การประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันด็อกเกอร์คอนเทนเนอร์

AN EVALUATION OF HARDWARE UTILIZATION BY APPLICATION DOCKER CONTAINERS

ธนู คะชา

บัณฑิตวิทยาลัย มหาวิทยาลัยศรีนครินทรวิโรฒ

2563

การประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันด็อกเกอร์คอนเทนเนอร์

ธนู คะชา

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ
ปีการศึกษา 2563

AN EVALUATION OF HARDWARE UTILIZATION BY APPLICATION DOCKER CONTAINERS

TANOO KACHA

A Master's Project Submitted in Partial Fulfillment of the Requirements

for the Degree of MASTER OF SCIENCE

(Information Technology)

Faculty of Science, Srinakharinwirot University

2020

สารนิพนธ์

เรื่อง

การประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันด็อกเกอร์คอนเทนเนอร์

ของ

ธนู คะชา

ได้รับอนุมัติจากบัณฑิตวิทยาลัยให้นับเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ของมหาวิทยาลัยศรีนครินทรวิโรฒ

(รองศาสตราจารย์ นายแพทย์ฉัตรชัย เอกปัญญาสกุล)

คณบดีบัณฑิตวิทยาลัย

คณะกรรมการสอบปากเปล่าสารนิพนธ์

..... ที่ปรึกษาหลัก

(ผู้ช่วยศาสตราจารย์ ดร.วีรยุทธ เจริญเรืองกิจ)

..... ประธาน

(อาจารย์ ดร.สุทธิพงศ์ ธัชยพงษ์)

..... กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.จันตรี ผลประเสริฐ)

ชื่อเรื่อง	การประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันด็อกเกอร์คอนเทนเนอร์
ผู้วิจัย	ธนู คะชา
ปริญญา	วิทยาศาสตรมหาบัณฑิต
ปีการศึกษา	2563
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ ดร. วีรยุทธ เจริญเรืองกิจ

งานวิจัยนี้มีวัตถุประสงค์เพื่อประเมินประสิทธิภาพของระบบและการใช้ทรัพยากรบนเซิร์ฟเวอร์เครื่องเดียวที่สอดคล้องกับการเพิ่มขึ้นของคอนเทนเนอร์แอปพลิเคชันโดยเริ่มต้นที่ 1 แอปพลิเคชันคอนเทนเนอร์เพื่อนำข้อมูลมาเปรียบเทียบประสิทธิผลกับ 2 และ 4 แอปพลิเคชันคอนเทอเนอร์ ซึ่งตัวชี้วัดที่นำมาศึกษาการใช้งานฮาร์ดแวร์คือ CPU Usage, Ram Usage และ disk Usage .ในการศึกษาประสิทธิภาพการทำงานใช้ Response Time และ Throughput ซึ่งผลการทดลองพบว่า การใช้งานฮาร์ดแวร์มีอัตราเพิ่มขึ้นตามจำนวนแอปพลิเคชันคอนเทนเนอร์ แต่การใช้งานฮาร์ดแวร์ในสถานการณ์ทั้ง 3 มีอัตราที่ใกล้เคียงกัน แม้จะมีอัตรา Work Load ที่เพิ่มมากขึ้น ส่วนประสิทธิภาพการทำงานมีอัตราที่เพิ่มขึ้นสูงสุดเพียง 66% เมื่อเปรียบเทียบระหว่าง 1 แอปพลิเคชันคอนเทนเนอร์กับ 2 แอปพลิเคชันคอนเทนเนอร์ นอกจากนี้ค่าดำเนินการยังมีอัตราเพิ่มสูงขึ้นอย่างมากตามจำนวนแอปพลิเคชันคอนเทนเนอร์อีกด้วย

คำสำคัญ : ด็อกเกอร์, คอนเทนเนอร์, ทดสอบประสิทธิภาพ, ค่าดำเนินการ

Title	AN EVALUATION OF HARDWARE UTILIZATION BY APPLICATION DOCKER CONTAINERS
Author	TANOO KACHA
Degree	MASTER OF SCIENCE
Academic Year	2020
Thesis Advisor	Assistant Professor Werayuth Charoenruengkit , Ph.D.

This research aims to evaluate the system performance and the resource utilization on a single server corresponding to an increment of application containers. The study started with one application container. The evaluated metrics were compared to two and four application containers. The system utilization metrics used by this study are CPU usage, RAM usage, and disk usage. The performance metrics are the response time and throughput. The results showed that hardware utilization rates increased with the number of container applications, but hardware utilization in all three scenarios showed similar rates despite the increasing workloads. The highest performance gains were at a maximum of 66% when increasing from one application container to two application containers. Furthermore, this research demonstrates that the overhead rate increased significantly with an additional number of application containers.

Keyword : Docker, Container, Performance Testing, Overhead

กิตติกรรมประกาศ

สารนิพนธ์นี้สำเร็จลุล่วงได้ด้วยดี เพราะได้รับความอนุเคราะห์จาก ผู้ช่วยศาสตราจารย์ ดร. วีรยุทธ เจริญเรืองกิจ ซึ่งเป็นอาจารย์ที่ปรึกษาสารนิพนธ์ได้ให้คำปรึกษาและให้ความช่วยเหลือชี้แนะแนวทางในสิ่งที่เป็นประโยชน์ต่อการทำสารนิพนธ์นี้ด้วยความเอาใจใส่ตลอดมา ผู้วิจัยขอขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบคุณคณะกรรมการสอบสารนิพนธ์ที่ได้ให้คำแนะนำและข้อเสนอแนะสำหรับการปรับปรุงสารนิพนธ์ในสมบูรณ์ยิ่งขึ้น

ขอขอบคุณคณาจารย์และกรรมการบริหารหลักสูตรสาขาวิทยาการข้อมูล คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒทุกท่านที่ได้ให้ความรู้แก่ผู้วิจัย

ขอบขอบคุณ พระมหาคเชนทร์ คุณปัญญา คุณเกษมพันธ์ และคุณรภัทร ให้การสนับสนุนและช่วยเหลือในการทำวิจัยครั้งนี้สำเร็จลุล่วงด้วยดี

ธนู คะชา

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

เทคโนโลยีเวอร์ชวลไลเซชั่น (Virtualization) หรือการจำลองเครื่องคอมพิวเตอร์เสมือน เพื่อให้สามารถใช้งานทรัพยากรเครื่องคอมพิวเตอร์ได้เต็มประสิทธิภาพ เมื่อกล่าวถึงการจำลองเสมือน ผู้คนทั่วไปมักจะนึกถึง Virtual Machine (VM) เพราะเป็นเวอร์ชวลไลเซชั่นที่ได้รับความนิยมสูง โดย VM คือใช้ไฮเปอร์ไวเซอร์เป็นตัวกลางในการจัดสรรทรัพยากรของเครื่องคอมพิวเตอร์ที่มีอยู่ แต่มีข้อเสียคือไฟล์อิมเมจที่จำลองออกมานั้นมีขนาดใหญ่ ทำให้เวลาในการสร้างและการส่งมอบใช้เวลานาน ในขณะที่การจำลองสภาพแวดล้อมในระดับระบบปฏิบัติการลีนุกซ์ (Linux) ที่เรียกว่า Linux Container (LXC) จะมีขนาดไฟล์อิมเมจที่เล็กกว่ามาก ทำให้เวลาในการสร้างและการส่งมอบสั้นลง ซึ่งหนึ่งใน Linux Container ที่ได้รับความนิยมจากนักพัฒนาอย่างมากตัวหนึ่งคือ ด็อกเกอร์ (Docker) เพราะเป็นแพลตฟอร์มเปิด (Open Platform) ที่สามารถนำมาพัฒนาต่อเพื่อให้การใช้งานคอนเทนเนอร์ได้ง่ายยิ่งขึ้น โดยแยกการพัฒนาแอปพลิเคชันออกจากโครงสร้างพื้นฐาน (Infrastructure) แต่ยังคงสามารถจัดการโครงสร้างพื้นฐานได้ ส่งผลให้การพัฒนาแอปพลิเคชันมีความสะดวก รวดเร็ว และลดปัญหาการส่งมอบแอปพลิเคชัน จึงทำให้ด็อกเกอร์ได้รับความนิยมอย่างแพร่หลายในหมู่ของนักพัฒนา (Developer) และเป็นตัวเลือกแรกๆ สำหรับการพัฒนาแอปพลิเคชัน

แอปพลิเคชันที่ดีจะต้องมีคุณสมบัติที่สำคัญประการหนึ่งคือ ความสามารถในการรองรับการเข้าใช้งานจำนวนมากในช่วงเวลาเดียวกัน ยกตัวอย่างเช่น แอปพลิเคชันของสถาบันการศึกษาที่ใช้สำหรับการประกาศผลสอบ การให้บริการของแอปพลิเคชันมักจะเกิดปัญหาแอปพลิเคชันทำงานล่าช้า หรือในบางครั้งถึงขั้นหยุดการตอบสนองเพราะระบบเกิดการล้มเหลว เมื่อมีผู้เข้าใช้งานจำนวนมาก ส่งผลกับบุคลากรครูและนักเรียนในการทำงานและติดตามผลการเรียน ซึ่งการแก้ปัญหาดังกล่าว ด้วยการเพิ่มจำนวนเซิร์ฟเวอร์ หรือจำนวนคอนเทนเนอร์ หรือเพิ่มประสิทธิภาพเครื่องเซิร์ฟเวอร์นั้น จะสามารถที่จะแก้ปัญหาได้ แต่หากองค์กรหรือสถาบันนั้นๆ มีงบประมาณที่จำกัด การแก้ปัญหาด้วยการเปลี่ยนหรือเพิ่มจำนวนเซิร์ฟเวอร์อาจจะเป็นเรื่องยากที่จะทำได้ ด้วยเหตุนี้ การเพิ่มแอปพลิเคชันคอนเทนเนอร์บนเครื่องเซิร์ฟเวอร์ที่มีอยู่แล้วจึงน่าจะเป็นทางเลือกที่ดีกว่าสำหรับการแก้ปัญหา

อย่างไรก็ตาม การพิจารณาเพิ่มคอนเทนเนอร์บนเครื่องเซิร์ฟเวอร์ยังคงประสบกับคำถามที่ว่า การเพิ่มจำนวนแอปพลิเคชันคอนเทนเนอร์บนเครื่องเซิร์ฟเวอร์นั้น จะสามารถทำให้

แอปพลิเคชันมีประสิทธิภาพที่แตกต่างจากเดิมอย่างมีนัยสำคัญหรือไม่ และส่งผลกระทบอย่างไรต่อเครื่องเซิร์ฟเวอร์ ในงานวิจัยนี้ ผู้วิจัยได้ทำการทดสอบประสิทธิภาพแอปพลิเคชันในสภาพแวดล้อมที่แตกต่างกัน ด้วยการกำหนดจำนวนของแอปพลิเคชันที่ 1, 2 และ 4 แอปพลิเคชันคอนเทนเนอร์ เพื่อศึกษาและประเมินสมรรถนะการใช้งานทรัพยากรเครื่องเซิร์ฟเวอร์ของด็อกเกอร์

1.2 วัตถุประสงค์

1.2.1 เพื่อประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของด็อกเกอร์คอนเทนเนอร์

1.2.2 เพื่อเปรียบเทียบประสิทธิภาพการประมวลผล เมื่อคอนเทนเนอร์มีจำนวนมากขึ้น

1.2.3 เพื่อศึกษาค่าดำเนินการ(Overhead) เมื่อจำนวนคอนเทนเนอร์ที่เพิ่มขึ้นบนเครื่องเซิร์ฟเวอร์เดียว

1.3 ขอบเขตงานวิจัย

1.3.1 การติดตั้งแอปพลิเคชันในการทดสอบทั้งหมดใช้ด็อกเกอร์เท่านั้น

1.3.2 ใช้งานเว็บแอปพลิเคชันในการทดสอบประสิทธิภาพ

1.3.3 ใช้ Physical Server ในการติดตั้งคอนเทนเนอร์เพื่อทำการทดสอบประสิทธิภาพ

1.3.4 ซอฟต์แวร์และเครื่องมือสำหรับการทดสอบทั้งหมดเป็นโอเพ่นซอร์ส (Open Source)

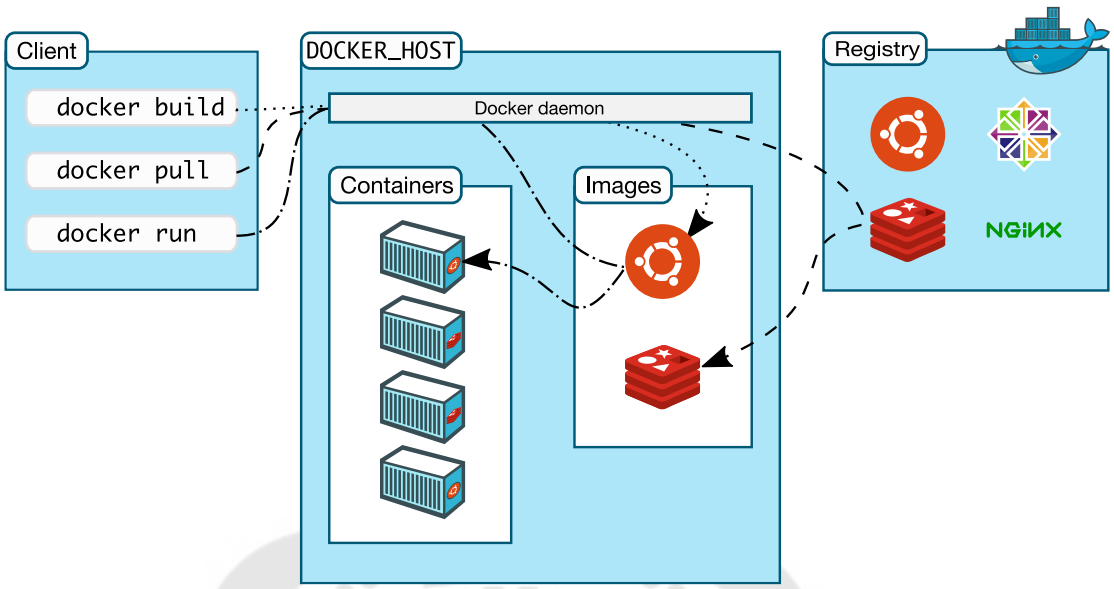
บทที่ 2
เทคโนโลยีและงานวิจัยที่เกี่ยวข้อง

2.1 Linux Container (LXC)

Linux Container (LXC) คือเทคนิคการจำลองเสมือน (Virtualization) อีกรูปแบบหนึ่ง โดยเป็นการจำลองเสมือนที่อยู่ในระดับของระบบปฏิบัติการ (Operating-System-Level) (Vaucher, 2015) ลักษณะการทำงานของ LXC จะทำงานร่วมกับเคอร์เนล (Kernel) ของระบบปฏิบัติการของเครื่องหลัก (Host OS) โดยไม่จำเป็นต้องติดตั้งระบบปฏิบัติการเพิ่มบนเครื่องหลัก เพื่อให้การจำลองเสมือนสามารถทำงานเป็นอิสระจากกันโดยไม่รบกวนการทำงานของเครื่องหลักและการจำลองเสมือนอื่นๆ (ขุนเพร็ชร์, 2015) ซึ่งการจำลองเสมือนรูปแบบนี้เรียกว่า คอนเทนเนอร์ (Container) LXC มีส่วนประกอบหลักสำคัญคือ เนมสเปซ (Namespaces) จะทำหน้าที่แยกการประมวลผลที่เกิดขึ้นของแต่ละส่วน เช่น IPC, UTS, Mount, PID, Network และ User เพื่อจำแนกการทำงานของคอนเทนเนอร์ออกจากกันและ Control Groups (cgroups) ดูแลควบคุมการใช้งานทรัพยากรและการจัดกลุ่มของการประมวลผลเพื่อจัดลำดับความสำคัญของการใช้ทรัพยากรของคอนเทนเนอร์ (linuxcontainers, 2018; Mavridis และ Karatza, 2017)

