5ST-O16: การประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันบนดอกเกอร์คอนเทนเนอร์

The evaluation on hardware utilization application of Docker container

ธนู คะชา[1] วีรยุทธ เจริญเรืองกิจ[1] และ ศุภร คนธภักดี[1*]

Tanoo Kacha[1], Werayuth Charoenruengkit[1] and Subhorn Khonthapagdee[1]

บทคัดย่อ

ในงานวิจัยนี้ได้ศึกษาการประเมินการใช้งานฮาร์แวร์ของดอคเกอร์คอนเทนเนอร์ ซึ่งโดยทั่วไปการเพิ่มจำนวนเว็บแอปพลิเคชันคอนเทนเนอร์สามารถเพิ่มประสิทธิผลตามจำนวนของดอคเกอร์คอนเทนเนอร์ที่เพิ่มขึ้น แต่จากการทดลองพบว่ามีการสูญเสียทรัพยากรบางส่วนในการจัดการดอคเกอร์คอนเทนเนอร์ที่ถูกเพิ่มขึ้น งานวิจัยนี้ทดสอบการใช้ทรัพยากรณ์ของเว็บแอปพลิเคชันคอนเทนเนอร์กับฐานข้อมูลที่อยู่บนเครื่องเซิฟเวอร์เพียงเครื่องเดียว โดยใช้ Nodejs เป็นเว็บแอปพลิเคชันคอนเทนเนอร์ เพื่อเรียกใช้ข้อมูลจาก Postgres database ในการทดลองจะเริ่มที่ 1 คอนเทนเนอร์ เพื่อเปรียบเทียบประสิทธิผลกับ 2 และ 4 คอนเทนเนอร์ผ่าน load balancer โดยแบ่งออกเป็น 3 แบบทดสอบเสมือนผู้ใช้งานจำนวน 100, 200 และ 300 คน ผลการทดสอบรายงานจากการหาเฉลี่ยค่า CPU usage, Response time, และ Throughput พบว่า การเพิ่มจำนวนคอนเทนเนอร์สามารถเพิ่มประสิทธิภาพในการประมวผลและลดเวลาในการตอบสนองของเว็บแอปพลิเคชั่น แต่เพิ่มประสิทธิภาพนั้นไม่เป็นแบบเส้นตรง เนื่องจากเซิฟเวอร์มีการแบ่งทรัพยากรบางส่วนใช้ในการจัดการคอนเทนเนอร์

คำสำคัญ: ดอกเกอร์ คอนเทนเนอร์ ทดสอบประสิทธิภาพ

Abstract

The purpose of this research is to evaluate hardware utilization of multiple Docker containers. Typically, the effectiveness can increase with the increasing number of Docker containers. However, our investigation finds that some resources have to be allocated to manage those containers. This research focused on examining the resource utilization of web application containers connecting to a database on a single server. Nodejs is used as a web container to retrieve data from a Postgres database. The research demonstrates the effectiveness of a single container, as a baseline to compare with 2 containers, and 4 containers via a load balancer. The experiments conduct 3 trials to simulate the usage of the system from 100, 200 and 300 concurrent users. The measurement indicators used in the experiment are the averaged CPU usage, Response time, and Throughput, the results demonstrate that increasing number of containers improve the performance and reduce web application responsiveness; however, the increasing efficiency is not a linear proportional to the number of containers because the server resources have to be allocated for the container management.

Keywords: Docker, containers, performance testing

[1] สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ

[1] Department of Computer Science, Faculty of Science, Srinakharinwirot University

[*] Corresponding author. E-mail: subhorn@g.swu.ac.th

บทนำ

ดอกเกอร์ (Docker) เป็นแพลตฟอร์มเปิด (open platform) ที่นำเอาเทคโนโลยี Linux Container (LXC) มาพัฒนาปรับปรุงเพื่อให้ใช้งานได้ง่ายยิ่งขึ้น (Vaucher, 2015) โดยการแยกแอปพลิเคชั่นออกจากโครงสร้างพื้นฐาน (Infrastructure) และยังสามารถจัดการโครงสร้างพื้นฐานได้ง่ายได้เช่นกัน ส่งผลให้การพัฒนาแอปพลิเคชันรวดเร็วและลดปัญหาการส่งมอบอีกด้วย (Docker, 2013) จึงเป็นเหตุให้ได้รับความนิยมอย่างแพร่หลาย

โดยทั่วไปแล้วดอกเกอร์สามารถทำงานพร้อมกันได้มากกว่า 1 คอนเทนเนอร์ (Container) Kubernetes หรือ Docker Swarm เป็นเครื่องมือในการจัดการจำนวนคอนเทนเนอร์บนแต่ละเครื่องเซิฟเวอร์ (server) เพื่อสมดุลการทำงานระหว่างเครื่องเซิฟเวอร์ต่างๆ ภานในคัสเตอร์ (Cluster) ซึ่งประสิทธิภาพของแอปพลิเคชันเพิ่มขึ้นความจำนวนของคอนเทนเนอร์ อย่างไรก็ตามยังมีปริมาณทรัพยากรสูญเสียเพิ่มเติมบางส่วน (overhead) ที่จำเป็นสำหรับดอกเกอร์ เพื่อจัดการคอนเทนเนอร์ที่เพิ่มจำนวนขึ้น จากการศึกษาของผู้วิจัยนั้นยังไม่พบรายงานเกี่ยวกับ Overhead เมื่อหลายคอนเทนเนอร์ทำงานบนเครื่องเซิฟเวอร์เดียวกัน

