи загальну кількість рядків у файлах CSV віднімають 1 для кожного файлу, щоб не підрахувати заголовки. Різниця в ці три типи даних дозволили нам зробити висновки щодо того, наскільки добре різні налаштування виконується. Потім дані були проаналізовані шляхом порівняння не-віртуалізованих налаштувань віртуалізованого налаштування, а також порівнюючи віртуалізовані операційні системи, Centos 7, Coreos та технічний попередній перегляд PhotonOS до не-віртуалізованих операційних систем CentOS, Coreos.

Задача полягала в тому, щоб знайти та кількісно визначити накладні витрати, викликані віртуалізацією апаратного забезпечення, але також, щоб побачити, наскільки добре працює PhotonOS що до інших операційних систем як власники контейнера.

4.2.4. SQL Queries

Для вимірювання наскільки добре додатки в двох різних контейнерах на тому ж хості спілкуються один з одним та зчитують / записують дані, були проведені два тести. Перший тест вимірюється час, необхідний для завершення однієї SQL вибравши запит з веб-сервера до сервера бази даних. Другий тест вимірюється час, коли потрібно завершити 10 000 запитів SQLINSERT з веб-сервера до сервера баз даних. Для цього ми створили два

сценарії PHP, щоб виконати на кожному з веб-серверів. Проведено тести SQL-запитів на одній установці за один раз, з незалежними змінами, що використовуються, використовуються як видно на рисунку 25. Залежні змінні були часом, щоб завершити SQL-Select запит та час, необхідний для виконання 10 000 запитів SQL-вставки. Щоб запустити тест, були використані два скрипти PHP, один для SQL-Select запит та один для запитів SQL-insert. Обидва скрипти працювали, спочатку створюючи MySQL підключення до контейнера для бази даних, вказавши назву контейнера та ім'я користувача з паролем для сервера MySQL. Потім було зроблено запит, щоб встановити активну база даних до бази даних WordPress. У SQL вибраний тест був зроблений запит щоб отримати два записи з tablewp_users через запит SQL-Select. Сценарій записав поточний час і виконує запит одного часу, після чого він записав знову. Час, необхідний для завершення запиту, потім розраховували та надрукували на дісплей. Скрипт тесту SQL вставки працював так само, але замість цього зчитування таблиці, що містить дві посади, один раз, запит SQL-insert було зроблено 10 000 разів, запис простого слова "тест" до попередньо створеного порожнього столу. Сценарій закінчений, розрахунок часу для завершення запитів, і результат було надруковано на дісплеї.

Даний метод був заснований Джереєм Шопері у 2014 році. Метод, який ми використовували трохи відлічається від його, тому що не було детального опису проведення тесту. Ми створили наші власні PHP-скрипти, тому що вони не були присутніми у сценарії. Джереєм Шопер хотів виміряти різницю в продуктивності, коли запуск додатків всередині віртуальних машин у порівнянні з внутрішніми контейнерами та використанням макросекційних показників. Наша мета полягала в тому, щоб виміряти різницю продуктивності при запуску контейнерів на віртуальному контейнері власників у порівнянні з не-віртуалізованим контейнером. Навіть хоч наш дослідницьке питання трохи відрізнялося, я вирішила використовувати свій метод, оскільки метою його дослідження було те ж саме; для вимірювання

віртуалізованої продуктивності порівняно з не віртуалізованою продуктивністю. Звіт оновленого порівняння продуктивності віртуальних машин та контейнерів Linux за допомогою IBM. Мета була такою ж, як у звіті Джереєма Шопера, щоб навантажити ресурси в системах, порівняти та кількісно оцінити продуктивність та побачити відмінності. Різниця між двома методами полягала в тому, що дослідження IBM це була серія орієнтирів більш конкретних частин системи під час проведення досліджень Джереєм Шопері. Будучи макросерамічним методом засобу він вимірював продуктивність системи шляхом запуску програми, яка сама використовувала різні частини системи. Причина, чому ми не вибрали метод IBM, то це було важко зрозуміти і, отже, важко повторити. Ми вирішили виміряти продуктивність програми стек в цілому, а не лише окремі компоненти.

Hypervisor VMware ESXI обрали, завдяки тому, що ін. тісно пов'язаний з Photon OS, який призначений для запуску в контексті інтегрованих контейнерів Vsphere і налаштован на vsphere.