2.2 ด็อกเกอร์ (Docker)

ด็อกเกอร์เป็นโอเพ่นซอร์ส (Open Source) ด้วยการนำเทคโนโลยี LXC มาพัฒนาต่อยอดโดยมุ่งเน้นการ Build, Ship และ Run เพื่อช่วยให้นักพัฒนาสามารถใช้งานคอนเทนเนอร์ได้ง่ายและสะดวกมากยิ่งขึ้น แต่ใน Docker เวอร์ชัน 11 เป็นต้นไป ไม่ได้กำหนดให้ LXC เป็นสภาพแวดล้อมเริ่มต้นในการใช้งานด็อกเกอร์ โดยด็อกเกอร์พัฒนา libcontainer ที่พัฒนาขึ้นด้วยภาษา GO เพื่อแทนที่ LXC (Kwon และ Lee, 2020; Yasrab, 2018) ด็อกเกอร์มีโครงสร้างการทำงานในลักษณะ Client-Server โดย Client ทำหน้าที่สั่งการไปยังเซิร์ฟเวอร์ ด้วย Command Line Interface (CLI) ผ่านทาง REST API ซึ่งเซิร์ฟเวอร์หรือ Docker daemon ทำหน้าที่เป็นแกนกลางในการบริหารจัดการคอนเทนเนอร์ทั้งหมด ซึ่งโครงสร้างการทำงานนี้เรียกว่า Docker Engine (Docker, 2015) ด็อกเกอร์มีส่วนประกอบในการทำงานตามภาพประกอบ 1



ภาพประกอบ 1 โครงสร้างการทำงานของด็อกเกอร์

ที่มา https://docs.docker.com/get-started/overview/

สามารถอธิบายจากภาพประกอบ 1 ได้ดังนี้

2.2.1 Docker Images

Docker Image คือไฟล์ต้นแบบ (Template) สำหรับคอนเทนเนอร์ มีลักษณะเป็นไฟล์แบบ read-only ที่รวมไฟล์ที่จำเป็น เช่น library, middleware, OS และ network configuration เป็นต้น เพื่อให้แอปพลิเคชันสามารถทำงานได้ Docker Images สามารถดาวน์โหลดมาใช้งานได้ผ่าน Docker Registries อีกทั้งสามารถนำ Images ตั้งค่าหรือติดตั้งส่วนเสริมและสร้างใหม่โดยใช้ Dockerfile เพื่อให้เป็น Images ที่เหมาะกับสภาพแวดล้อมการทำงานของระบบ (Docker, 2015; Kwon และ Lee, 2020)

2.2.2 Docker Registries

Docker Registries คือแหล่งรวมของ Docker Images ที่ให้บริการในรูปแบบ Cloud ซึ่งมี Images จำนวนมากให้เลือกใช้งานและยังสามารถอัพโหลด Images ส่วนตัวเก็บไว้ได้เช่นกัน ทำให้การนำ Images ไปใช้ติดตั้งหรือส่งมอบงานง่ายยิ่งขึ้น

2.2.3 Docker Container

Docker Container คือการสร้างสภาพแวดล้อมสำหรับการทำงานของแอปพลิเคชัน โดยการสร้างสภาพแวดล้อมถูกกำหนดด้วย Docker Images ซึ่งคอนเทนเนอร์ที่ถูกสร้างขึ้นจะไม่ส่งผลกระทบกับการทำงานของคอนเทนเนอร์อื่นๆ บนเครื่อง

Docker Container สามารถเข้าถึงทรัพยากรของเครื่องเซิร์ฟเวอร์ได้เท่าที่ระบบปฏิบัติการจะจัดสรรให้ และยังกำหนดทรัพยากรให้แก่คอนเทนเนอร์ได้อีกด้วย นอกจากนี้ Docker Container สามารถกลับไปเป็น Docker Images โดยที่ยังคงค่าสถานะการทำงานของคอนเทนเนอร์ ณ จุดที่ต้องการ

2.3 เครื่องมือสร้างโหลดเทส K6

K6 เป็นเครื่องมือสำหรับการทดสอบประสิทธิภาพที่ออกแบบการทดสอบด้วยการเขียน JavaScript และใช้งานโดยไม่เสียค่าใช้จ่าย ซึ่งการออกแบบด้วยวิธีนี้จะช่วยให้นักพัฒนาสามารถทำงานได้สะดวกขึ้น แต่โดยทั่วไป JavaScript ไม่เหมาะกับการทำงานที่มีประสิทธิภาพสูง K6 จึงถูกพัฒนาด้วยภาษา GO และฝัง JavaScript Runtime เพื่อให้ JavaScript สามารถทำงานเทสโหลดประสิทธิภาพสูงได้และง่ายต่อการทดสอบ ในการทำงานของ K6 จะสั่งการทำงานและรายงานผลการทดสอบผ่าน Command Line Interface (CLI) (K6, 2019) ซึ่งภาพประกอบ 2 แสดงตัวอย่างของผลลัพธ์ที่ได้จาก K6 โดยมีรายละเอียดดังต่อไปนี้

2.3.1 vus จำนวนผู้ใช้เสมือนที่ใช้งานในปัจจุบัน

2.3.2 vus_max จำนวนผู้ใช้เสมือนสูงสุดที่เป็นไปได้

2.3.3 iteration จำนวนรวมของ Request ของ vus และจำนวนที่สามารถทำได้ในหนึ่งวินาที (Throughput)

2.3.4 iteration_duration เวลาในการตอบสนอง

2.3.5 check อัตราที่สามารถทำงานได้สำเร็จในการทดสอบ

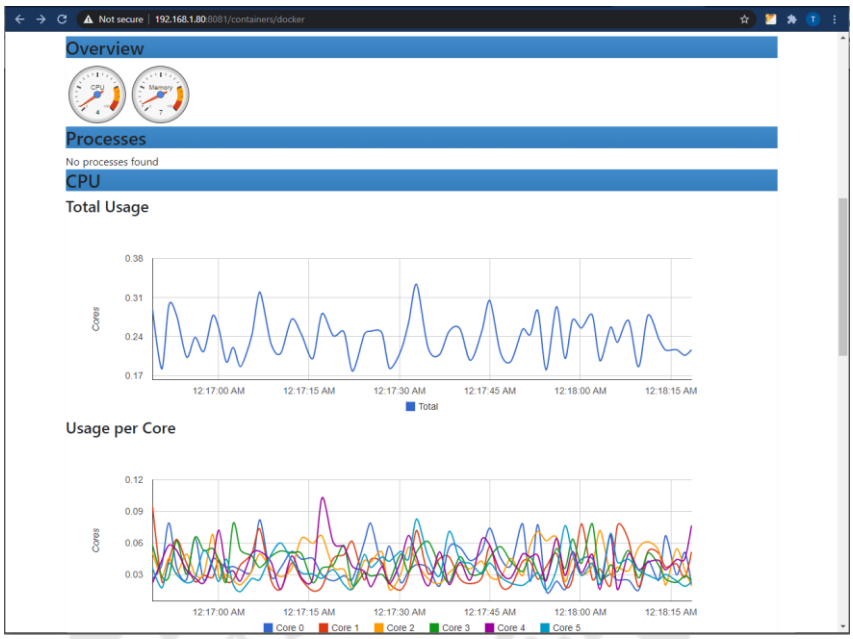
2.3.6 http_reqs จำนวน Request HTTP ที่สร้างขึ้นทั้งหมดโดย K6

```
         /\      |‾‾| /‾‾/   /‾‾/
    /\  /  \     |  |/  /   /  /
   /  \/    \    |     (   /   ‾‾\
  /          \   |  |\  \ |  (‾)  |
 / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: http_get_test.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 1m20s max duration (incl. graceful stop):
           * default: Up to 100 looping VUs for 50s over 5 stages (gracefulRampDown: 30s, gracefulStop: 30s)


running (0m50.3s), 000/100 VUs, 14000 complete and 0 interrupted iterations
default ✓ [======================================] 000/100 VUs  50s

     data_received..................: 37 MB  740 kB/s
     data_sent......................: 1.3 MB 26 kB/s
     http_req_blocked...............: avg=14.05µs  min=1.42µs  med=5µs     max=22.08ms  p(90)=6.48µs   p(95)=7.58µs
     http_req_connecting............: avg=7.22µs   min=0s      med=0s      max=21.97ms  p(90)=0s       p(95)=0s
     http_req_duration..............: avg=170.93ms min=7.59ms  med=73.67ms max=717.91ms p(90)=532.1ms  p(95)=580.69ms
       { expected_response:true }...: avg=170.93ms min=7.59ms  med=73.67ms max=717.91ms p(90)=532.1ms  p(95)=580.69ms
     http_req_failed................: 0.00%  ✓ 0     ✗ 14000
     http_req_receiving.............: avg=93.08µs  min=26.91µs med=93.54µs max=1.35ms   p(90)=118.66µs p(95)=127.31µs
     http_req_sending...............: avg=32.01µs  min=7.47µs  med=31.89µs max=581.84µs p(90)=38.71µs  p(95)=44.05µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s      max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=170.8ms  min=7.46ms  med=73.55ms max=717.78ms p(90)=531.99ms p(95)=580.59ms
     http_reqs......................: 14000  278.453764/s
     iteration_duration.............: avg=171.08ms min=7.71ms  med=73.83ms max=718.04ms p(90)=532.28ms p(95)=580.87ms
     iterations.....................: 14000  278.453764/s
     vus............................: 99     min=1   max=99
     vus_max........................: 100    min=100 max=100
```

ภาพประกอบ 2 ตัวอย่างการทำงานของ K6

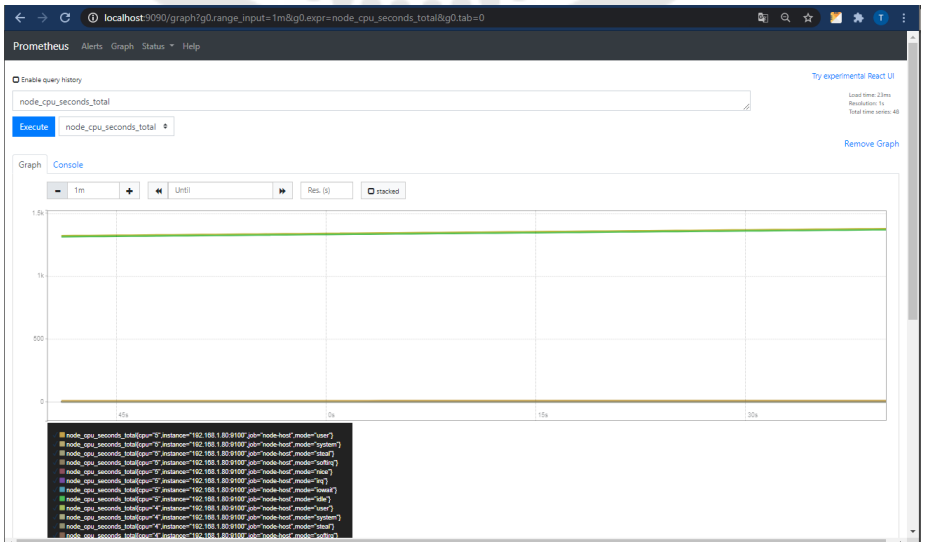
2.4 **เครื่องมือตรวจสอบการใช้งานทรัพยากร** Monitor Tool

2.4.1 **cAdvisor** (Container Advisor) คือเครื่องมือในการตรวจสอบการทำงานของคอนเทนเนอร์ โดย cAdvisor ทำงานในสภาพแวดล้อมคอนเทนเนอร์ ด้วยการรวบรวมข้อมูลการใช้งานทรัพยากรและลักษณะการทำงานของคอนเทนเนอร์ ซึ่งในแต่ละคอนเทนเนอร์จะเก็บข้อมูลการใช้งานทรัพยากรแยกออกจากกัน เช่น CPU, RAM และ Network เพื่อนำมาวิเคราะห์และรายงานผลข้อมูลเกี่ยวกับการทำงานในแต่ละคอนเทนเนอร์โดยแสดงผลผ่าน Web browser ดังภาพประกอบ 3



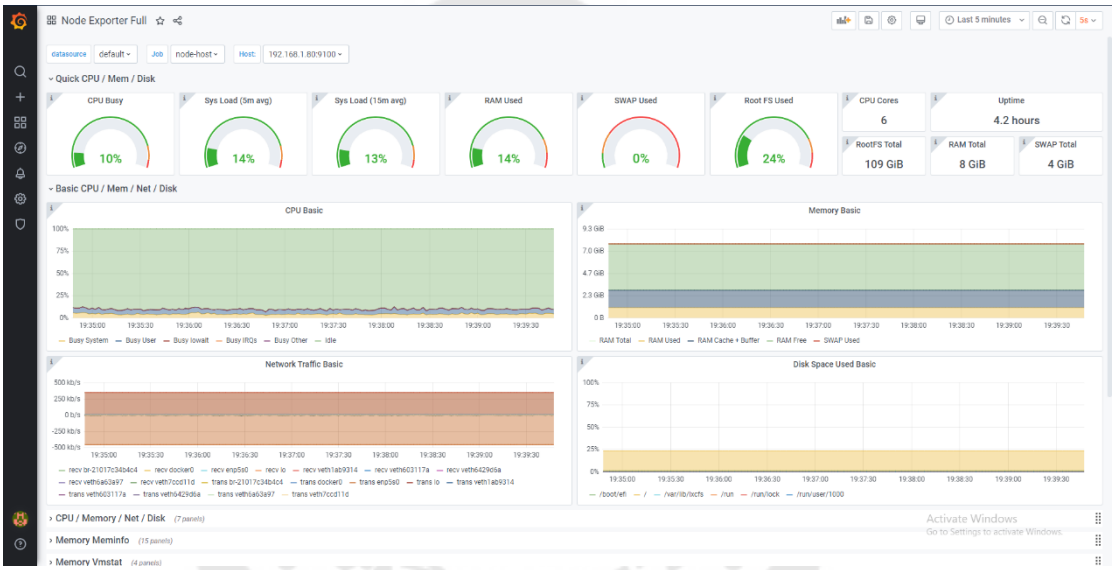
ภาพประกอบ 3 การแสดงผลข้อมูลของ cAdvisor

2.4.2 **Prometheus** เป็นเครื่องมือตรวจสอบและแจ้งเตือนการทำงานของระบบ โดย Prometheus เริ่มต้นมาจากการพัฒนาให้กับโปรเจค SoundCloud ในปัจจุบันเป็นโครงการโอเพ่นซอร์สที่แยกตัวออกมาโดยเข้าร่วมกับ Cloud Native Computing Foundation ซึ่ง Prometheus จัดเก็บข้อมูล metric ในรูปแบบ Time series ในการเรียกเก็บข้อมูล Prometheus ใช้ node exporter เป็นตัวช่วยในการส่งข้อมูลการใช้งานฮาร์ดแวร์ โดยส่งข้อมูลผ่าน API และสามารถเก็บข้อมูลได้จากหลายแหล่งพร้อมกันดังในภาพประกอบ 4



ภาพประกอบ 4 การแสดงผลข้อมูลของ Prometheus

2.4.3 **Grafana** คือเครื่องมือในการสร้าง Dashboard เพื่อนำเสนอข้อมูล Matric ที่เฉพาะเจาะจง ในรูปแบบของกราฟ Time series โดย Grafana จะทำงานร่วมกับ Data Source ต่าง ๆ เช่น Graphite, Influx DB, OpenBSD, Elasticsearch และ Prometheus เป็นต้น จะช่วยให้ผู้ใช้งานสร้างและแก้ไข Dashboard รวมถึงกำหนดเส้นทางการแจ้งเตือนสำหรับการกำหนดค่าต่าง ๆ ได้อย่างสะดวก Grafana มีรูปแบบการแสดงผลของกราฟและข้อมูลในตารางที่สวยงาม โดยข้อมูลที่แสดงอยู่บนกราฟจะแสดงข้อมูลในรูปแบบกึ่ง Realtime (ไม่เกิน 5 วินาที)และสามารถ Export ข้อมูลออกมาได้หลากหลายรูปแบบดังภาพประกอบ 5 ที่แสดงตัวอย่างการแสดงผลการทำงานของเครื่องเซร์ฟเวอร์