งานวิจัยนี้ทำการทดสอบประสิทธิภาพของแอปพลิเคชันที่ทำงานอยู่บนดอกเกอร์คอนเทนเนอร์ ในสถานการณ์ที่แตกต่างกัน โดยการเพิ่มจำนวนคอนเทนเนอร์ของแอปพลิเคชันบนเซิฟเวอร์เดียว เพื่อประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของดอกเกอร์และเปรียบเทียบประสิทธิภาพของการประมวลผลของจำนวนคอนเทนเนอร์ที่เพิ่มขึ้น จากการศึกษางานวิจัยที่เกี่ยวข้องกับ overhead โดยการทดสอบประสิทธิการทำงานของแอปพลิเคชันระหว่างบนเครื่องโดยตรง (bare-metal) กับบนดอกเกอร์พบว่าเมื่อ CPU ใช้งานอยู่ระหว่าง 65% ถึง 75% overhead อยู่ที่ประมาณ 10% แต่เมื่อ CPU ใช้งานมากกว่า 80% overhead จะลดลงน้อยกว่า 5% (Casalicchio,& Perciballi, 2017) เมื่อคอนเทนเนอร์ประมวลผลงานที่แตกต่างกัน คอนเทอเนอร์จะสามารถใช้งานทรัพยากรส่วนต่างๆ ได้อย่างดี ยกเว้นคอนเทอเนอร์ที่ประมวลผลงานในลักษณะเดียวกัน (Jha et al., 2018)

วัตถุประสงค์ของงานวิจัย

1. เพื่อประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันบนดอกเกอร์คอนเทนเนอร์
2. เพื่อเปรียบเทียบประสิทธิภาพในการประมวลผลเมื่อคอนเทนเนอร์มีจำนวนเพิ่มขึ้นบนเซิฟเวอร์เครื่องเดียว

วิธีการศึกษา

1. สภาพแวดล้อมการทดลอง

สภาพแวดล้อมในการทดลองแบ่งออกเป็นเซิฟเวอร์, โหลดเทส และ มอร์นิเตอร์ ซึ่งมีทั้งหมด 3 เครื่องโดยอยู่ในระบบเน็ตเวิร์ก (Network) เดียวกันเชื่อมต่อการทำงานด้วย API เพื่อควบคุมปัจจัยที่ไม่เกี่ยวข้องในการใช้ทรัพยากรของเครื่องเซิฟเวอร์ในระหว่างการทดลอง โดยในแต่ละสถานการณ์จำลองที่มีสถาปัตยกรรมแตกต่างกันจะถูกติดตั้งบนเซิฟเวอร์ที่มีขนาดทรัพยากรเท่ากัน ซึ่งมีขนาดทรัพยากรดังนี้

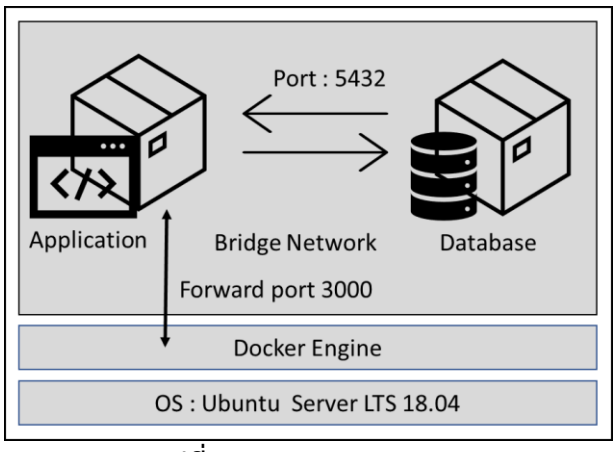
ตารางที่ 1 เซิฟเวอร์ แอปพลิเคชัน Resource

ทรัพยากร	รายละเอียด
CPU	6 Core
RAM	8 Gb
Storage	SSD 120 Gb
OS	Linux เซิฟเวอร์ 18.04 LTS
Docker	Docker CE Version 19.03.12

2. การออกแบบสถานการณ์จำลอง

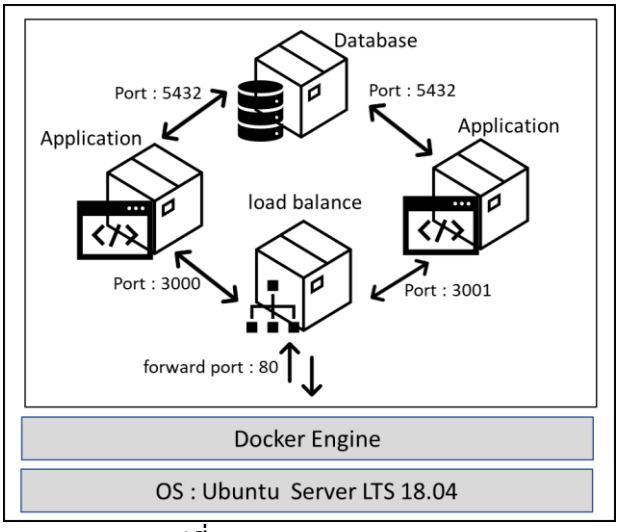
การออกแบบสถานการณ์จำลองในการทดสอบ ใช้เทคโนโลยีการจำลองสภาพแวดล้อมสำหรับซอฟต์แวร์ เรียกว่า คอนเทนเนอร์ ในงานวิจัยนี้ออกแบบสถานการณ์จำลอง 3 สถานการณ์ดังนี้