CentOS був обраний, як суперважлива операційна система підприємства, яке використовувало контейнер-хост.

Coreos обрали, як репрезентативний легкий контейнер операційної системи власника, оскільки вона спеціально розроблена, щоб виступати як операційна система власника контейнера та зазвичай використовується у контейнеризації. Для того, щоб імітувати це, тести могли бути встановлені, щоб зупинити в середньому числа з'єднання користувачів для веб-сайту, можливо, використовуючи 70% ресурсів веб-сервера. Однак це не було обрано, тому що накладні витрати були більш видимими при високі утилізація ресурсів.

4.3. Надійність та дійсність

Для підвищення надійності результатів кожен тест був проведений п'ять разів. Різниця між продуктивністю контейнеризованого застосування на не-віртуалізованих та віртуалізованих власниках контейнера повинна бути

схожою на всіх системах, використовуючи той же гіпервізор і метод тестування незалежно від використовуваного обладнання. Послідовність вимірюваннями продуктивності вводу/виводу була проведена для підвищення надійності результатів, оскільки вводу/виводу могло бути не достатньо. Ми не змогли запустити ту ж саму версію докера на всіх власниках контейнера. Остання версія Photon OS, як видно ми спробували запустити версії на інших контейнерах власника, близьких до версії на Photon OS. Загроза надійності за різними версіями могли б означати лише порівняння між різними операційними системами, оскільки як віртуалізується, так і не-віртуалізована установка однакова Container Host використовував ту ж саму версію Docker. Одна річ, яка, можливо, вплинула на результати, полягала у тому, що сам гіпервізор споживається близько 0-0,5% від процесора у віртуалізованих установках, оскільки він мав послуги, що прослуховують слухання адміністратора. Оскільки наші тести дуже швидко використовують всі обчислювальні ресурси, можливо, була внутрішня цільова загроза, оскільки використання ЦП гіпервізор може змінити продуктивність. Гіпервізор також спожив деяку пам'ять але наші випробування не напружують ресурси пам'яті систем.

4.4. Результати та аналіз

В даному розділі представимо проаналізовані результати наших тестів. Інформація буде надаватись для кожного графіка та таблиці. Текст містить аналіз результатів. Графіки ілюструють результати та містить аналіз результатів.

4.4.1. Запити HTTP

Результати, зазначені на рисунку 4.5, показали зменшення продуктивності в віртуалізованих середовищах порівняно з невірними середовищами. Не віртуалізована машина виступала на 20% краще в середньому, ніж віртуалізовані центи, а не віртуалізована машина Coreos

виконується на 27% краще в середньому, ніж віртуалізовані Coreos при порівнянні кількості успішних HTTP-запитів, заповнених під час тестів.

Однак слід зазначити, що перший тестовий запуск на віртуальній машині Coreos здійснюється значно гірше, ніж у власній середній, зменшення продуктивності 33%. Точна причина була невідомою, але ми можемо спекулювати, що під час тесту було виконано подію, як заплановану роботу. Дисконтування першого тесту на віртуалізований машині Coreos, продуктивність віртуалізованої машини Coreos у порівнянні з не віруалізованою машиною Coreos була на 18% нижче. За кількістю, середнє зниження продуктивності, коли апаратна віртуалізація контейнерного власника становила 19%. Результати на рисунку 4.5 показали, що HOST Photon OS Container виконував краще, ніж інша віртуалізована операційна система. Хоча все ще технічний попередній перегляд, VMware пояснює що Photon OS "перевіряється та налаштований для платформ постачальника VMware", яка могла б пояснити незначне збільшення продуктивності порівняно з іншими операційними системами.