ภาพประกอบ 5 การแสดงผลของ Grafana

2.5 งานวิจัยที่เกี่ยวข้อง

(1) บทความวิจัยเรื่อง A Holistic Evaluation of Docker Containers for Interfering Microservice (Jha และคนอื่น ๆ, 2018) งานวิจัยนี้ได้ทำการทดสอบประสิทธิภาพของด็อกเกอร์คอนเทนเนอร์ในเรื่องของการแย่งทรัพยากรตัวเครื่อง โดยวิธีการทดสอบแบ่งออกเป็น 3 เหตุการณ์ หนึ่งคอนเทนเนอร์ติดตั้งหนึ่ง Microservice เป็นเหตุการณ์ที่ 1 เพื่อเป็นบรรทัดฐาน (Baseline) ในการกำหนดค่าทรัพยากรของคอนเทนเนอร์ เหตุการณ์ที่ 2 จะเป็นหนึ่งคอนเทนเนอร์ที่ภายในติดตั้งหลาย Microservice โดยไม่จำกัดทรัพยากรของคอนเทนเนอร์ แต่จำกัดที่ทรัพยากรเครื่องโดยจะต้องมากกว่า Baseline เท่าตัว เหตุการณ์ที่ 3 หลายคอนเทนเนอร์ติดตั้งเพียงหนึ่ง

Microservice (ใช้วิธีการทำ Load Balance) โดยมีการแยกเหตุการณ์ย่อยออกมา คือ ไม่จำกัดทรัพยากรและจำกัดทรัพยากรของคอนเทอเนอร์ ซึ่งในการทดสอบนั้น Microservice ที่นำมาใช้คือ Linpack, STREAM ,Bonnie++ และ Netperf เป็นเครื่องมือสำหรับการทดสอบการใช้งานของทรัพยากรเครื่อง

(2) บทความวิจัยเรื่อง Measuring Docker Performance : What a mess!! (Casalicchio และ Perciballi, 2017) งานวิจัยนี้เป็นการทดสอบประสิทธิภาพของด็อกเกอร์เพื่อดูค่าใช้จ่ายในการทำงานที่มีภาระงานสูงเมื่อเทียบกับ Bare-Metal โดยสนใจที่การทำงานของ CPU และ Disk I/O เครื่องมือที่ใช้การสร้างภาระงาน sysbeach ส่วนเครื่องมือในการตรวจสอบประสิทธิภาพได้แก่ mpstat และ iostat เป็นเครื่องมือตรวจสอบฝั่งของ Host ส่วนฝั่งของด็อกเกอร์ใช้ cAdviser และ docker stat โดยในการวัดประสิทธิภาพหรือหาค่าใช้จ่ายของ CPU นั้น ดูที่เปอร์เซ็นต์การใช้งาน CPU และนำมาเปรียบเทียบระหว่าง Host กับ Docker ซึ่งค่าใช้จ่ายอยู่ที่ 5% – 10% ส่วน disk I/O ทำในลักษณะเดียวกันโดยค่าใช้จ่ายอยู่ที่ 10% – 30% ซึ่งผลลัพธ์ที่ออกมาในส่วนของ CPU นั้น ทำได้ชัดเจนเนื่องจากมีเครื่องมือตรวจสอบหลายส่วน ทำให้ได้ค่าการใช้งานที่ชัดเจน ส่วน disk I/O มีเพียง iostat ในฝั่ง Host ส่วนในฝั่งของ Docker นั้น cAdviser ไม่ได้ให้ข้อมูล I/O จึงทำให้ค่าใช้จ่าย I/O ไม่ชัดเจน

(3) บทความวิจัยเรื่อง Using Docker in High Performance Computing Applications (Chung, Quang-Hung, Nguyen, และ Thoai, 2016) ในงานวิจัยนี้ทำการประเมินสมรรถนะระหว่าง Virtual Machines (VMs) กับ Docker container บนเครื่องคอมพิวเตอร์ประสิทธิภาพสูง (High Performance Computer) ด้วยวิธีการทดสอบประสิทธิภาพโดยใช้ High Performance Linpack (HPL) ทดสอบการประมวลผลของ CPU และใช้ Graph 500 ทดสอบการทำงานของ RAM ซึ่งในการทดสอบกำหนดสภาพแวดล้อมที่แตกต่างกัน โดยการเพิ่ม Instances ทั้ง VMs และ Docker container ในการทดสอบแต่ละครั้ง ซึ่งเพิ่มในจำนวนที่เท่ากันดังนี้ 2, 4, 8, 16 และ 32 ผลการทดสอบด้วย HPL VMs สามารถประมวลผลได้ดีกว่า Docker container ที่ 2 และ 32 Instances ส่วนด็อกเกอร์สามารถประมวลผลได้ดีกว่า VMs ที่ 4, 8 และ 16 Instances การเพิ่มขึ้นของจำนวน Instances สามารถเพิ่มประสิทธิภาพการประมวลผล แต่สูญเสีย Overhead เพิ่มขึ้นตามจำนวน Instances ที่เพิ่มขึ้นด้วยเช่นกัน ในการทดสอบด้วย Graph 500 Docker Container มีประสิทธิภาพการเรียกข้อมูลได้เร็วกว่า VMs ที่ 2, 4, 8 และ 16 Instances VMs กลับมีประสิทธิภาพที่ลดลงเป็นเส้นตรง

(4) บทความวิจัยเรื่อง Performance Comparison between Container-based and VM-based Services (Salah, Zemerly, Yeun, Al-Qutayri, และ Al-Hammadi, 2017) ในงานวิจัยนี้ทดสอบประสิทธิภาพระหว่างคอนเทนเนอร์ที่ทำงานอยู่บน AWS Elastic Container Service (ECS) และ VMs ที่ทำงานอยู่บนระบบ AWS Elastic Compute Cloud (EC2) โดยในการทดสอบได้แบ่งออกเป็น 3 สถานการณ์ เพื่อศึกษาการทำงานที่ใช้ Web Service สำหรับการทดสอบโดยมาตรวัดที่ตรวจสอบประสิทธิภาพมี 3 ส่วนคือ Throughput, Response Time และ CPU Utilization เครื่องมือที่ใช้สร้างภาระงานคือ JMeter ผลการทดลอง EC2 มี Throughput ที่มากกว่า ECS ทั้ง 3 สถานการณ์ การใช้งาน CPU EC2 และ ECS มีอัตราใช้งานใกล้เคียงกัน ส่วน Response Time EC2 ต่ำกว่า ECS ทั้ง 3 สถานการณ์

(5) บทความวิจัยเรื่อง A performance evaluation between Docker container and Virtual Machines in cloud computing architectures (Herrera-Izquierdo และ Grob, 2017) ในงานวิจัยนี้ทดสอบประสิทธิภาพของ Virtual Machines (VMs) และ Docker container เพื่อเปรียบเทียบประสิทธิภาพการทำงานบนเครื่อง High-performance computing (HPC) เพื่อทดสอบการประมวลผลของ CPU โดยใช้ High Performance Linpack (HPL) ในการทดสอบและทดสอบบนสภาพแวดล้อมที่แตกต่างกันคือการเพิ่มจำนวน Instances ที่ 4, 8 และ 16 Instances ผลการทดสอบ Docker container สามารถประมวลผลดีกว่า VMs

จากการศึกษางานวิจัยในข้างต้นทดสอบการทำงานระว่าง VMs กับ Docker Container หรือ Bare-Metal บนเครื่อง HPC และ EC2 แต่ด้วยข้อจำกัดงบประมาณของงานวิจัย ผู้วิจัยจึงจำกัดขอบเขตงานวิจัยด้วยการศึกษาการใช้ทรัพยากรและประสิทธิภาพการทำงานของระบบด้วยเครื่องคอมพิวเตอร์ทั่วไปเพียงเครื่องเดียว ซึ่งทำให้สามารถกำจัดปัจจัยด้านความแตกต่างของเครื่องคอมพิวเตอร์ และความไม่แปรปรวนของ Networks ที่จะมีผลต่อการวัดประสิทธิภาพ นอกจากนี้ ผู้วิจัยได้เลือกใช้แอปพลิเคชันทั่วไปที่ใกล้เคียงกับการใช้งานระบบในการทดสอบเพื่อให้เห็นลักษณการทำงานที่ใกล้เคียงกับการทำงานที่เกิดขึ้นจริง แทนการใช้เครื่องมือ Benchmark ในบางงานวิจัยใช้ในการทดสอบประสิทธิภาพ ซึ่งงานวิจัยนี้ต้องการศึกษาประสิทธิภาพที่เกิดขึ้นจากการเพิ่มจำนวนของคอนเทนเนอร์หรือการเพิ่ม Instances และการใช้เครื่องมือในการตรวจสอบการใช้ทรัพยากรของเซิร์ฟเวอร์เพื่อศึกษาการทำงานของคอนเทนเนอร์และประสิทธิภาพการทำงานรวมไปถึงค่าดำเนินการที่เกิดขึ้น (Overhead)

บทที่ 3
วิธีการดำเนินวิจัย

จากปัญหาและทฤษฎีที่ได้กล่าวมาในข้างต้น ในบทนี้อธิบายถึงขั้นตอนดำเนินการออกแบบและวิธีการทดสอบประสิทธิภาพ เพื่อการประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันบนด็อกเกอร์คอนเทนเนอร์โดยมีรายละเอียดดังนี้

3.1 สภาพแวดล้อมการทดลอง

สภาพแวดล้อมในการทดลองแบ่งออกเป็น 3 ส่วนได้แก่ เครื่องเซิร์ฟเวอร์เป็นเครื่องหลักที่ติดตั้งแอปพลิเคชันสำหรับในการทดสอบ เครื่องโหลดเทสเป็นเครื่องสำหรับสร้างโหลดส่งไปยังเครื่องเซิร์ฟเวอร์ และเครื่องมอร์นิเตอร์เป็นเครื่องใช้ตรวจสอบข้อมูลการทำงานของเครื่องเซิร์ฟเวอร์เพื่อควบคุมปัจจัยที่ไม่เกี่ยวข้องในทำงานของด็อกเกอร์และแอปพลิเคชันในระหว่างการทดสอบ โดยทั้งหมดอยู่ภายใต้ระบบเน็ตเวิร์คเดียวกัน ในการทดลองสนใจการทำงานของเครื่องเซิร์ฟเวอร์ ซึ่งมีรายละเอียดดังนี้

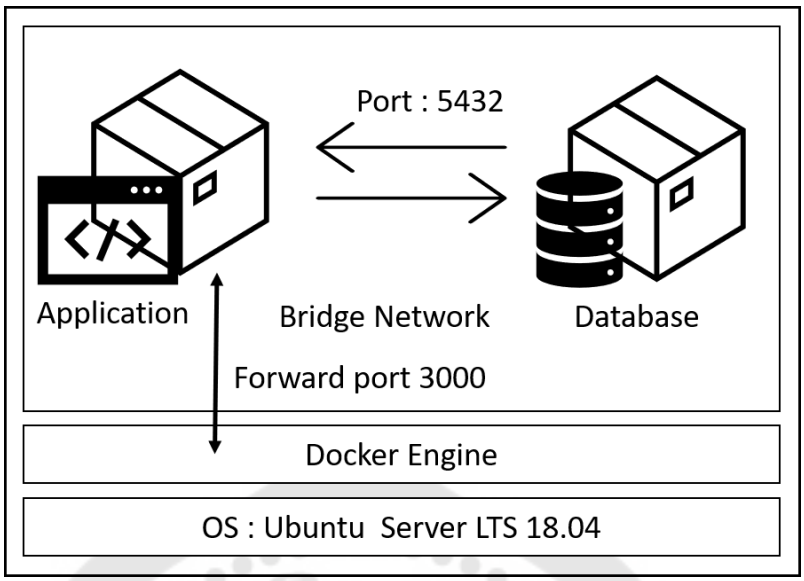
ตาราง 1 รายละเอียดเครื่องเซิร์ฟเวอร์

ทรัพยากร	รายละเอียด
CPU	AMD FX 6300 6 Core 6 Threads
RAM	DDR 3 BUS 1333 8 Gb
Storage	SSD 120 Gb
OS	Linux Server 18.04 LTS
Container Engine	Docker CE Version 19.03.12

3.2 การออกแบบสถานการณ์ในการทดลอง

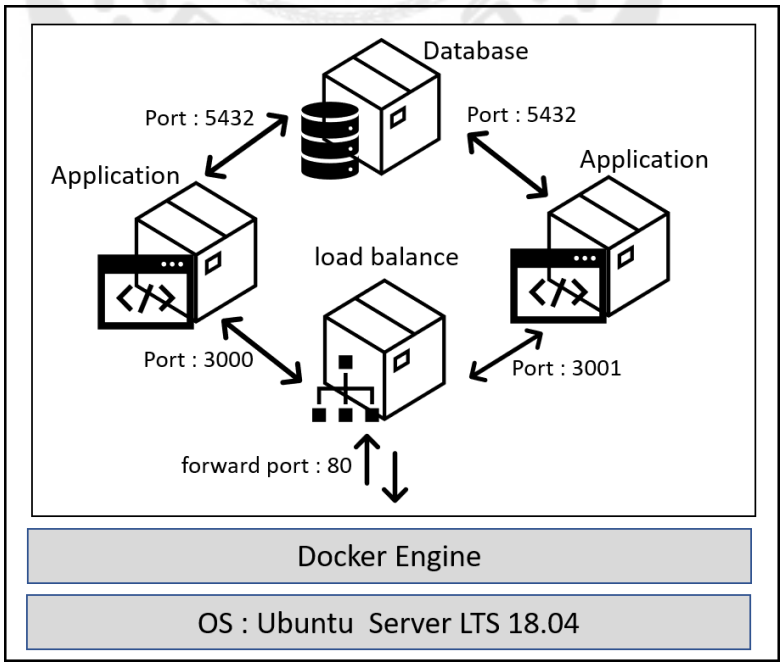
การออกแบบสถานการณ์ในการทดสอบประสิทธิภาพเพื่อประเมินสมรรถนะการทำงานของแอปพลิเคชันที่ทำงานบนด็อกเกอร์คอนเทนเนอร์ ผู้วิจัยได้ออกแบบสถานการณ์ที่แตกต่างกัน 3 สถานการณ์ ซึ่งมีรายละเอียดดังนี้

3.2.1 สถานการณ์ 1 ประกอบด้วยคอนเทนเนอร์ของแอปพลิเคชันและฐานข้อมูลอย่างละ 1 คอนเทนเนอร์ ดังภาพประกอบ 6