2.1 สถานการณ์ 1 ประกอบด้วย คอนเทนเนอร์ ทั้งหมดจำนวน 2 คอนเทนเนอร์ ได้แก่ แอปพลิเคชันและ database



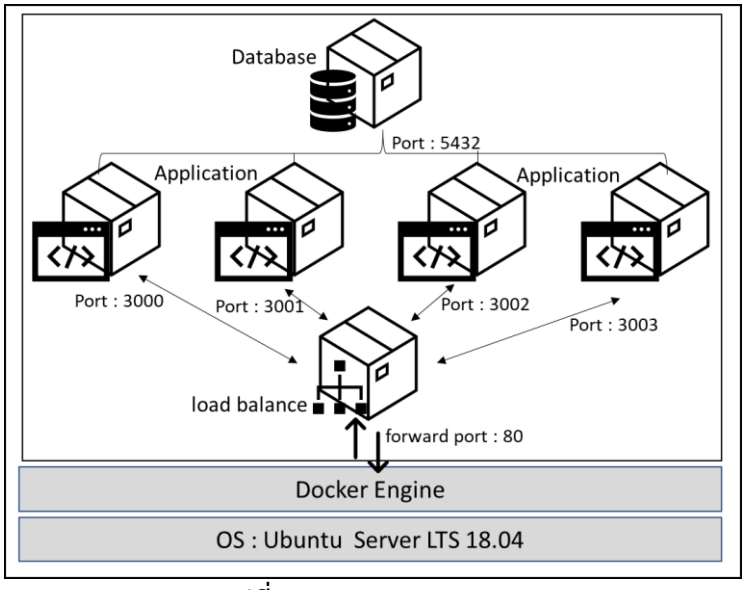
รูปที่ 1 Structure Scenario 1

2.2 สถานการณ์ 2 ประกอบด้วย คอนเทนเนอร์ ทั้งหมดจำนวน 4 คอนเทนเนอร์ ได้แก่ แอปพลิเคชัน 2 คอนเทนเนอร์, database และ load balance



รูปที่ 2 Structure Scenario 2

2.3 สถานการณ์ 3 ประกอบด้วยคอนเทนเนอร์ทั้งหมดจำนวน 6 คอนเทนเนอร์ ได้แก่ แอปพลิเคชัน 4 คอนเทนเนอร์, database และ load balance



รูปที่ 3 Structure Scenario 3

โดยแต่ละสถานการณ์มี database เพียงหนึ่ง คอนเทนเนอร์เท่านั้นเพราะ database ที่ใช้ทดลองเป็น standalone instance ทำงานและเก็บข้อมูลอย่างอิสระ ซึ่งการมี database มากกว่าหนึ่งคอนเทนเนอร์จะทำให้เกิดปัญหาความไม่สอดคล้องกันของข้อมูล ในสถานการณ์ 2 และสถานการณ์ 3 แตกต่างจากสถานการณ์ 1 เนื่องด้วยจำนวน คอนเทนเนอร์ของแอปพลิเคชันเพิ่มขึ้นส่งผลให้ต้องมี load balance สำหรับกระจายงานให้แอปพลิเคชัน