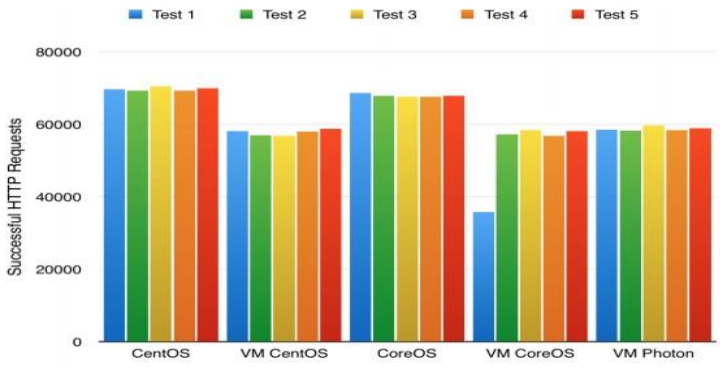

Рис.4.5 Успішні запити HTTP

Як видно на рисунку 4.5 операції за секунду збільшуються швидше в не віртуалізовані системи, але оскільки кількість транзакцій спочатку низька це ледве помітно. Різниця стала більш помітною при вищих операціях числа в секунду. Можливим поясненням, що накладні витрати кожної транзакції, невеликий у порівнянні з загальним часом транзакції, як і раніше. Накладні витрати були більш частковими, тобто які були віднесені до віртуалізації

апаратного забезпечення, додатковий абстрактний шар між застосуванням та ресурсами. Причиною зниження продуктивності також може бути внутрішнім віртуальним перемиканням гіпервізора. Віртуальні машини, що працюють на VMware Vsphere гіпервізора внутрішні віртуальні вимикачі, підключені до карткових карт фізичної мережі гіпервізору. Трафік, що йде до віртуальних машин, необхідних для проходу через один додатковий шар перемикачів, перш ніж він потрапить до заявок у порівнянні з не-віртуалізованим контейнером власників.

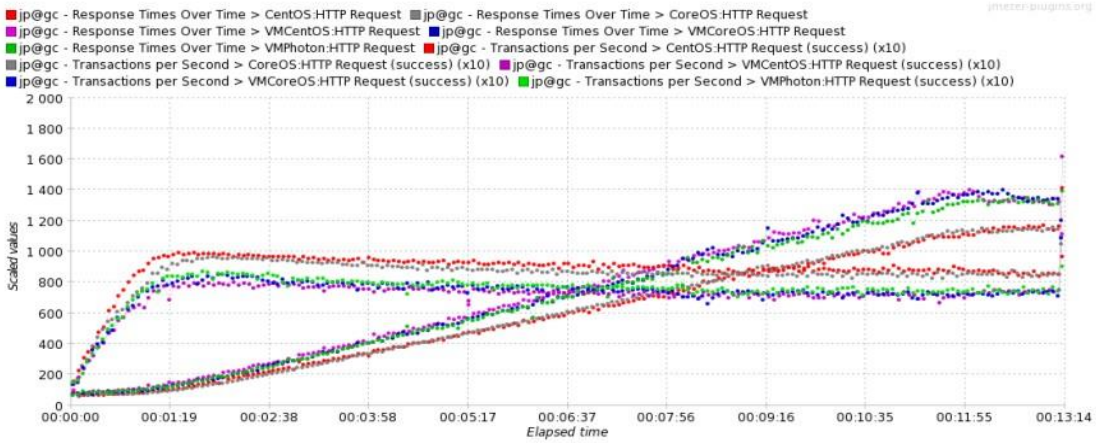

Рис.4.6 : Графік часу та кількості транзакцій за секунду

Графік на рисунку 4.6 представлений лише як репрезентативні результати щодо фактичного повного результату тесту. Це робиться через труднощі поєднання всіх результатів JMETER у графік, який можна легко тлумачити. На рисунку 4.6 кількість транзакцій за секунду зменшується до десяти, щоб зробити комбінацію графіків можливим. Час значно відрізнявся між різними тестами, хоча гранулярність часу була така ж сама, тобто 100 мілісекунд. З часом гранулярність, наприклад, 4 секунди, графіки були близькі до того ж, що може означати, що існує невелика проблема вимірювання. Тому що сервери були б повністю використані, деякі з одночасних запитів були покладені в утримання під час очікування часу CPU, щоб стати доступними. Оскільки час відгуку був виміряний на клієнті Apache Jmeter, час процесора, необхідний для завершення запиту, може бути однаковим, але порядок, в

якому були відправлені відповіді з сервера не були у порядку, до якого одночасні запити були відправлені з Apache клієнту.

4.4.2. SQL Queries

У другій частині тестів SQL Queries було зроблено з контейнера веб-сервера до контейнера бази даних, щоб виміряти різницю часу завершення. Середні результати для запитів SQL-SELECT представлені на рисунку 4.7. Значення середні за часом виконання, щоб заповнити єдиний SQL та вибрати запит, який менший.