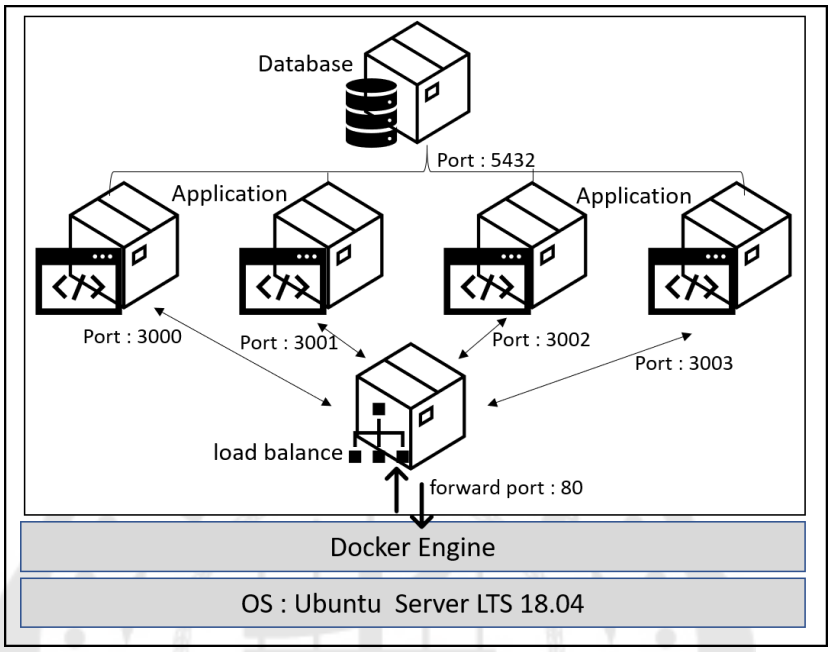
ภาพประกอบ 6 โครงสร้างของสถานการณ์ 1

3.2.2 สถานการณ์ 2 ประกอบด้วยคอนเทนเนอร์ทั้งหมดจำนวน 4 คอนเทนเนอร์ โดยเพิ่มคอนเทนเนอร์ของแอปพลิเคชันเป็น 2 คอนเทนเนอร์ เพื่อเพิ่มประสิทธิภาพการทำงานของแอปพลิเคชันและเพิ่มคอนเทนเนอร์ Load Balance สำหรับกระจายโหลดให้แอปพลิเคชัน แต่ฐานข้อมูลยังคงไว้ 1 คอนเทนเนอร์ ดังภาพประกอบ 7



ภาพประกอบ 7 โครงสร้างของสถานการณ์ 2

3.2.3 สถานการณ์ 3 ประกอบด้วยคอนเทนเนอร์ทั้งหมดจำนวน 6 คอนเทนเนอร์ โดยยังมีโครงสร้างเหมือนกับสถานการณ์ 2 แต่เพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชันเป็น 4 คอนเทนเนอร์ ดังภาพประกอบที่ 8

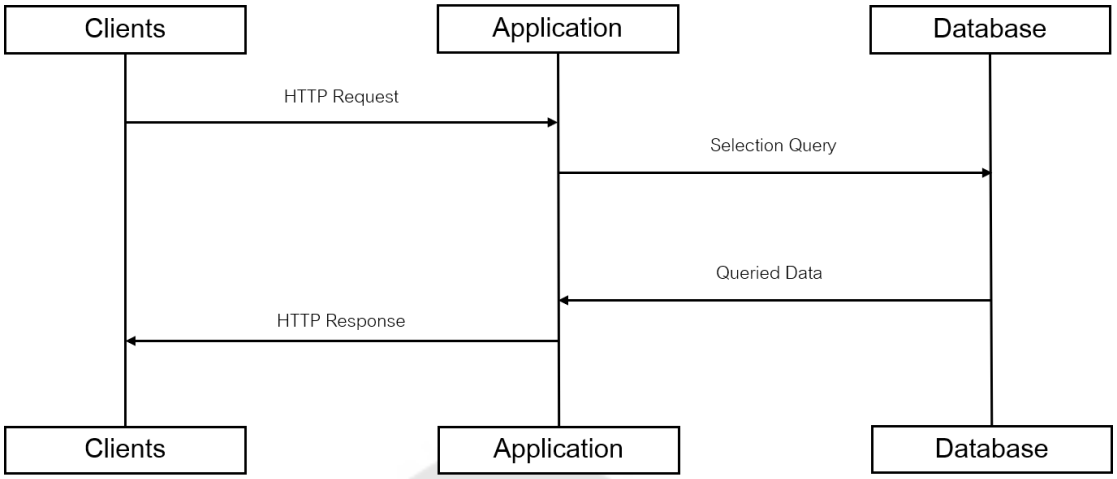


ภาพประกอบ 8 โครงสร้างของสถานการณ์ 3

จะเห็นได้ว่าในแต่ละสถานการณ์มีฐานข้อมูลเพียงหนึ่งคอนเทนเนอร์เนื่องจากฐานข้อมูลที่ใช้ทดสอบเป็น Standalone Instance ที่ทำงานและเก็บข้อมูลอย่างอิสระเพื่อป้องกันไม่ให้เกิดปัญหาความไม่สอดคล้องกันของข้อมูล โดยในสถานการณ์ 2 และ 3 เป็นการขยายจำนวนคอนเทนเนอร์แอปพลิเคชันเพื่อเพิ่มความสามารถในการรองรับโหลดและประสิทธิภาพประมวลผล ซึ่งในทุกสถานการณ์คอนเทนเนอร์จะไม่ถูกกำหนดปริมาณการใช้งานทรัพยากร

3.3 สภาพแวดล้อมแอปพลิเคชัน

สภาพแวดล้อมแอปพลิเคชันในการทดสอบเป็นการทำงานระหว่างแอปพลิเคชันกับฐานข้อมูล โดยแอปพลิเคชันจะสุ่มเรียกข้อมูลจำนวน 20 แถว จากทั้งหมด 10,000 แถว ดังในภาพประกอบ 9 ซึ่งแอปพลิเคชันถูกพัฒนาด้วย JavaScript โดยมี Node.js เวอร์ชัน 13.14 เป็น Run-Time และฐานข้อมูลใช้ของ PostgreSQL เวอร์ชัน 12.1 ส่วน Load Balance เลือกใช้ NGINX เวอร์ชัน 1.19 โดยใช้อัลกอริทึมในการกระจายงานคือ Round Robin



ภาพประกอบ 9 Application Flow

3.4 วิธีการทดสอบ

การทดสอบการทำงานของแอปพลิเคชันใช้วิธีการทดสอบด้วยการโหลดเทส(Load Test) โดยใช้ K6 ซึ่งออกแบบการทดสอบด้วย JavaScript การออกแบบโหลดเทสในการทดสอบเป็นการจำลองลักษณะการเข้าใช้งานที่มีอัตราเพิ่มขึ้นอย่าต่อเนื่องของผู้ใช้ ซึ่งแบ่งออกเป็น 3 ลักษณะดังนี้

3.4.1 TestCase 1 กำหนดให้อัตราการส่ง Request เริ่มต้นที่ 10 Requests/sec และในทุกๆ 10 วินาที จะมีอัตราการส่งที่เพิ่มขึ้น 10 Requests/sec จนกระทั่งถึงอัตราส่งที่ 100 Requests/sec และคงไว้ที่อัตราดังกล่าวเป็นเวลา 10 นาที

3.4.2 TestCase 2 กำหนดให้อัตราการส่ง Request เริ่มต้นที่ 10 Requests/sec และในทุกๆ 10 วินาที จะมีอัตราการส่งที่เพิ่มขึ้น 10 Requests/sec จนกระทั่งถึงอัตราส่ง 200 Requests/sec และคงไว้ที่อัตราดังกล่าวเป็นเวลา 10 นาที

3.4.3 TestCase 3 กำหนดให้อัตราการส่ง Request เริ่มต้นที่ 10 Requests/sec และในทุกๆ 10 วินาที จะมีอัตราการส่งที่เพิ่มขึ้น 10 Requests/sec จนกระทั่งถึงอัตราส่ง 300 Requests/sec และคงไว้ที่อัตราดังกล่าวเป็นเวลา 10 นาที

ภาพประกอบ 10 แสดงโคดการออกแบบโหลดเทสด้วย JavaScript โดย import เป็นการเรียกใช้ฟังก์ชันในการทำงาน ส่วน export let เป็นการกำหนดเงื่อนไขในการสร้างโหลดเทส ซึ่งถูกกำหนดใน stages โดย duration คือกำหนดระยะเวลาในการส่งโหลดเทส target คือการกำหนดจำนวนการส่ง Requests และ export default function การนำค่าจาก export let ไปดำเนินการกับปลายทางที่กำหนดไว้ใน const

```
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,
  })
}
```

ภาพประกอบ 10 การออกแบบโหลดเทส TestCase 1

3.5 มาตรวัดประสิทธิภาพ

การวัดประสิทธิภาพการทำงานของเซิร์ฟเวอร์จะทำการทดสอบด้วย 3 TestCase ที่กำหนดไว้ข้างต้นด้วย K6 โดยจะรายงานผลข้อมูลของโหลดเทสซึ่งมีรายละเอียดมาตรวัดดังนี้

3.5.1 Iteration คือ จำนวน request ทั้งหมดที่ส่งไปยังเซิร์ฟเวอร์

3.5.2 Response Time / sec คือ เวลาที่ เซิร์ฟเวอร์ ตอบสนองกลับต่อ Client โดยเริ่มนับเวลาตั้งแต่เริ่มส่ง request จนกระทั่งเซิร์ฟเวอร์ตอบกลับ

1. Max เวลามากที่สุดที่ใช้ในการตอบกลับ
2. Min เวลาน้อยที่สุดในการตอบกลับ
3. Avg เวลาเฉลี่ยในการตอบกลับ

3.5.3 Success คือ การตอบกลับของเซิร์ฟเวอร์ที่สำเร็จโดยวัดจาก HTTP response status codes 200 แสดงผลเป็นเปอร์เซ็นต์ ซึ่งในการตั้งค่าให้สนใจเฉพาะ status code 200 เท่านั้น นอกเหนือจากนั้นถือว่าเป็น Failure ทั้งหมด

3.5.4 Failure คือ การตอบกลับของเซิร์ฟเวอร์ที่ไม่สำเร็จแสดงผลเป็นเปอร์เซ็นต์

3.5.5 Throughput คือ ปริมาณงานที่เซิร์ฟเวอร์สามารถประมวลผลได้และตอบกลับในเวลา 1 วินาที

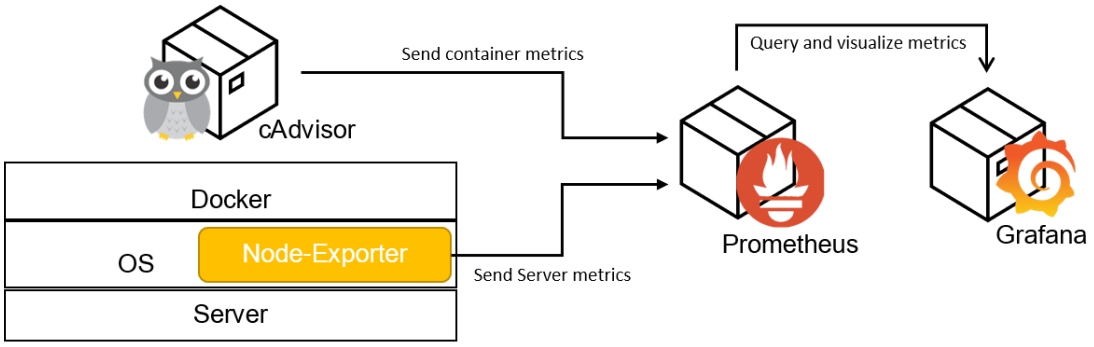
โดยในการทดสอบในแต่ละ TestCase จะดำเนินการทดสอบทั้งหมด 3 ครั้ง และนำข้อมูลที่ได้มาวิเคราะห์เพิ่มเติมเพื่อตรวจสอบประสิทธิภาพ (Waraporn, วราภรณ์, Srinakharinwirot University. Faculty of, Napawit, และ นภวิชญ์, 2020) ซึ่งในภาพประกอบ 11 เป็นหน้าแสดงผลลัพธ์จากโหลดเทสด้วย K6

```
         /\      |‾‾| /‾‾/   /‾‾/
    /\  /  \     |  |/  /   /  /
   /  \/    \    |     (   /   ‾\
  /          \   |  |\  \ |  (‾)  |
 / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: http_get_scen1_case1.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 12m10s max duration (incl. graceful stop):
           * default: Up to 100 looping VUs for 11m40s over 11 stages (gracefulRampDown: 30s, gracefulStop: 30s)


running (11m40.3s), 000/100 VUs, 127111 complete and 0 interrupted iterations
default ✓ [======================================] 000/100 VUs  11m40s

     ✓ succeeded

     checks.........................: 100.00% ✓ 127111 ✗ 0
     data_received..................: 335 MB  478 kB/s
     data_sent......................: 12 MB   18 kB/s
     http_req_blocked...............: avg=6.02µs   min=1.25µs  med=4.98µs   max=19.97ms  p(90)=6.51µs   p(95)=7.29µs
     http_req_connecting............: avg=408ns    min=0s      med=0s       max=19.85ms  p(90)=0s       p(95)=0s
     http_req_duration..............: avg=510.96ms min=7.54ms  med=538.04ms max=717.3ms  p(90)=596.22ms p(95)=611.34ms
       { expected_response:true }...: avg=510.96ms min=7.54ms  med=538.04ms max=717.3ms  p(90)=596.22ms p(95)=611.34ms
     http_req_failed................: 0.00%   ✓ 0       ✗ 127111
     http_req_receiving.............: avg=81.37µs  min=19.67µs med=70.41µs  max=9.52ms   p(90)=105.96µs p(95)=118.89µs
     http_req_sending...............: avg=28.17µs  min=6.59µs  med=24.07µs  max=6.92ms   p(90)=34µs     p(95)=38.71µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=510.85ms min=7.41ms  med=537.94ms max=717.19ms p(90)=596.12ms p(95)=611.21ms
     http_reqs......................: 127111  181.504984/s
     iteration_duration.............: avg=511.17ms min=7.76ms  med=538.25ms max=717.5ms  p(90)=596.44ms p(95)=611.54ms
     iterations.....................: 127111  181.504984/s
     vus............................: 100     min=1     max=100
     vus_max........................: 100     min=100   max=100
```

ภาพประกอบ 11 การแสดงผลการโหลดเทสด้วย K6

3.6 การตรวจสอบการใช้งานทรัพยากร

การตรวจสอบการใช้งานทรัพยากรของเซิร์ฟเวอร์ใช้ Node-Exporter ที่ติดตั้งอยู่บน OS และการตรวจสอบใช้งานทรัพยากรของคอนเทนเนอร์ใช้ cAdvisor ที่ติดตั้งอยู่บนคอนเทนเนอร์ ซึ่งข้อมูลการใช้งานทรัพยากรเซิร์ฟเวอร์และด็อกเกอร์คอนเทนเนอร์ จะถูกส่งไปยัง Prometheus เพื่อจัดเก็บและ Grafana เรียกข้อมูลมาแสดงผล (Visualize) ดังภาพประกอบ 12



ภาพประกอบ 12 สภาพแวดล้อมการตรวจสอบทรัพยากรเซิร์ฟเวอร์และคอนเทนเนอร์

3.6.1. มาตรวัดการตรวจสอบการใช้งานทรัพยากรเซิร์ฟเวอร์

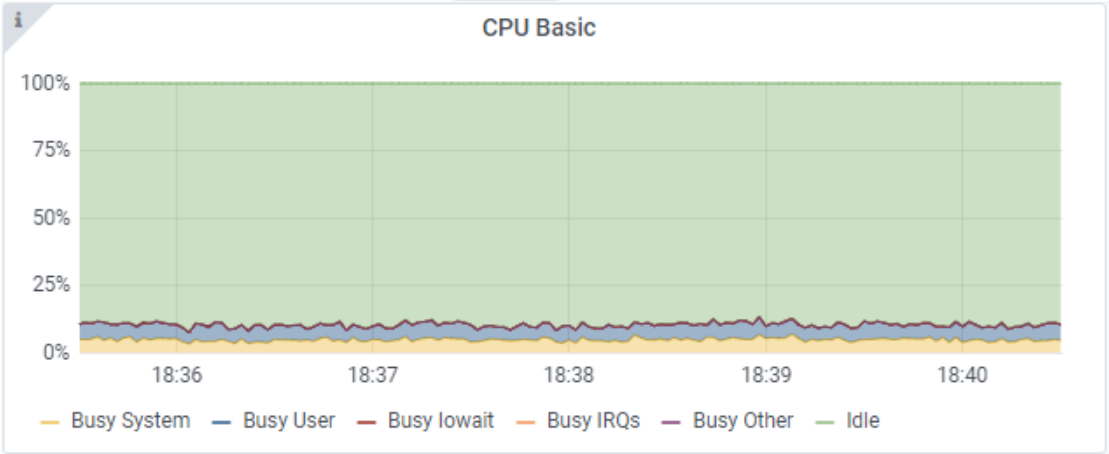
มาตรวัดการตรวจสอบการใช้งานทรัพยากรเซิร์ฟเวอร์ที่ได้จาก Grafana โดยบันทึกข้อมูลการใช้งานทรัพยากรในช่วงเวลาตั้งแต่เริ่มการทดสอบด้วยโหลดเทสจนจบการทดสอบในแต่ละ TestCase ซึ่งมาตรวัดที่ใช้ในการตรวจสอบการใช้งานทรัพยากรเซิร์ฟเวอร์ เพื่อนำมาวิเคราะห์มีดังนี้