3. สภาพแวดล้อมแอปพลิเคชัน

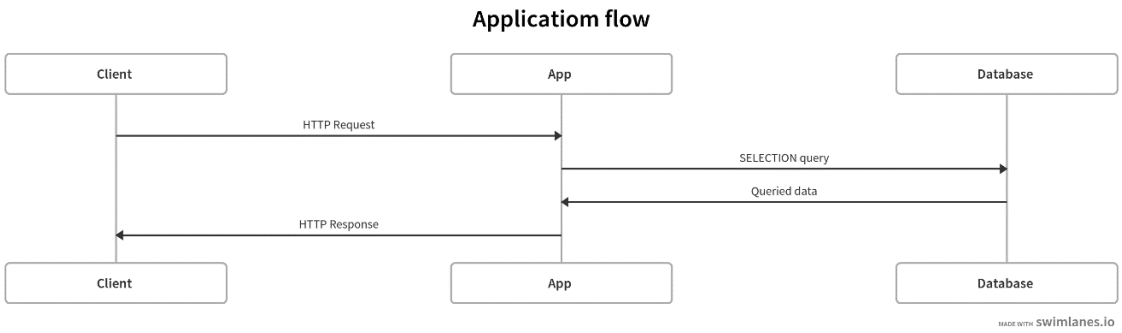
สภาพแวดล้อมแอปพลิเคชันเป็นการเชื่อมต่อการทำงานระหว่าง แอปพลิเคชัน กับ database โดยการทำงานแอปพลิเคชัน เรียกข้อมูลจาก database จำนวน 20 ชุด ซึ่ง แอปพลิเคชัน ถูกพัฒนาด้วย JavaScript โดยมี Nodejs เป็น run-time เชื่อต่อกับ database ด้วยวิธี Object Relational Mapping (ORM) โดยใช้ Sequelize มีการทำงานเชื่อต่อกับ database ในลักษณะ connection pool ได้ตั้งค่า pool การเชื่อมต่อมากที่สุดไว้ที่ 500 connection ในส่วนของdatabase ใช้ของ Postgres โดยตั้งค่าการเชื่อมต่อสูงสุดที่ 1,000 connection ข้อมูลภายในมีโครงสร้าง id และ name จำนวน 10,000 แถว และ Load Balance ที่เพิ่มขึ้นในสถานการณ์ 2 และ 3 ใช้ Nginx โดยเลือกใช้อัลกอริทึม Round Robin ทำการตั้งค่าการเชื่อมต่อการทำงานไว้ที่ 1,000 ซึ่งการเชื่อมต่อการทำงานระหว่าง Nginx, Nodejs และ Postgres ที่ได้ทำการตั้งค่าไว้เพื่อให้ระบบสามารถรองรับจำนวนผู้ใช้ อิมเมจซอฟแวร์ที่ใช้มีรายละเอียดดังนี้

ตารางที่ 2 Software Environment

อิมเมจซอฟต์แวร์	รายละเอียด
Nodejs	node:13-alpine
Postgres	postgres:12.1-alpine
Nginx	nginx:alpine

4. การออกแบบวิธีการทดลอง

การออกแบบวิธีการทดลองใช้วิธีการ Load Testing เป็นการทดสอบประสิทธิภาพประเภทหนึ่ง โดยออกแบบ Load Testing 3 กรณี เพื่อให้เข้าใจพฤติกรรมการใช้ทรัพยากรของ docker คอนเทนเนอร์ และนำผลการทดลองจากสถานการณ์ที่มีจำนวน คอนเทนเนอร์ไม่เท่ากันมาเปรียบเทียบกัน โดยการกำหนดพฤติกรรมการใช้งานระบบภายใต้สภาวะปกติและสูงสุด เพื่อให้มั่นใจว่าแอปพลิเคชันสามารถทำงานได้เมื่อมีผู้ใช้เข้าถึงในช่วงเวลาเดียวกัน โดยทำการจำลองขั้นตอนการเรียกข้อมูลของผู้ใช้ไปยังแอปพลิเคชัน จากนั้นแอปพลิเคชันเรียกข้อมูลไปยัง database และส่งกลับไปยังแอปพลิเคชันเพื่อตอบกลับไปยังผู้ใช้แสดงใน รูปที่ 4



รูปที่ 4 แอปพลิเคชัน flow

ในการสร้าง Load Testing ในงานวิจัยนี้เลือกใช้ K6 เป็น open sources โดยกำหนดพฤติกรรมของผู้ใช้ด้วย JavaScript ในการสั่งการทำงานของ K6 สั่งผ่าน CLI ในการออกแบบ Load Testing กำหนดพฤติกรรมของผู้ใช้แบ่งออกเป็น 3 ลักษณะดังนี้

4.1 TestCase 1 คือ อัตราการส่ง Request เริ่มต้นที่ 10 โดยในทุกๆ 10 วินาทีจะเพิ่มขึ้น 10 Request จนกระทั่งถึงอัตราส่ง 100 Request และคงไว้ที่อัตราดังกล่าวจนครบ 10 นาที

4.2 TestCase 2 คือ อัตราการส่ง Request เริ่มต้นที่ 10 โดยในทุกๆ 10 วินาทีจะเพิ่มขึ้น 10 Request จนกระทั่งถึงอัตราส่ง 200 Request และคงไว้ที่อัตราดังกล่าวจนครบ 10 นาที

4.3 TestCase 3 คือ อัตราการส่ง Request เริ่มต้นที่ 10 โดยในทุกๆ 10 วินาทีจะเพิ่มขึ้น 10 Request จนกระทั่งถึงอัตราส่ง 300 Request และคงไว้ที่อัตราดังกล่าวจนครบ 10 นาที

```
import http from 'k6/http'
import {check} from 'k6'

export let options = {
    stages: [
        {duration: '10s', target: 10},
        {duration: '10s', target: 20},
        {duration: '10s', target: 30},
        {duration: '10s', target: 40},
        {duration: '10s', target: 50},
        {duration: '10s', target: 60},
        {duration: '10s', target: 70},
        {duration: '10s', target: 80},
        {duration: '10s', target: 90},
        {duration: '10s', target: 100},
        {duration: '10m', target: 100},
    ],
}
export default function () {
  const response = http.get('http://192.168.1.80:3000/product/all/20')
  check(response, {
      'succeeded': r => r.status === 200,
  })
}
```

รูปที่ 5 K6 Load Testing TestCase 1

5. การวัดประสิทธิภาพ

ในการวัดประสิทธิภาพการทำงานของเซิฟเวอร์จะทำการทดสอบด้วย 3 TestCase ที่กำหนดไว้ข้างต้นด้วย K6 โดยจะรายงานผลข้อมูลของ Load Testing ซึ่งมีรายละเอียดดังนี้