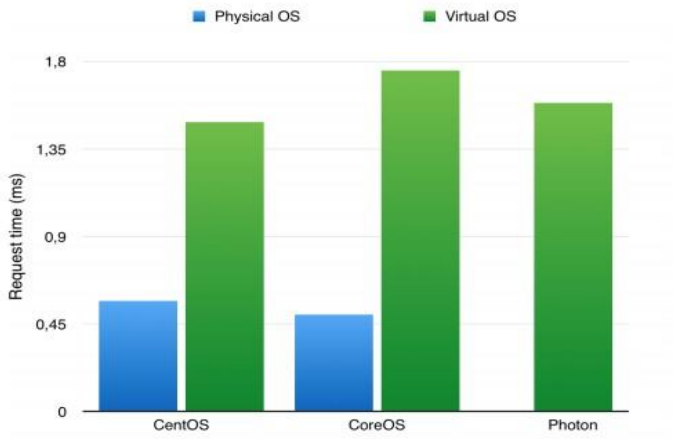

Рис. 4.7 : Графік операційної системи за часом виконання, щоб завершити обраний один SQL-запит

Результати тесту із запиту SQL-SELECT (рис. 4.7) показали збільшення часу виконання під час роботи віртуалізації операційних систем порівняно з не-віртуалізованими.

CentOS показало збільшення 162%, а Coreos показав збільшення на 253%. Photon OS не може бути порівненим, але виконується на рівні з іншими віртуалізованими операційними системами з різницею на 6,6% у порівнянні з CentOS та різницею 9,6% у порівнянні з Coreos. Різниця між операційними системи були нижчими в порівнянні з різницею між віртуалізованими або не віртуалізованими, з найбільшою різницею між CentOS та Coreos, що становить 15% віртуалізований.

Спостереження - це різний час, необхідний для завершення одного SQL між різними тестами роботи. Одне пояснення може бути те, що ми використовували фізичні жорсткі диски (з читанням і-запису), ймовірно, він не розташований у місці розташування даних шукаючи за запитом SQL-Select. Кожен запит тоді буде виступати як випадковий різний час зчитування завершення в результаті.

Результати від запитів SQL-insert, де час завершення 10 000 запитів SQLINSERT з веб-сервера до сервера баз даних, наведено на рисунку 4.8.

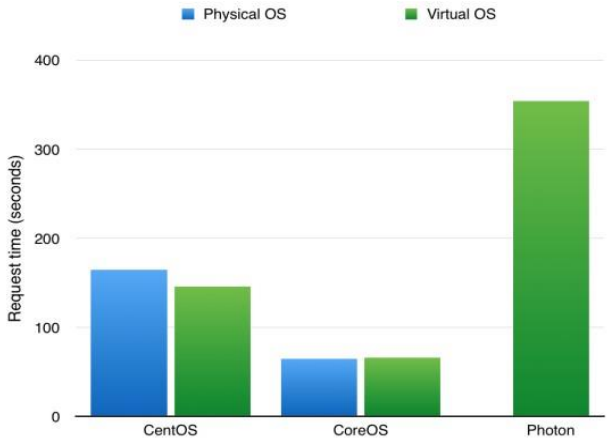

Рис.4.8 Операційні системи за часом виконання 10 000 SQL запитів

Результати з тестів запиту SQL-insert показали відносно низьку різницю між не-віртуалізованими операційними системами та віртуалізованими операційними системами, найбільша різниця для центрів є 13%. Різниця між операційними системи були значно більшими шляхом порівняння; Coreos мали найнижчий час виконання 65,7 секунд, в середньому, Centos становив 122% повільніше, ніж віртуалізований Coreos та PhotonOS склала 143% повільніше порівняно з віртуалізованими центром. При порівнянні результатів SQL-виборки та SQL-вставки один до одного ми зробили два спостереження: перший полягав в тому, що в SQL-insert випробовує середній час виконання віртуалізованого власника контейнерів були нижчими у порівняні з тим, що він запускає його не віртуалізованого. Одна теорія полягала в тому, що загальний час був взятий на запит даних з диску, який був суттєво більшим, ніж час, який був взятий для операційної системи, через гіпервізор, з запитом

до даних, які будуть записані. Це може означати, що для робочого часу операції, такі як запис даних, вартість продуктивності апаратної віртуалізації мала менший вплив. Інша теорія полягала у спостереженні пов'язаним з різницею між не-віртуалізованими та віртуалізованими результатами запитів SQL-Select. Результати показали, що існує різниця між не-віртуалізованим та віртуалізованим показником. Характер навантаження був повною протилежністю тесту на запит SQL-insert. Замість великої кількості записаної операції подвійного зчитування. Результати показали, що чим менша фізична робота (наприклад, I / O-операції), тим більше був би вплив на апаратного забезпечення віртуалізації.