3.6.1.1. มาตรวัดการใช้งานของ CPU

1. Busy System อธิบายอัตราการใช้งานทรัพยากร CPU ของระบบปฏิบัติการหรือ Kernel

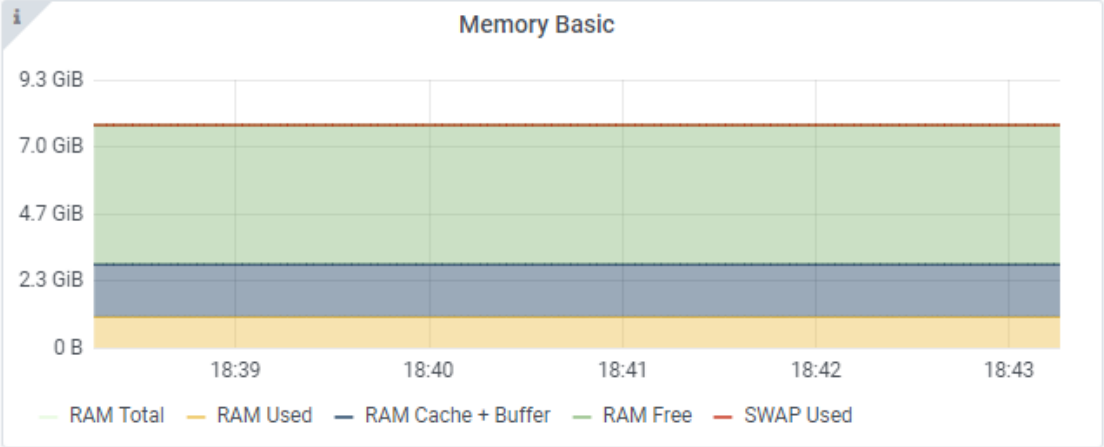
2. Busy User อธิบายอัตราการใช้งานทรัพยากร CPU โดยบัญชีผู้ใช้บนระบบปฏิบัติการ

3. Busy IRQs อธิบายการใช้งาน CPU ในการจัดการ Interrupt การขัดจังหวะการทำงานของงานที่ทำอยู่ดังภาพประกอบ 13



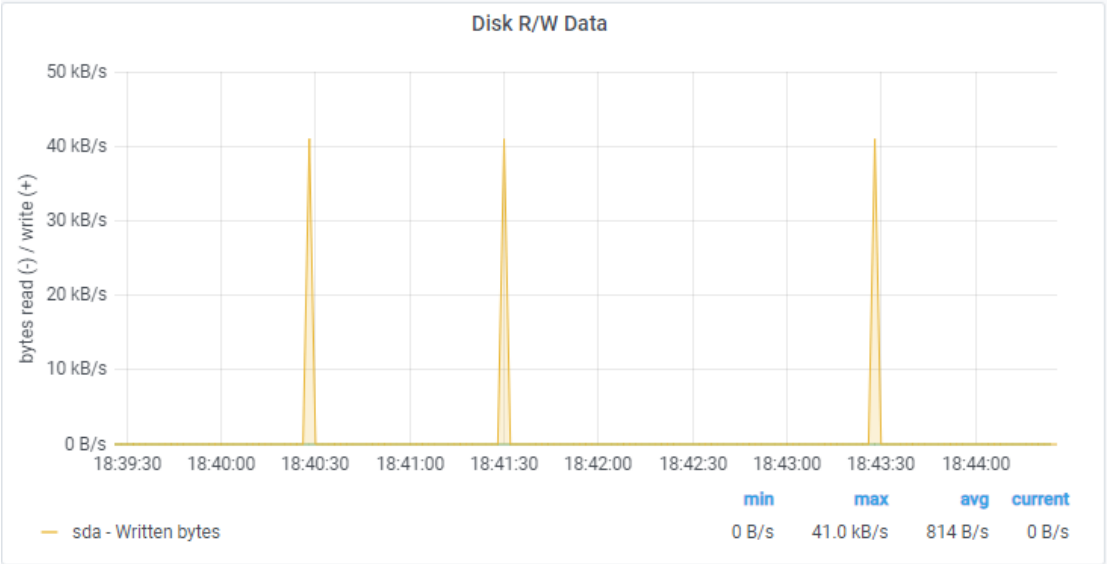
ภาพประกอบ 13 Grafana Dashboard CPU Basic

3.6.1.2. มาตรวัดการใช้งานของ RAM ใช้ข้อมูลของ RAM Used อธิบายปริมาณการใช้งาน RAM ทั้งหมดดังภาพประกอบ 14



ภาพประกอบ 14 Grafana Dashboard Memory Basic

3.6.1.3. มาตรวัดการใช้งานของ Disk ใช้ข้อมูล sda Written bytes อธิบายปริมาณการเขียนข้อมูลและ sda Read bytes อธิบายการอ่านข้อมูลของ Disk ภาพประกอบ15

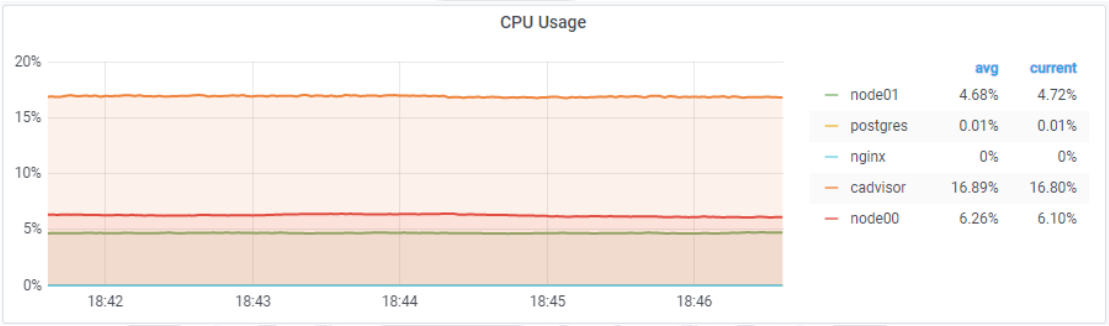


ภาพประกอบ 15 Grafana Dashboard Disk R/W Data

3.6.2. มาตรวัดการตรวจสอบการใช้งานทรัพยากรคอนเทนเนอร์

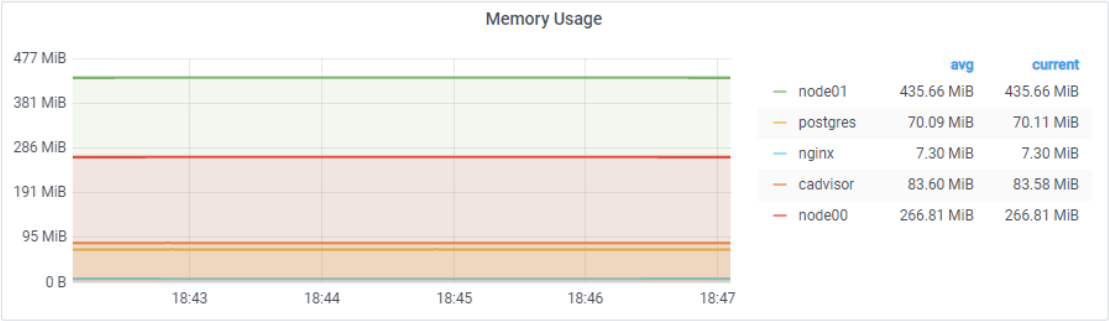
การตรวจสอบการใช้งานทรัพยากรคอนเทนเนอร์ที่ได้จาก Grafana โดยบันทึกข้อมูลการใช้งานทรัพยากรในช่วงเวลาตั้งแต่เริ่มการทดสอบด้วยโหลดเทสจนจบการทดสอบในแต่ละ TestCase ซึ่งมาตรวัดที่ใช้ในการตรวจสอบการใช้งานทรัพยากรของคอนเทนเนอร์เพื่อนำมาวิเคราะห์มีดังนี้

3.6.2.1 มาตรวัด CPU ใช้ข้อมูลจาก CPU Usage ซึ่งอธิบายอัตราการใช้งาน CPU ของแต่ละคอนเทนเนอร์ ดังภาพประกอบ16



ภาพประกอบ 16 Grafana Dashboard CPU Usage

3.6.2.2 มาตรวัด RAM ใช้ข้อมูลจาก Memory Usage อธิบายปริมาณการใช้งาน RAM ของแต่ละคอนเทนเนอร์ดังภาพประกอบ 17



ภาพประกอบ 17 Grafana Dashboard Memory Usage

3.6.2.3 มาตรวัด Disk ใช้ข้อมูลจาก I/O Rx อธิบายปริมาณการอ่านข้อมูล และ I/O Tx อธิบายปริมาณการเขียนข้อมูลของแต่ละคอนเทนเนอร์ดังภาพประกอบ 18

ภาพประกอบ 18 Grafana Dashboard Disk I/O Rx Tx

บทที่ 4
ผลการศึกษาและอภิปลายผล

งานวิจัยการประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันบนด็อกเกอร์คอนเทนเนอร์ โดยแบ่งออกเป็น 3 สถานการณ์และทดสอบด้วยโหลดเทส โดยข้อมูลที่แสดงผลการใช้งานทรัพยากรของคอนเทนเนอร์จะแสดงข้อมูลการทำงานของทุกคอนเทนเนอร์ในแต่ละสถานการณ์ โดย App00, App01, App02 และ App03 หมายถึงลำดับคอนเทนเนอร์แอปพลิเคชัน ส่วน Database หมายถึงคอนเทนเนอร์ฐานข้อมูล

4.1 สถานการณ์จำลอง 1

4.1.1 ประสิทธิภาพของสถานการณ์ 1

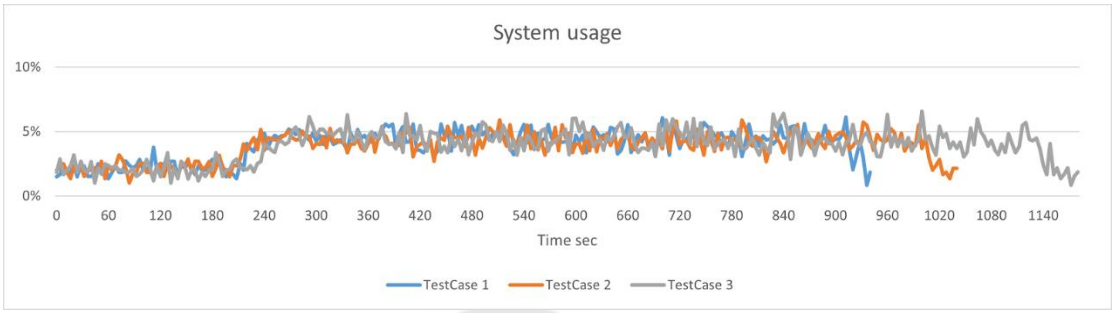
ผลการทดสอบโหลดเทสปรากฏว่า แอปพลิเคชันสามารถประมวลผล Workload ได้สำเร็จทั้งหมดซึ่งดูจากในตาราง 2 โดยมีอัตรา Success% ที่ 100 ในทุก TestCase และมีอัตรา Throughput และเวลาเฉลี่ยในการประมวลผลแตกต่างกันในแต่ละ TestCase ตามการเพิ่มขึ้นตามจำนวน iteration อย่างสัมพันธ์กัน ซึ่งผลการทดสอบปรากฏดังตาราง 2

ตาราง 2 ผลการทดสอบด้วยโหลดเทสของสถานการณ์ 1

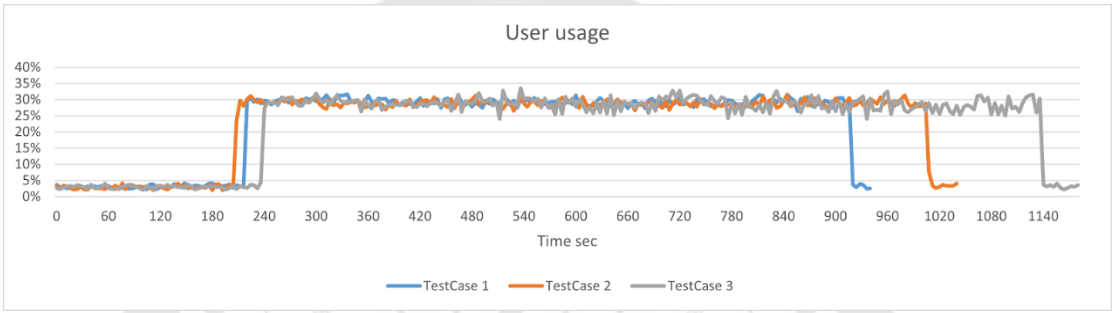
Case	iteration	Response Time (s)			Success %	Failure%	Throughput /s
		Max	Min	Avg			
TestCase 1	127245	0.692	0.008	0.511	100	0	181.683
TestCase 2	146168	1.437	0.008	0.958	100	0	182.600
TestCase 3	166092	2.053	0.008	1.350	100	0	184.399

4.1.2 การใช้งานทรัพยากรเครื่องเซิร์ฟเวอร์

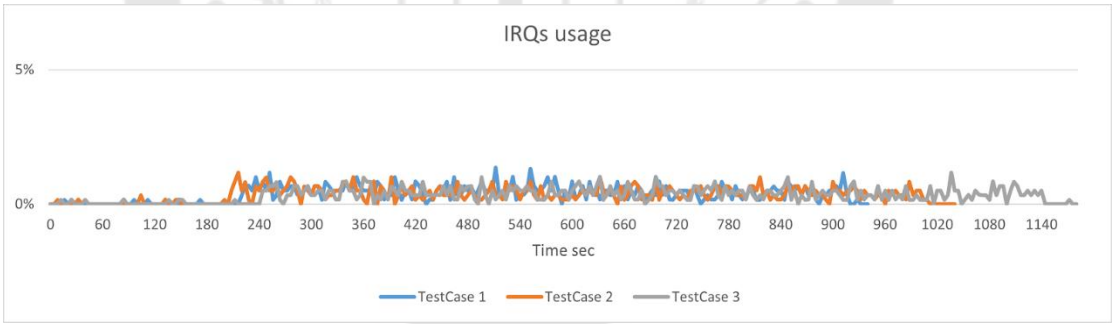
4.1.2.1 CPU



(a)



(b)

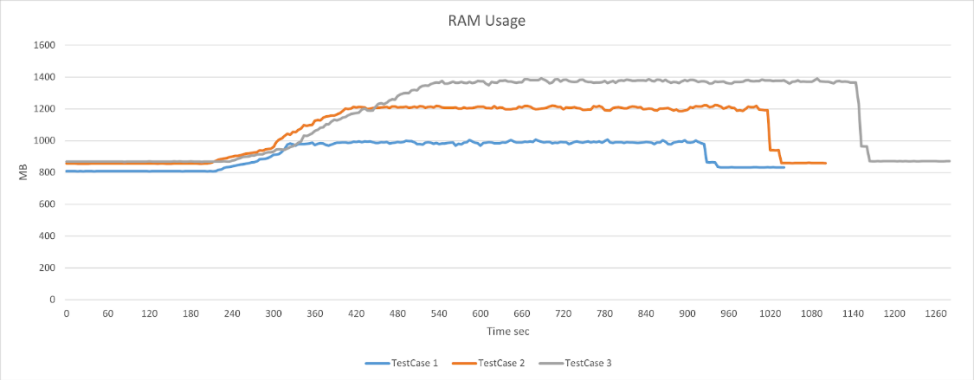


(c)