1. Iteration คือ จำนวน request ทั้งหมดที่ส่งไปยัง เซิฟเวอร์
2. Response Time / sec คือ เวลาที่ เซิฟเวอร์ ตอบสนองกลับต่อ Client โดยเริ่มนับเวลาตั้งแต่เริ่มส่ง request จนกระทั่ง เซิฟเวอร์ ตอบกลับ
   I. Max เวลามากที่สุดที่ใช้ในการตอบกลับ
   II. Min เวลาน้อยที่สุดในการตอบกลับ
   III. Avg เวลาเฉลี่ยในการตอบกลับ
3. Success คือ การตอบกลับของ เซิฟเวอร์ ที่สำเร็จเท่าไหร่โดยวัดจาก HTTP response status codes 200 OK แสดงผลเป็นเปอร์เซน ซึ่งในการตั้งค่าให้สนใจเฉพาะโค้ด 200 KO เท่านั้น นอกนอกเหนือจากนั้นถือว่าเป็น Failure ทั้งหมด
4. Failure คือ การตอบกลับของ เซิฟเวอร์ ที่สำเร็จเท่าไหร่โดยวัดจาก HTTP response status codes แสดงผลเป็นเปอร์เซน
5. Throughput คือ ปริมาณงานที่ เซิฟเวอร์ สามารถประมวลผลได้และตอบกลับในเวลา 1 วินาที

โดยในการทดสอบในแต่ละ TestCase จะดำเนินการทดสอบทั้งหมด 3 ครั้ง และนำข้อมูลที่ได้มาวิเคราะห์เพิ่มเติมเพื่อตรวจสอบประสิทธิภาพ (วราภรณ์ และ นภวิชญ์, 2020)

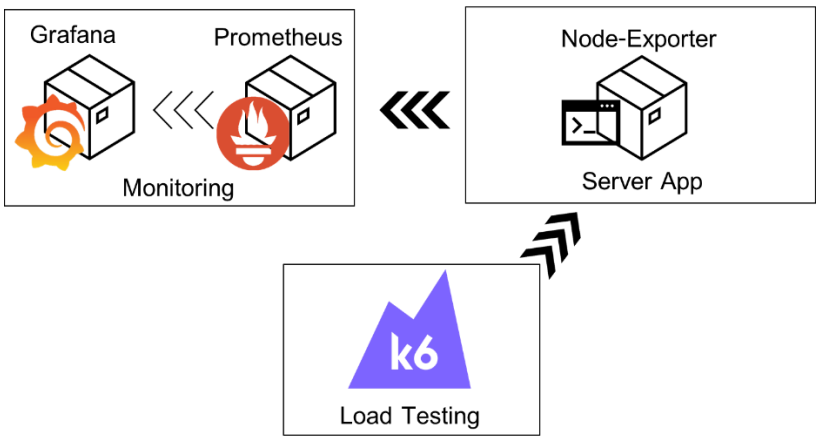
6. การตรวจสอบทรัพยากรเครื่อง(Hardware Resources)

การตรวจสอบทรัพยากรเครื่อง(Hardware Resources)โดยใช้เครื่องมือ open source ซึ่งในการตรวจสอบทรัพยากรเครื่อง ในงานวิจัยนี้ตรวจสอบทั้ง เซิฟเวอร์ และ Docker คอนเทนเนอร์ มีส่วนประกอบดังนี้

1. การตรวจสอบทรัพยากร เซิฟเวอร์ เครื่องมือที่ใช้คือ Node-Exporter เป็นเครื่องมือในการเก็บข้อมูลการใช้งานทรัพยากร Sever ทั้งหมด โดยข้อมูลที่เก็บเป็นวินาทีซึ่ง Node-Exporter ถูกติดตั้งอยู่ใน Linux OS

2. การแสดงผลข้อมูลทรัพยากรเนื่องจากตรวจข้อมูลทรัพยากรทั้ง เซิฟเวอร์ และ Docker คอนเทนเนอร์ ข้อมูลการใช้ทรัพยากรจาก Node-Exporter จะถูกส่งไปยัง database ที่มีลักษาณะการเก็บข้อมูลแบบ Time series โดยเลือกใช้ Prometheus และใช้ Grafana ในการแสดงผลข้อมูลการใช้งานทรัพยากร(Visualize) Grafana เรียกข้อมูลจาก Prometheus ในการแสดงผล

ในการตรวจสอบทรัพยากรเครื่อง Node-Exporter ถูกติดตั้งอยู่บน เซิฟเวอร์ แต่ Prometheus และ Grafana ถูกติดตั้งบนเครื่อง monitor โดยข้อมูลการใช้งานทรัพยากรถูกส่งผ่าน API เพื่อลดความแปรปรวนของ เซิฟเวอร์ ซึ่ง Prometheus และ Grafana ซึ่งทั้งสองโปรแกรมการติดตั้งในลักษณะคอนเทนเนอร์ (Casalicchio, & Perciballi, 2017)