В даному розділі розглянуто метод опису топології, який представлений у описі різних апаратних та програмних компонентів, які використовуються у даній роботі. Також було проведено три експерименти : один тест HTTP-Get, другий SQL тест та третій тест SQL-insert. Проведення експериментів показало, що потрібно для того, щоб виміряти продуктивність застосування всередині контейнерів, розгорнутої Docer, а також було проведено порівняння не-віртуалізованого контейнера хостів з віртуалізованим контейнером. Це допомогло визначити відмінності в контейнеризованому застосуванні між самими операційними системами контейнера.

ЗАГАЛЬНІ ВИСНОВОКИ ПО РОБОТІ

В даній роботі ми зробили дослідження та розглянули наслідки виконання програми при запуску контейнерів всередині віртуальних машин. Це було зроблено шляхом порівняння не віртуального контейнера на хості до віртуального контейнера власника і виміряли продуктивність двох їх застосувань. Наш висновок полягає в тому, що запуск віртуальних машин власника може вплинути на виконання їх застосування негативно з урахуванням абстрактного шару, який надається гіпервізором. Однак вплив віртуалізації контейнерних хостів відрізняється по типу операції, виконаної всередині додатків, що працюють на ньому. Ми виявили, що на великих кількостях послідовного запиту SQL продуктивність на віртуальних контейнерах власника навіть нарівні з фізичними. Навпаки було знайдено в тесті SQL-Select. З тесту запиту HTTP ми побачили 20% зменшення завершення запиту HTTP в середньому під час роботи на віртуальній машині вміcто фізичного устаткування. Не віруалізовані обидві операційні системи Coreos і Centos мали час відгуку з часом, а також більше транзакцій за секунду, ніж віртуалізовані Coreos і Cento.

Дані теста показують, що інтенсивні програми CPU побачать більшу роботу зменшення віртуалізованого середовища. Вибір контейнерної операційної системи - це складна проблема, оскільки всі вони пропонують різні переваги. Результати тестів запиту HTTP показують, що PhotonOS виконується краще у віртуалізованих установках, ніж Coreos і Centos, коли він приходить до загальної кількості заповнених запитів HTTP. Coreos і Photon OS також дуже надає невеликий розподіл, який у порівнянні з CentOS вимагає менше ресурсів пам'яті та пам'яті яка має бути встановлена і запущена. Однак у тестовій системі SQL-insert виконується все набагато гірше, то всі інші перевірені операційні системи, будь то віртуалізовані або не віртуалізовані. Це показує, що також важливо розглянути заявку перед вибором операцій системи для такого роду реалізації.

Ймовірно, що результати не повністю задаються. Це тому, що наші тести виявили, що вплив апаратної віртуалізації залежить від операції. При роботі важких операцій, як записувати на диск, вплив апаратної віртуалізації було незначним, але при роботі менших операцій, як невелика операція зчитування, вплив був великий. Наш висновок на основі цих результатів є те, що чим менше операція, тим більший вплив. Для того, щоб зробити результат більш узагальнюючим, необхідно зробити більше вимірювань при різних обсягах запису та зчитування. Накладні витрати, викликані віртуалізацією апаратного забезпечення також стають більш помітними у вищих рівнях віртуалізації у наших тестах, що означає за нашими результатами в тестах HTTS можуть бути лише загальними на тих рівнях віртуалізації.

Наш метод був заснований на попередніх дослідженнях Mathijs Jeroen Schiepers, але коли аналізували наші дані, ми виявили, що наші висновки можуть бути сильним шляхом поліпшення або доповнення наших тестів. У тестах SQL можуть бути порівняння між виконанням одного запиту та декількох запитів того ж типу, оскільки час Execute може суттєво відрізняється від даних зчитування та запису. Кожен з наших тестів включав лише один запит SQL-Select та 10 000 запитів SQL-Вставити.