ภาพประกอบ 19 การใช้งาน CPU บนเซิร์ฟเวอร์ในสถานการณ์ 1

จากภาพประกอบ 19(a), 19(b) และ 19(c) แสดงการใช้งาน CPU ของสถานการณ์ 1 ที่ได้จากการวัดการใช้ CPU ที่ System, IRQs, และ User ตามลำดับ พบว่า อัตราการใช้งาน CPU ใช้ช่วงสูงสุดของแต่ละ Testcase ไม่พบความแตกต่าง แม้ว่ามีปริมาณงานที่เพิ่มขึ้น ซึ่ง User เรียกใช้งาน CPU มากที่สุด โดยในช่วงสิ้นสุดการทำงานของแอปพลิเคชัน TestCase 3 มีระยะเวลาการใช้งาน CPU นานที่สุด

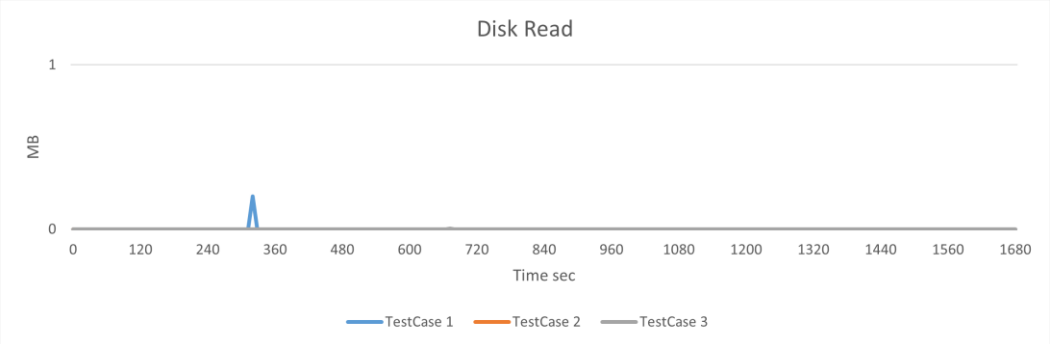
4.1.2.2 RAM



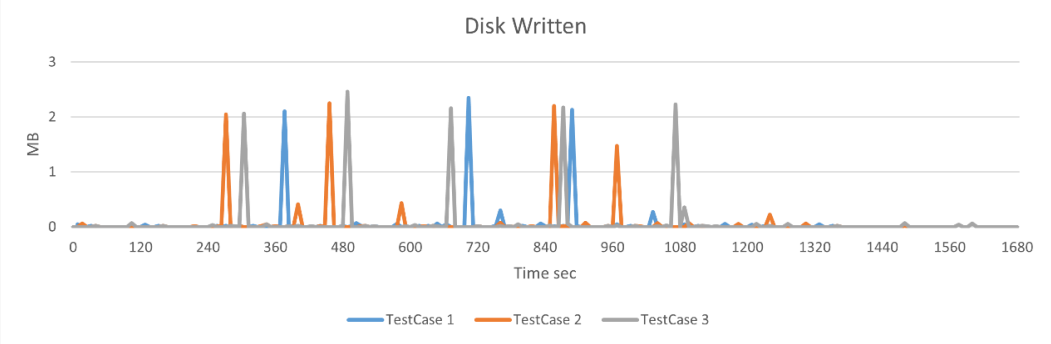
ภาพประกอบ 20 การใช้งาน RAM บนเซิร์ฟเวอร์ในสถานการณ์ 1

ภาพประกอบ 20 แสดงให้เห็นปริมาณการใช้งาน RAM ของเครื่องเซิร์ฟเวอร์ที่มีปริมาณเพิ่มขึ้นตามจำนวน Workload โดย TestCase 3 มีปริมาณการใช้งาน RAM มากที่สุดและระยะเวลาใช้งานยาวนานที่สุด

4.1.2.3 Disk



(a)



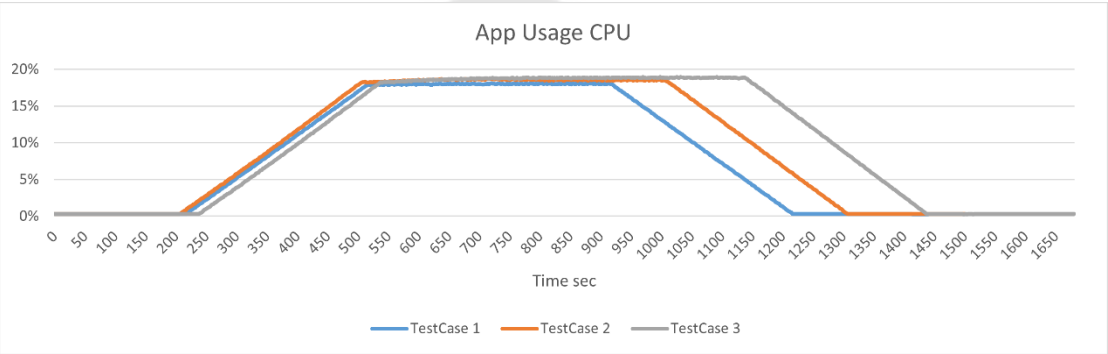
(b)

ภาพประกอบ 21 การใช้งาน Disk บนเซิร์ฟเวอร์ในสถานการณ์ 1

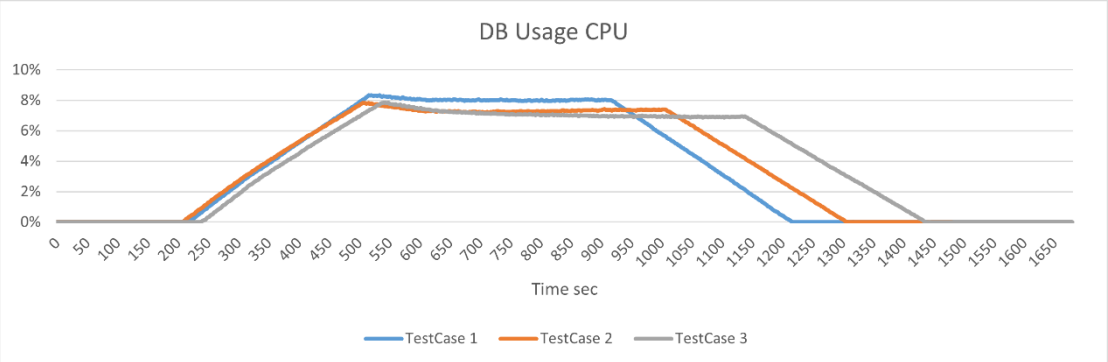
จากภาพประกอบ 21(a) แสดงปริมาณการอ่านข้อมูล และ 21(b) แสดงปริมาณการเขียนข้อมูลของ Disk ซึ่งการอ่านข้อมูลช่วงเกิด Workload มีการอ่านข้อมูลเพียงเล็กน้อยที่ TestCase 1 เท่านั้น ส่วนการเขียนข้อมูลในขณะที่เกิด Workload มีการเขียนข้อมูลเป็นเพียงบางช่วงเวลา ซึ่งเกิดจากการบันทึก Logfile ของแอปพลิเคชันในทุก TestCase อย่างไรก็ตามปริมาณการเขียนข้อมูลที่เกิดขึ้นนั้นมีปริมาณที่ใกล้เคียงกัน

4.1.3 การใช้งานทรัพยากรของด็อกเกอร์คอนเทนเนอร์

4.1.3.1 CPU



(a)



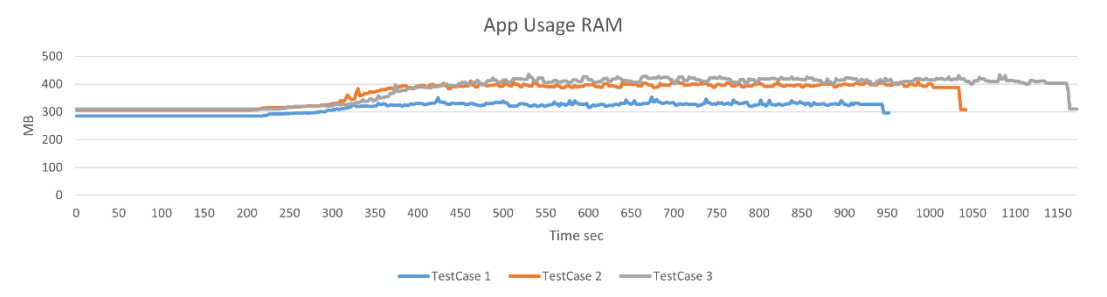
(b)

ภาพประกอบ 22 แสดงการใช้งาน CPU ของด็อกเกอร์คอนเทนเนอร์สถานการณ์ 1

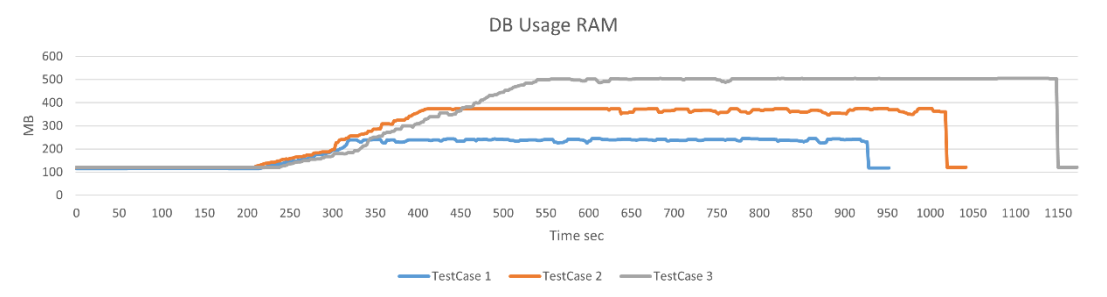
ภาพประกอบ 22(a) และ 22(b) แสดงอัตราการใช้งาน CPU ของคอนเทนเนอร์แอปพลิเคชันและฐานข้อมูล ซึ่งอัตราการใช้งาน CPU ของทั้ง 2 คอนเทนเนอร์แอปพลิเคชันเพิ่มขึ้นอย่างต่อเนื่องตามอัตรา Workload จนกระทั่งถึงอัตรากำลังส่งคงที่ คอนเทนเนอร์แอปพลิเคชัน

ยังคงระดับการใช้งาน CPU ทั้ง 3 TestCase แต่คอนเทนเนอร์ฐานข้อมูลมีอัตราการใช้งานลดลงอย่างต่อเนื่อง

4.1.3.2 RAM



(a)

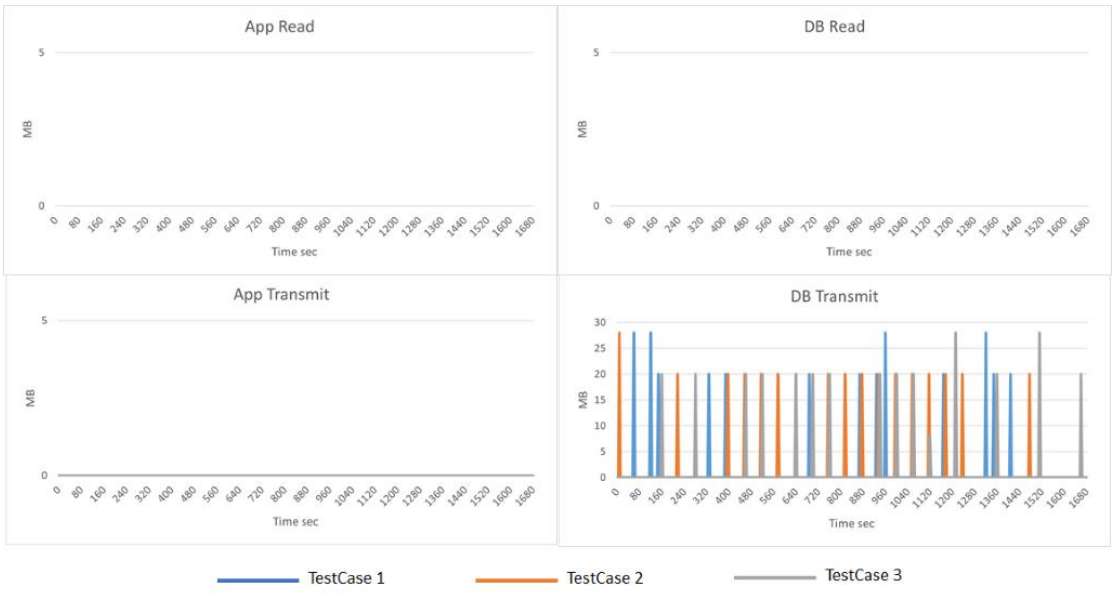


(b)

ภาพประกอบ 23 แสดงปริมาณการใช้งาน RAM ของคอนเทนเนอร์สถานการณ์ 1

ภาพประกอบ 23(a) และ 23(b) แสดงปริมาณการใช้งาน RAM ของคอนเทนเนอร์แอปพลิเคชันและฐานข้อมูล ซึ่งมีปริมาณที่เพิ่มขึ้นอย่างต่อเนื่องตามอัตรา Workload จนกระทั่งถึงอัตรากำลังส่งคงที่ โดยคอนเทนเนอร์แอปพลิเคชันยังคงระดับการใช้งาน RAM จนจบการทดสอบ ซึ่ง Testcase 2 และ 3 มีปริมาณใกล้เคียงกัน แต่ฐานข้อมูลมีการใช้งานเพิ่มขึ้นสัมพันธ์กับอัตรา Workload

4.1.3.3 Disk



ภาพประกอบ 24 แสดงการทำงาน Disk ของคอนเทนเนอร์สถานการณ์ 1

ภาพประกอบ 24 แสดงอัตราการใช้งาน Disk I/O ของคอนเทนเนอร์แอปพลิเคชันและคอนเทนเนอร์ฐานข้อมูล การใช้งานเกิดขึ้นที่คอนเทนเนอร์ฐานข้อมูลเท่านั้น โดยเป็นการส่งออกข้อมูลเป็นบางช่วงเวลา

4.2 **สถานการณ์จำลอง** 2

4.2.1 **ประสิทธิภาพของสถานการณ์** 2

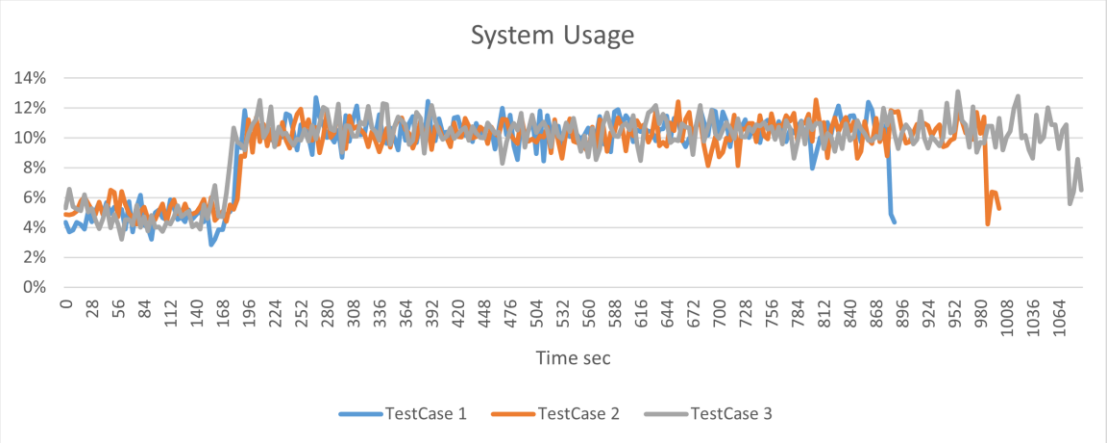
ผลการทดสอบโหลดเทสปรากฏว่า แอปพลิเคชันสามารถประมวลผล Workload ได้สำเร็จทั้งหมด โดยมีอัตรา Success% ที่ 100 :ซึ่งมีอัตรา Throughput ที่ลดลง แต่เวลาเฉลี่ยในการประมวลผลเพิ่มขึ้นตามการเพิ่มขึ้นของจำนวน iteration อย่างสัมพันธ์กัน ผลการทดสอบปรากฏดังตาราง 3