รูปที่ 6 Environment Monitor

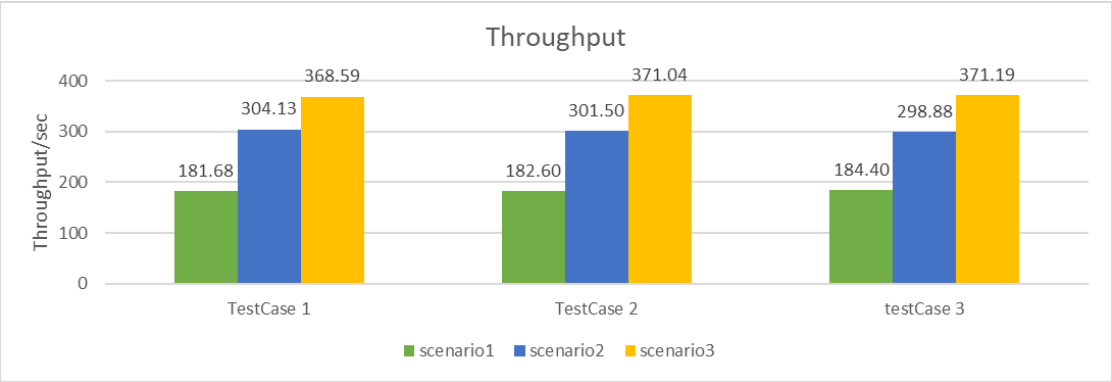
การตรวจสอบประสิทธิภาพได้รวบรวมข้อมูลการใช้งานทรัพยากร เซิฟเวอร์ โดยบันทึกข้อมูลการใช้งานทรัพยากรในช่วงเวลาตั้งแต่เริ่มการทดลองจนถึงจบการทดลองของแต่ละ TestCase ซึ่งทรัพยากรที่ใช้ในการตรวจสอบประสิทธิภาพเพื่อวิเคราะห์พฤติกรรมของแต่ละสถานการณ์จำลองมีดังนี้

1. CPU โดยประกอบด้วยข้อมูล 2 ส่วน
    I. System อธิบายการใช้งานทรัพยากร CPU ของระบบ โดยที่ไม่รวบกับการใช้งานของคอนเทนเนอร์ และ IRQs การใช้งานทรัพยากร CPU ในการจัดการ Interrupt ของระบบ
    II. User อธิบายการใช้งานทรัพยากร CPU ของ Docker คอนเทนเนอร์
2. RAM อธิบายปริมาณการใช้งาน RAM ทั้งหมดของ เซิฟเวอร์ โดยหน่วยวัดเป็น MB
3. Disk I/O อธิบายปริมาณการอ่านการและเขียนข้อมูลของ Disk โดยมีหน่อยวัดเป็น MB
4. Network อธิบายปริมาณการรับและส่งข้อมูลของ network

ผลการศึกษา

ผลจากการทดสอบด้วย Load Testing และผลการใช้งานทรัพยากรที่ได้รวบรวมข้อมูลนำมาเปรียบเทียบเพื่อประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชั่นบนดอกเกอร์คอนเทนเนอได้ผลดังนี้

1. Throughput



รูปที่ 6 Throughput

ภาพที่ 4 พบว่าแต่ละสถานการณ์มีอัตรา Throughput ที่เพิ่มขึ้น โดยจะเห็นได้ชัดระหว่างสถานการณ์ที่ 1 และสถานการณ์ที่ 2 แสดงให้เห็นว่าจำนวน แอปพลิเคชัน ที่เพิ่มขึ้นนั้นสามารถประมวลผล Workload ได้ปริมาณมากตามไปด้วยโดยที่ Throughput ที่เพิ่มขึ้นสามารถคิดเป็น % โดยการคำนวณตามสูตร

%Throughput ที่เพิ่มขึ้น = (Ty - Tx)/ Tx * 100 %

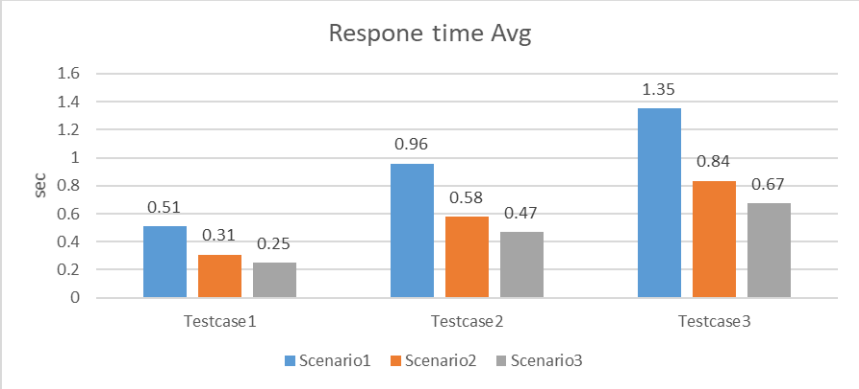
โดยที่ Ty เป็น Throughput ของสถานการณ์ที่กำลังเปรียบเทียบ และ Tx เป็น Throughput ที่เพิ่มขึ้นของสถานการณ์ที่มีการเพิ่มจำนวน คอนเทนเนอร์ร. ผลของการคำนวณแสดงในตารางที่ 2