У тесті запиту HTTP також було б краще перевірити системні рівні віртуалізації, щоб отримати зображення продуктивності на робочому навантаженні ближче до того, що було б робоче середовище. Можна зробити тест, щоб побачити, на якій кількості одночасного користувачі, що моделюються Apache, підштовхнуть систему до навантаження близько 70-90% CPU, а потім зробити лінійне збільшення від одного користувача до цього заданого числа. У наших поточних тестах, максимальна кількість транзакцій за секунду знаходиться при t = 100S (12% у тест), після чого зменшується, оскільки система перевантажена. Наші міркування для вибору методу полягає в тому, що ми хотіли підштовхнути віртуалізацію на високий рівень, щоб зробити накладні витрати апаратної віртуалізації більш помітними. Тепер ми

отримуємо картину того, скільки існує накладних витрат від апаратної віртуалізації у повністю використаній системі.

Ймовірно є одна річ, яка могла покращити наш метод, це також тестування, використовуючи менше апаратного забезпечення (CPU і пам'ять) для не-віртуалізованих операційних систем, а потім зменшити апаратне забезпечення для віртуальних машин до такої ж кількості процесора та пам'яті. У наших тестах віртуальна машина, що працює на гіпервізорі, була призначена всім обладнанням на віртуалізації хостів, тому що сам гіпервізор використовує лише 0-0,5% апаратних ресурсів згідно з нашим видом управління через клієнт Vsphere. Однак невідомо, якщо віртуальний контейнер власника був фактично конкуруючи за ресурси з гіпервізором під час високої фази віртуалізованих тестів, таким чином вплинуло на результати.

Одним з можливих характерних сценаріїв для такого роду реалізації є використання віртуальних машин як додаткових платформ, що складаються з декількох компонентів, де кожен компонент є контейнером. Платформа програми може бути веб-магазином та необхідними компонентами веб-сервера для хоста сайту для клієнтів. Також може бути сервер баз даних для утримання інформації про продукти та використовувану систему інвентаризації працівників певного підрозділу. Замість того, щоб запустити кожен компонент у власній віртуальній машині кожен компонент виконується як контейнер всередині однієї віртуальної машини. Компоненти будуть між технологічними, що є більш ефективним з порівнянням спілкування між віртуальними машинами, а також отримання апаратної ізоляції між ними різними платформами застосування, що працюють на тому ж фізичному апаратному засобі. Маючи всі компоненти, які необхідні для запуску додаткової платформи всередині тієї ж віртуальної машини робить перехід між фізичними машинами легше, тому що адміністратори не мають можливості переносу кожного компоненту окремо.

Будемо вважати, що ці результати мають відношення до IT-технологій, тому що навіть інші компанії такі як VMware та Microsoft вводять час, так і

гроші на розробку технологій щоб краще підтримувати віртуальні контейнери. Віртуальний контейнер власника приносить нові можливості адміністраторам центру обробки даних, але питання щодо наслідків виконання продуктивності, необхідно вирішити. Як ми вважаємо віртуальним контейнером власника буде основною частиною майбутніх центрів обробки даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. D. Kumar. (2015) Server consolidation benefits – with real world examples. (Accessed on 07/27/2016). [Online]. Available: http://www.sysprobs.com/ server-consolidation-benefits-with-real-world-examples.

2. Docker. (2016) What is docker? [Online]. Available: https://www.docker.com/ what-docker.

3. S. Hogg. (2014) Software containers: Used more frequently than most realize | network world. (Accessed on 05/09/2016). [Online]. Available: http://www.networkworld.com/article/2226996/cisco-subnet/ software-containers--used-more-frequently-than-most-realize.html.

4. M. J. Scheepers, “Virtualization and containerization of application infrastructure: A comparison,” 21st Twente Student Conference on IT, pp. 1–7, 2014.

5. R. Dua, A. Raja, and D. Kakadia, “Virtualization vs containerization to support paas,” in Cloud Engineering (IC2E), 2014 IEEE International Conference on, March 2014, pp. 610–614.

6. VMware, Inc. (2015) Photon OS by VMware. [Online]. Available: https: //vmware.github.io/photon/.

7. Canonical Ltd. What’s LXC? [Online]. Available: https://linuxcontainers.org/lxc/ introduction/.

8. W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE, 2015, pp. 171–172.

9. D. Kumar. (2014) Docker vs vms. (Accessed on 07/27/2016). [Online]. Available: http://devops.com/2014/11/24/docker-vs-vms/.