ตาราง 3 ผลการทดสอบโหลดเทสสถานการณ์ 2

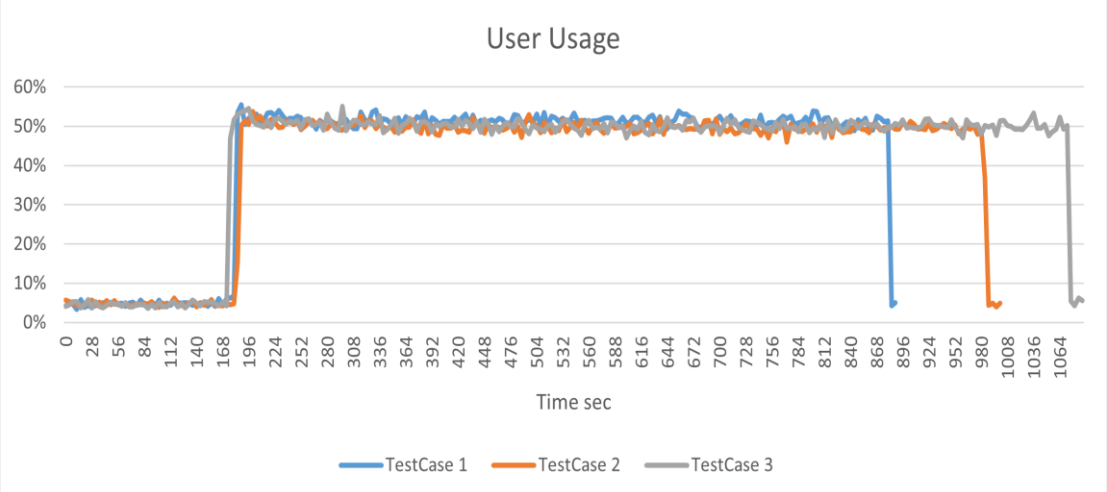
Case	iteration	Response Time (s)			Success %	Failure%	Throughput /s
		Max	Min	Avg			
TestCase 1	212980	0.756	0.008	0.307	100	0	304.130
TestCase 2	241407	1.437	0.007	0.580	100	0	301.501
TestCase 3	269290	2.193	0.007	0.836	100	0	298.880

4.2.2 การใช้งานทรัพยากรเครื่องเซิร์ฟเวอร์

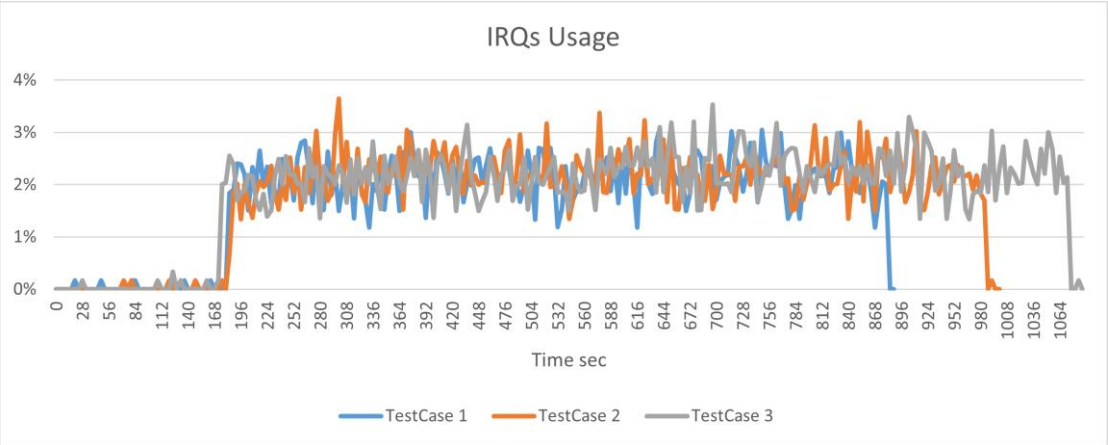
4.2.2.1 CPU



(a)



(b)

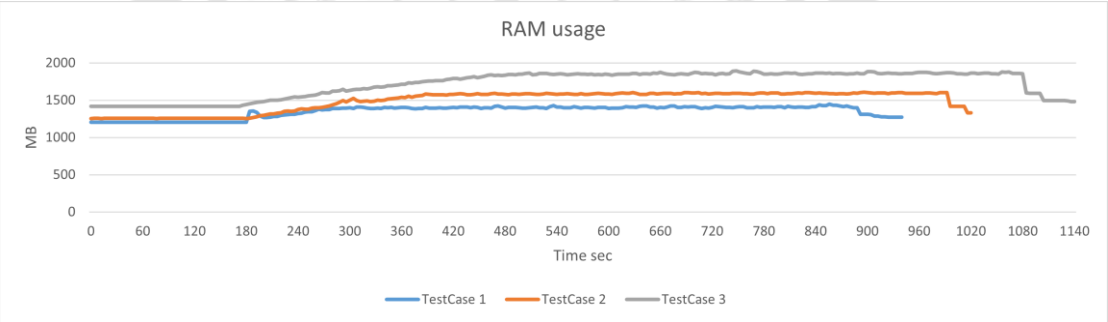


(c)

ภาพประกอบ 25 การใช้งาน CPU บนเซิร์ฟเวอร์ในสถานการณ์ 2

ภาพประกอบ 25(a), 25(b) และ 25(c) พบว่า อัตราการใช้งาน CPU ในทุก TestCase มีความใกล้เคียงกันถึงแม้จะมีปริมาณ Workload ที่ต่างกัน โดย System อยู่ที่ 10 – 12% User อยู่ที่ 50 – 52% และ IRQs อยู่ที่ 2 – 3% ซึ่งมี

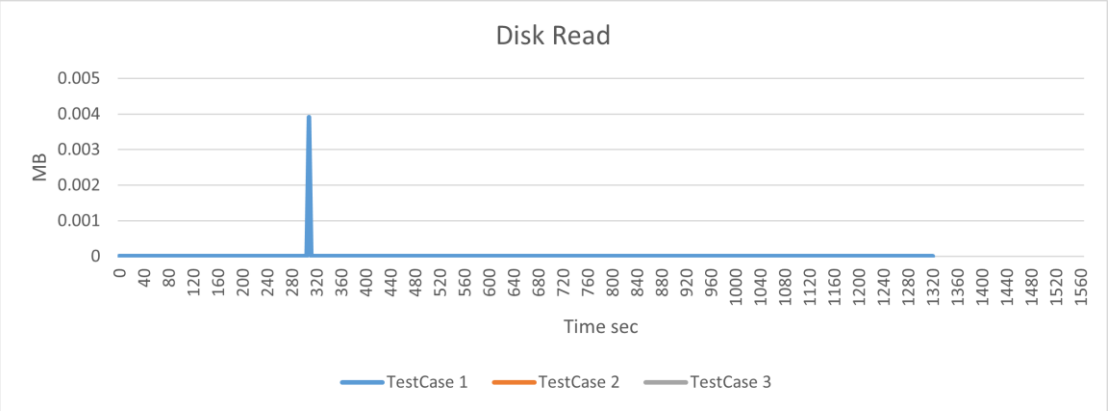
4.2.2.2 RAM



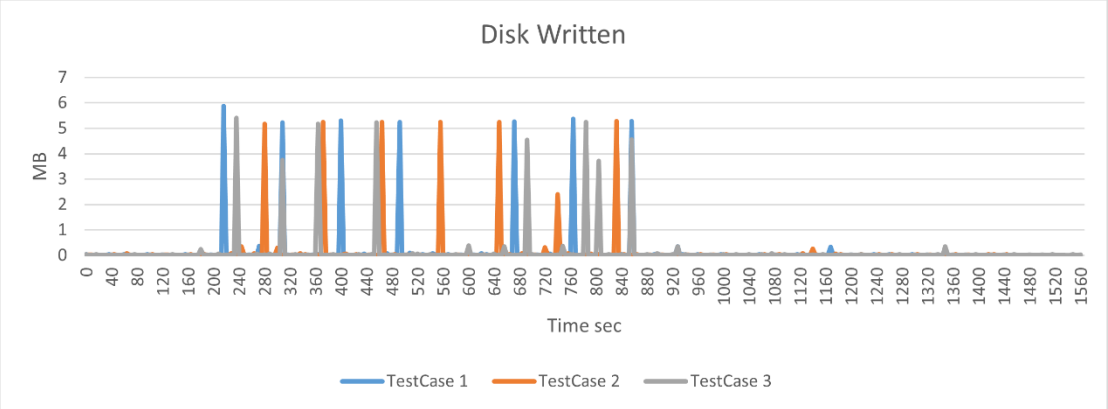
ภาพประกอบ 26 การใช้งาน RAM บนเซิร์ฟเวอร์ในสถานการณ์ 2

ภาพประกอบ 26 แสดงปริมาณการใช้งาน RAM ของเครื่องเซิร์ฟเวอร์ที่มีปริมาณเพิ่มขึ้นตามจำนวน Workload โดย TestCase 3 มีปริมาณการใช้งาน RAM มากที่สุดและระยะเวลาใช้งานยาวนานที่สุด

4.2.2.3 Disk



(a)



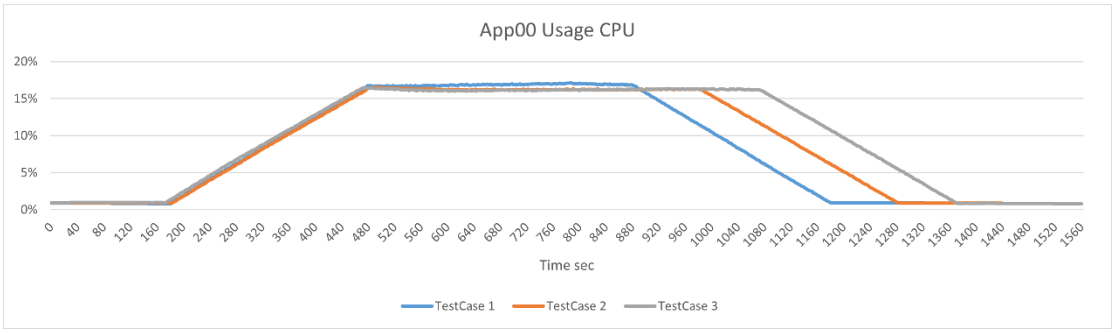
(b)

ภาพประกอบ 27 การใช้งาน Disk บนเซิร์ฟเวอร์ในสถานการณ์ 2

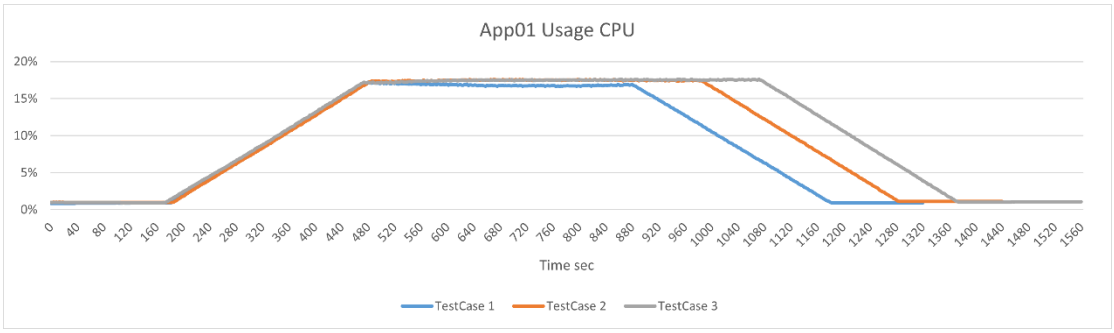
ภาพประกอบ 27(a) แสดงปริมาณการอ่านข้อมูล ซึ่งการอ่านข้อมูลช่วงเกิด Workload มีการอ่านข้อมูลเพียงเล็กน้อยที่ TestCase 1 เท่านั้น และ 27(b) แสดงปริมาณการเขียนข้อมูลในขณะที่เกิด Workload ซึ่งมีการเขียนข้อมูลเป็นบางช่วงเวลาที่เกิดจากการบันทึก log file ของแอปพลิเคชันในทุก TestCase อย่างไรก็ตาม ปริมาณการเขียนข้อมูลที่เกิดขึ้นนั้นมีปริมาณที่ใกล้เคียงกัน

4.2.3 การใช้งานทรัพยากรของคอนเทนเนอร์ในสถานการณ์ 2

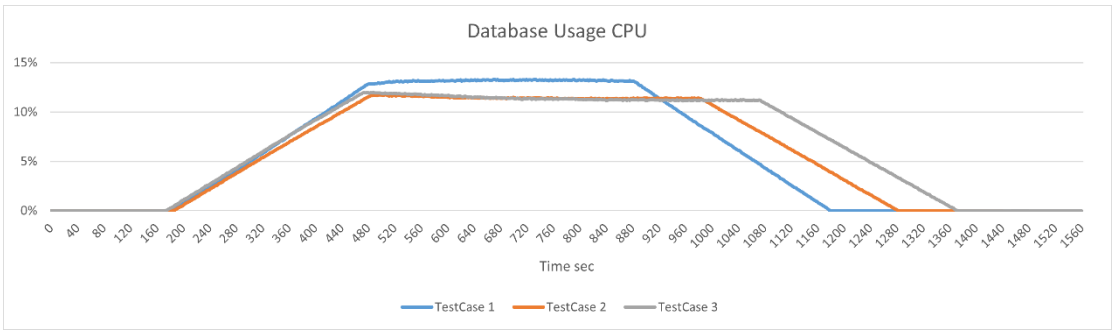
4.2.3.1 CPU



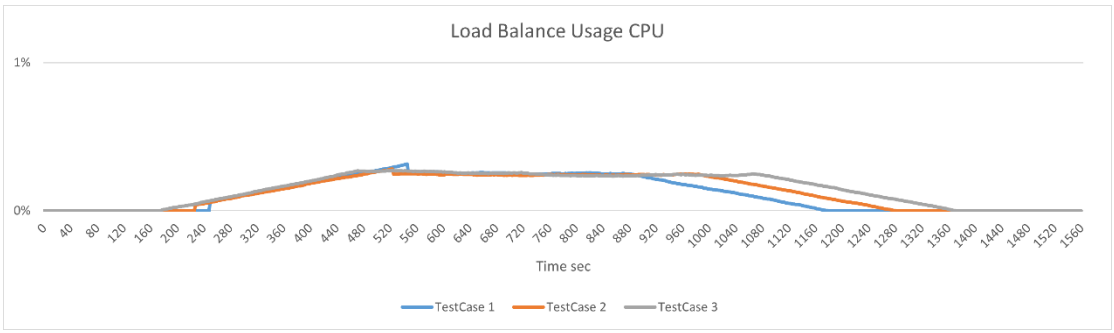
(a)



(b)



(c)



(d)