ตารางที่ 3 Compare Throughput%

% Throughput ที่เพิ่มขึ้น	สถานการณ์ 2 vs สถานการณ์ 1	สถานการณ์ 3 vs สถานการณ์ 2
TestCase 1	67.40%	21.20%
TestCase 2	65.12%	23.06%
TestCase 3	62.08%	24.20%

ในอุดมคติการเพิ่มของ Throughput จากสถานการณ์ 2 เทียบกับสถานการณ์ 1 ควรเพิ่มขึ้นเป็น 2 เท่า เนื่องจากมีการเพิ่มจำนวน คอนเทนเนอร์ จาก 1 ไปเป็น 2 แต่พบว่า Throughput เพิ่มขึ้นได้สูงสุดแค่ 67.40% (ใน Testcase 1) เช่นเดียวกับการเพิ่มขึ้นของ Throughput จากสถานการณ์ 2 ที่มีการเพิ่มจำนวน คอนเทนเนอร์ จาก 2 คอนเทนเนอร์ เป็น 4 คอนเทนเนอร์ แต่การทดลองแสดงให้เห็นว่ามีการเพิ่มขึ้นของ Throughput สูงเพียงสุดเพียง 24.20% (ใน Testcase 3)

2. Response Time



รูปที่ 7 Responetime

ภาพที่ 7 แสดงผล Response Time ที่เปรียบเทียบในแต่ละสถานการณ์โดยสถานการณ์ 2 และ 3 มี Response Time ที่ลดลงแสดงให้เห็นว่าการจำนวน คอนเทนเนอร์ ที่เพิ่มขึ้นสามารถเพิ่มประสิทธิภาพของแอปพลิเคชัน โดยที่ประสิทธิภาพของ Response Time ที่เพิ่มขึ้นสามารถคิดเป็น % โดยการคำนวณตามสูตร

% ประสิทธิภาพ Response Time ที่เพิ่มขึ้น = (Ry - Rx)/Rx * 100 %

โดยที่ Ry เป็น Throughput ของสถานการณ์ที่กำลังเปรียบเทียบ และ Rx เป็น Throughput ที่เพิ่มขึ้นของสถานการณ์ที่มีการเพิ่มจำนวน คอนเทนเนอร์.ผลของการคำนวณแสดงในตารางที่ 4

ตารางที่ 4 Compare Response Time %

% ประสิทธิภาพ Response time ที่เพิ่มขึ้น	สถานการณ์ 2 vs สถานการณ์ 1	สถานการณ์ 3 vs สถานการณ์ 2
TestCase 1	66.09%	22.15%
TestCase 2	65.15%	23.07%
TestCase 3	61.55%	24.20%

ซึ่งในอุดมคติการลดลงของ Response Time จากสถานะการที่ 2 เทียบกับสถานะการที่ 1 ควรลดลงเป็น 2 เท่า หรือ % ประสิทธิภาพ Response Time เพิ่มขึ้น 100% เนื่องจากมีการเพิ่มจำนวน คอนเทนเนอร์ จาก 1 คอนเทนเนอร์เป็น 2 คอนเทนเนอร์ แต่พบว่า ประสิทธิภาพ Response Time กลับเพิ่มขึ้นได้แค่ 67.40% (ใน Testcase 1) เช่นเดียวกับ การเพิ่มขึ้นของ % ประสิทธิภาพ Response Time จากสถานะการที่ 2 ที่มีการเพิ่มจำนวน คอนเทนเนอร์ จาก 2 คอนเทนเนอร์เป็น 4 คอนเทนเนอร์ แต่การทดลองแสดงให้เห็นว่ามีการเพิ่มขึ้นของ % ประสิทธิภาพ Response Time ต่ำสุดเพียง 24.20% (ใน Testcase 3)

3. การตรวจสอบทรัพยากรเครื่อง(Hardware Resources)

ในงานวิจัยนี้มุ่งเน้นการตรวจสอบการใช้งานของ CPU เป็นหลัก เนื่องจาก แอปพลิเคชัน ที่ใช้ในการทดลองใช้งาน CPU ในการทำงานเป็นหลักซึ่งฮาร์ดแวร์ในส่วนอื่นมีปริมาณการใช้งานที่ต่ำไม่เห็นผลความแตกต่างในการใช้งานมากนัก โดยปริมาณการใช้งาน CPU เป็นช่วงเวลาที่ใช้งานสูงสุดจากนั้นนำมาหาค่าเฉลี่ยซึ่งได้ผลในตารางที่ 5

ตารางที่ 5 Average CPU Usage

Test Case	Scenario	System%	IRQs%	User%
Test Case1	Scenario1	4.54	0.49	29.31
	Scenario2	10.51	2.16	51.49
	Scenario3	19.79	2.48	71.51
Test Case2	Scenario1	4.32	0.44	28.86
	Scenario2	10.36	2.20	49.71
	Scenario3	19.53	2.50	71.29
Test Case 3	Scenario1	4.43	0.43	28.72
	Scenario2	10.47	2.23	50.20
	Scenario3	19.39	2.54	71.28