10. S. Seshachala. (2014, 11) Docker vs vms - devops.com. (Accessed on 05/20/2016). [Online]. Available: http://devops.com/2014/11/24/docker-vs-vms/ .

11. S. J. Vaughan-Nichols. (2014, 08) Why Containers Instead of Hypervisors? (Accessed on 05/24/2016). [Online]. Available: http://blog.smartbear.com/ web-monitoring/why-containers-instead-of-hypervisors/.

12. P. M. Mell and T. Grance, “Sp 800-145. the nist definition of cloud computing,” National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

13. Oracle. 1.1.1. Brief History of Virtualization. [Online]. Available: https: //docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html.

14. VMware, Inc. (2007, 09) Understanding Full Virtualization, Paravirtualization and Hardware Assist. [Online]. Available: https://www.vmware.com/files/pdf/ VMware_paravirtualization.pdf.

15. R. Vanover. (2009, 06) Type 1 and Type 2 Hypervisors Explained. (Accessed on 05/24/2016). [Online]. Available: https://virtualizationreview.com/blogs/ everyday-virtualization/2009/06/type-1-and-type-2-hypervisors-explained.aspx.

16. P. Rubens. (2015) What are containers and why do you need them? | cio. (Accessed on 05/12/2016). [Online]. Available:http://www.cio.com/ article/2924995/ enterprise-software/what-are-containers-and-why-do-you-need-them.html.

17. VMware, Inc., “vSphere ESXi Hypervisor,” 2016. [Online]. Available: http: //www.vmware.com/se/products/vsphere/features/esxi-hypervisor.html.

18. Vmware vsphere with operations management: High availability | vmware sverige. (Accessed on 05/06/2016). [Online]. Available: http: //www.vmware.com/se/products/vsphere/features/high-availability.

19. Vsphere fault tolerance: Vmware | vmware sverige. (Accessed on 05/06/2016). [Online]. Available: http://www.vmware.com/se/products/vsphere/ features/fault-tolerance.html .

20. B. Corrie. (2015, 06) Introducing Project Bonneville - Cloud-Native AppsCloudNative Apps - VMware Blogs. (Accessed on 05/24/2016). [Online].

Available: http://blogs.vmware.com/cloudnative/introducing-project-bonneville/ [21] CoreOS. Using CoreOS. (Accessed on 05/24/2016). [Online]. Available: https: //coreos.com/using-coreos/.

21. CoreOS. “Using fleet with coreos,” https://coreos.com/using-coreos/clustering/, (Accessed on 02/25/2016).

22. About centos. (Accessed on 05/06/2016). [Online]. Available: https: //www.centos.org/about/.

23. Docker. (2016) What is docker? (Accessed on 05/20/2016). [Online]. Available: https://www.docker.com/what-docker.

24. Netcraft Ltd. (2016) November 2015 Web Server Survey. [Online]. Available: http: //news.netcraft.com/archives/2015/11/16/november-2015-web-server-survey.html.

25. Apache Software Foundation. (2016) Foundation Project. [Online]. Available: http://www.apache.org/foundation/ .

26. Apache Software Foundation. (2016) Apache JMeter - User’s Manual: Introduction. [Online]. Available: http://jmeter.apache.org/usermanual/intro.html [28] M. AB, “Mysql ab :: Sun to acquire mysql,” https://web.archive. org/web/20080117192218/http://www.mysql.com:80/news-and-events/ sun-to-acquire-mysql.html, 01 2008, (Accessed on 05/03/2016).

27. T. T. FAQ, “Transactional database,” http://www.tech-faq.com/ transactional-database.html, 10 2012, (Accessed on 05/03/2016).

28. F. Penov. (2010) networking - what % of traffic is network overhead on top of http/s requests - stack overflow. (Accessed on 07/27/2016). [Online]. Available: http://stackoverflow.com/questions/3613989/ what-of-traffic-is-network-overhead-on-top-of-http-s-requests.

29. How to benchmark disk I/O. [Online]. Available: https://www.binarylane.com.au/ support/solutions/articles/1000055889-how-to-benchmark-disk-i-o.