จากตารางที่ 5 การใช้งานส่วนใหญ่เป็นของ User ซึ่งเป็นการเข้าใช้งาน CPU ของคอนเทนเนอร์ โดยที่สถานการณ์ 3 มีปริมาณการใช้งานมากที่สุดนอกจากนี้ System และ IRQ ยังมีปริมาณการใช้งานที่เพิ่มขึ้นด้วยเช่นกันผลมาจากการเพิ่มจำนวน คอนเทนเนอร์ โดยปริมาณที่ใช้งาน CPU ของ User ที่เพิ่มขึ้นสามารถคำนวณตามสูตร

$$\%\text{User CPU ที่เพิ่มขึ้น} = (Uy - Ux)/Ux * 100\ \%$$

โดยที่ Uy เป็น User% ของสถานการณ์ที่กำลังเปรียบเทียบ และ Ux เป็น User% ที่เพิ่มขึ้นของสถานการณ์ที่มีการเพิ่มจำนวนคอนเทนเนอร์ผลของการคำนวณแสดงในตารางที่ 6

ตารางที่ 6 Compare User Usage CPU

% User ใช้งาน CPU เพิ่มขึ้น	สถานการณ์ 2 vs สถานการณ์ 1	สถานการณ์ 3 vs สถานการณ์ 2
TestCase 1	75.26	38.88
TestCase 2	72.24	43.39
TestCase 3	74.75	42.00

สรุปและอภิปลายผล

จากการทดสอบสถานการณ์จำลองทั้ง 3 สถานการณ์ โดยสถานการณ์ที่ 1 เป็นการจำลองให้เหมือนกันการใช้งานปกติที่มีหนึ่งคอนเทนอร์เพื่อดูพฤติกรรมการใช้งานทรัพยากร ในสถานการณ์ที่ 2 และ 3 ทำการเพิ่มแอปพลิเคชันซึ่งจะนำข้อมูลพฤติกรรมการใช้งานทรัพยากรและประสิทธิภาพการทำงานนำมาเปรียบเทียบกัน โดยแต่ละสถานการณ์

ได้ทำการทดลองโหลดเทส 3 ชนิด ที่แตกต่างกันและแต่ะการทดสอบได้ทำการทดสอบ 3 ครั้ง เพื่อลดความแปรปรวน โดยนำผลทั้ง 3 ครั้งมาเฉลี่ยกัน ซึ่งผลจาการทดลองสรุปว่าการเพิ่มจำนวนของคอนเทนเนอร์สามารถเพิ่มประสิทธิภาพในการประมวลผล (Throughput) และยังลดเวลาในการประมวผล (Respones Time) แต่ผลการทดลองพบว่าการเพิ่มจำนวน คอนเทนเนอร์ ไม่ได้เป็นอัตราส่วนเส้นตรงกับประสิทธิภาพที่ดีขึ้น กล่าวคือการเพิ่มจำนวน คอนเทนเนอร์เป็น 2 เท่าอาจไม่ได้ Throughput เป็น 2 เท่า หรือ ลด Response time ได้ 2 เท่า การเพิ่มจำนวนแอปพลิเคชันส่งผลให้เพิ่มปริมาณการใช้งานทรัพยากรเซิฟเวอร์ แต่ปริมาณที่เพิ่มขึ้นนั้นไม่ได้ถูกใช้โดยแอปพลิเคชันเพียงเท่านั้น ทรัพยากรยังถูกเรียกใช้โดยระบบ (System) เพิ่มขึ้นเช่นกัน เนื่องจากระบบต้องนำทรัพยากรมาใช้ในการบริหารจัดการคอนเทนเนอร์ที่เพิ่มขึ้น หรือที่เรียกว่า overhead เพราะฉะนั้นการเพิ่มจำนวนแอปพลิเคชันสามารถทำบนเซิฟเวอร์เครื่องเดียวและสามารถเพิ่มประสิทธิภาพการทำงานแต่การเพิ่มจำนวนแอปพลิเคชันจำเป็นต้องคำนึงถึง overhead ด้วยเช่นกัน

เอกสารอ้างอิง

Casalicchio, E., & Perciballi, V. (2017). Measuring docker performance: What a mess!!! Paper presented at the Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion.

Docker. (2013) Docker Overview . สืบค้น 15 มกราคม (2564) จาก https://docs.docker.com/get-started/overview/ .

Jha, D. N., Garg, S., Jayaraman, P. P., Buyya, R., Li, Z., และ Ranjan, R. (2018, 2-7 July 2018). A Holistic Evaluation of Docker Containers for Interfering Microservices. Paper presented at the 2018 IEEE International Conference on Services Computing (SCC).

Vaucher, S. (2015). Comparing virtual machines and linux containers. Recuperado a partir de https://zenodo. org/record/47644/files/Comparing_Virtual_Machines_and_Linux _Containers_-_Final. pdf.

Waraporn, V., วราภรณ์, ว., Srinakharinwirot University. Faculty of, S., Napawit, T., และ นภาวิชญ์, ท. (2020). Performance comparison between monolith and microservices using docker and kubernetes: การเปรียบเทียบประสิทธิภาพระหว่างสถาปัตยกรรมแบบโมโนลิธ และไมโครเซอร์วิสโดยใช้ดอคเกอร์และคูเบอร์เนตส์. In: Srinakharinwirot University.