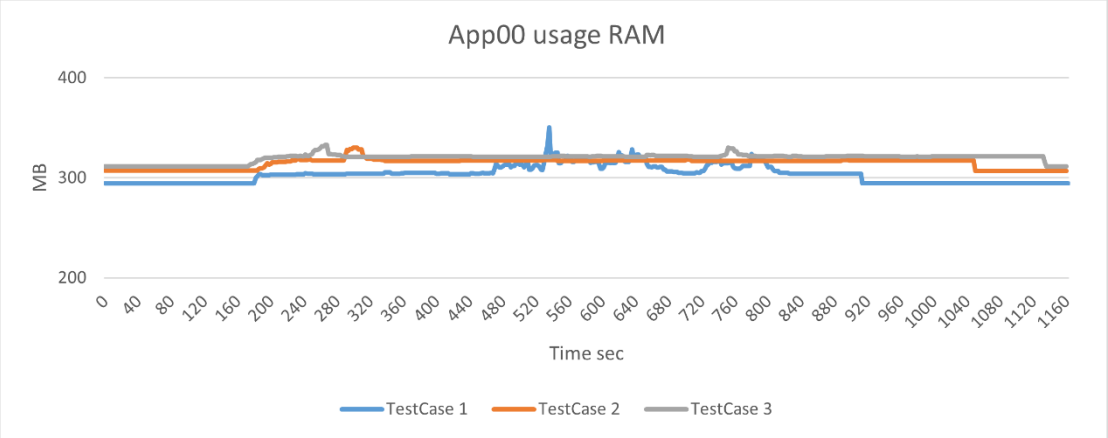
ภาพประกอบ 28 การใช้งาน CPU ของคอนเทนเนอร์ในสถานการณ์ 2

ภาพประกอบ 28(a) และ 28(b) แสดงการใช้งาน CPU ของคอนเทนเนอร์แอปพลิเคชันทั้ง 2 คอนเทนเนอร์ โดยมีอัตราการใช้งาน CPU เพิ่มขึ้นอย่างต่อเนื่องและใกล้เคียงกันในทุก TestCase แม้จะมี Workload ที่แตกต่างกัน

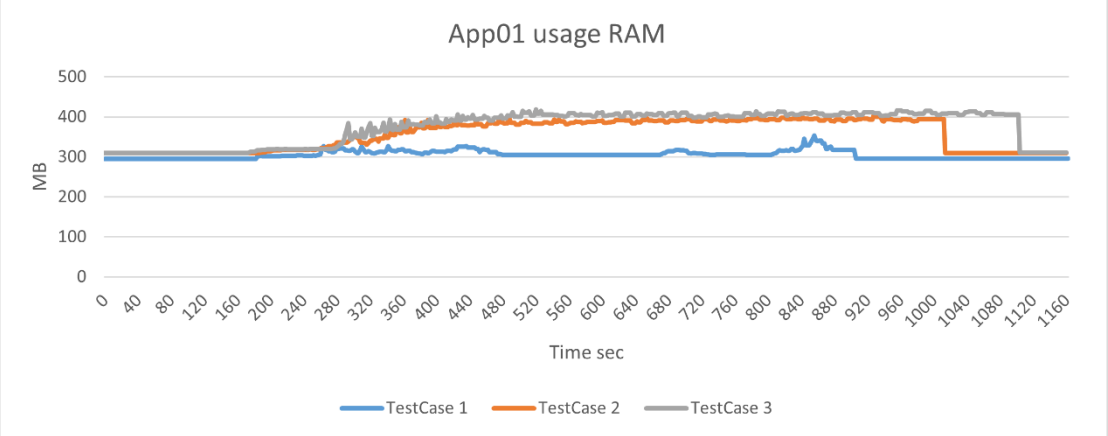
ภาพประกอบ 28(c) แสดงการใช้งาน CPU ของคอนเทนเนอร์ฐานข้อมูล โดยใน TestCase 1 มีการใช้งาน CPU มากที่สุด แต่ใน TestCase 2 และ 3 มีอัตราการใช้งานใกล้เคียงกันและลดลงอย่างต่อเนื่อง

ภาพประกอบ 28(d) แสดงการใช้งาน CPU ของคอนเทนเนอร์ Load Balance โดยในทุก TestCase มีอัตราการใช้งานต่ำ

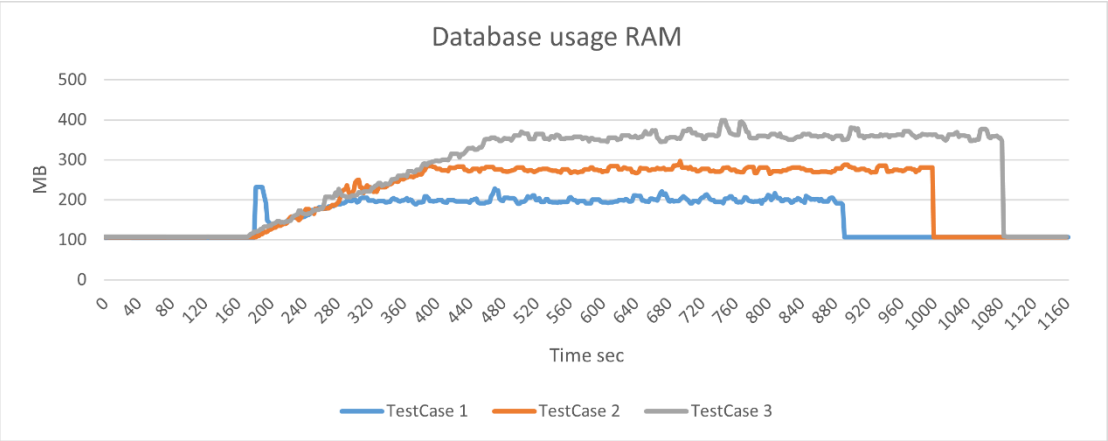
4.2.3.2 RAM



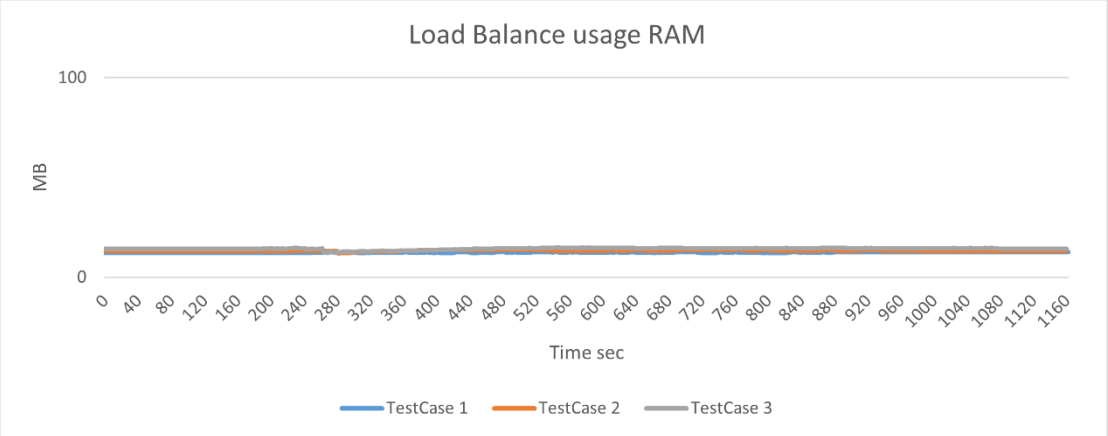
(a)



(b)



(c)



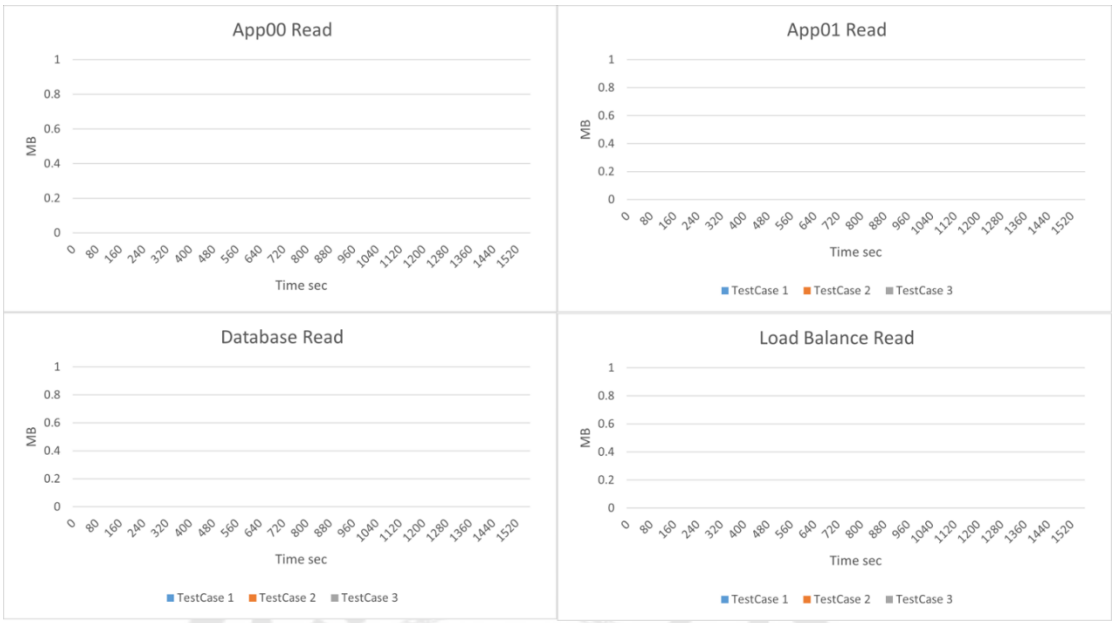
(d)

ภาพประกอบ 29 การใช้งาน RAM ของคอนเทนเนอร์ในสถานการณ์ 2

ภาพประกอบ 29(a) และ 29(b) แสดงอัตราการใช้งาน RAM ของคอนเทนเนอร์แอปพลิเคชัน พบว่าคอนเทนเนอร์ App00 มีอัตราการใช้งานไม่คงที่ในแต่ละ TestCase และมีอัตราการใช้งานเพียงเล็กน้อย ส่วนคอนเทนเนอร์ App01 มีอัตราการใช้งานเพิ่มขึ้น แต่ปริมาณการใช้งานของ TestCase 2 และ 3 มีระดับใกล้เคียงกัน

ภาพประกอบ 29(c) แสดงอัตราการใช้งาน RAM ของคอนเทนเนอร์ฐานข้อมูล ซึ่งการใช้งานมีอัตราที่เพิ่มและ TestCase 3 มีอัตราการใช้งานมากที่สุด แต่การใช้งาน RAM ในคอนเทนเนอร์ Load Balance มีอัตราการใช้งานที่ต่ำมากดังภาพประกอบ 29(d)

4.2.3.3 Disk



ภาพประกอบ 30 การใช้งาน Disk Read ของคอนเทนเนอร์ในสถานการณ์ 2

ภาพประกอบ 30 แสดงการอ่านข้อมูลของแอปพลิเคชัน ฐานข้อมูล และ Load Balance ซึ่งไม่พบการอ่านข้อมูลจาก Disk ในทุก TestCase ระหว่างการทดสอบ

ภาพประกอบ 31 การใช้งาน Disk Transmit ของคอนเทนเนอร์ในสถานการณ์ 2

ภาพประกอบ 31 แสดงการส่งข้อมูลของคอนเทนเนอร์ โดยคอนเทนเนอร์แอปพลิเคชันและ Load Balance ไม่พบการส่งข้อมูลในทุก TestCase แต่คอนเทนเนอร์ฐานข้อมูลมีการส่งข้อมูลเป็นบางช่วงเวลาด้วยปริมาณที่ใกล้เคียงกันในทุก TestCase

4.3 สถานการณ์จำลอง 3

4.3.1 ประสิทธิภาพของสถานการณ์ 3

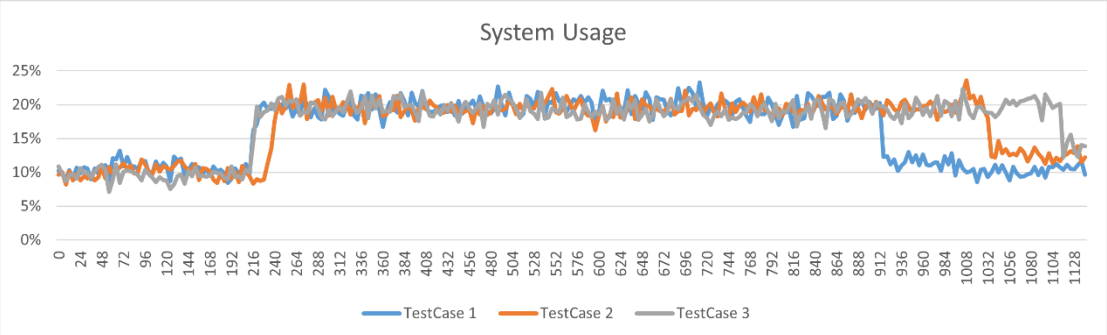
ผลการทดสอบโหลดเทสพบว่า แอปพลิเคชันสามารถประมวลผล Workload ได้สำเร็จทั้งหมด โดยมี Success% ที่ 100 ซึ่งมีอัตรา Throughput ที่เพิ่มขึ้น แต่ใน TestCase 2 และ 3 มีอัตรา Throughput แตกต่างกันเพียงเล็กน้อย ซึ่งเวลาเฉลี่ยในการประมวลผลเพิ่มขึ้นตามการเพิ่มขึ้นของจำนวน iteration อย่างสัมพันธ์กัน ผลการทดสอบปรากฏดังตาราง 4

ตาราง 4 ผลการทดสอบโหลดเทสสถานการณ์ 3

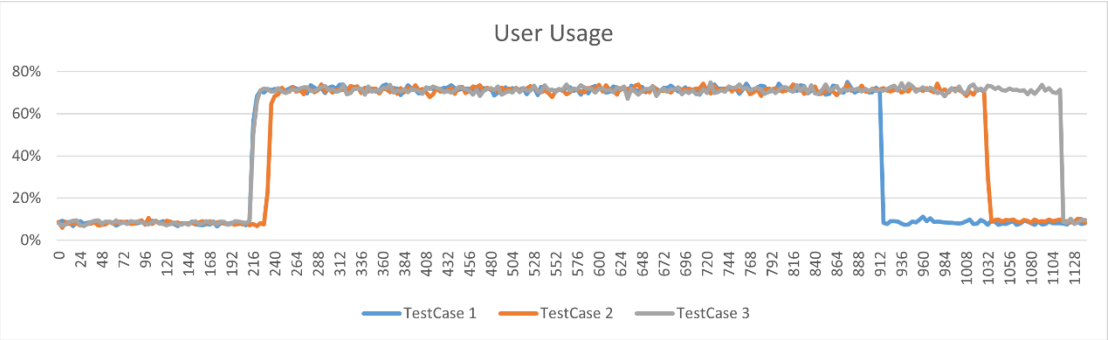
Case	iteration	Response Time (s)			Success %	Failure%	Throughput /s
		Max	Min	Avg			
TestCase 1	258122	0.001	0.008	0.252	100	0	304.130
TestCase 2	297064	0.002	0.008	0.471	100	0	371.040
TestCase 3	334433	0.045	0.008	0.673	100	0	371.195

4.3.2 การใช้งานทรัพยากรเครื่องเซิร์ฟเวอร์

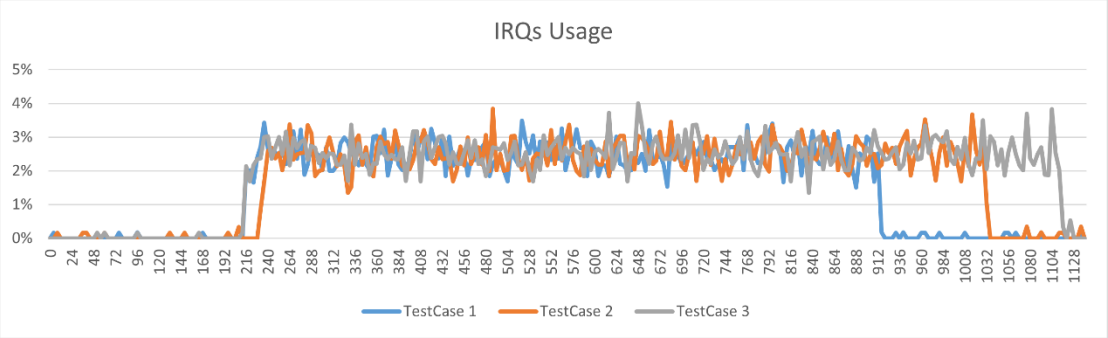
4.3.2.1 CPU



(a)



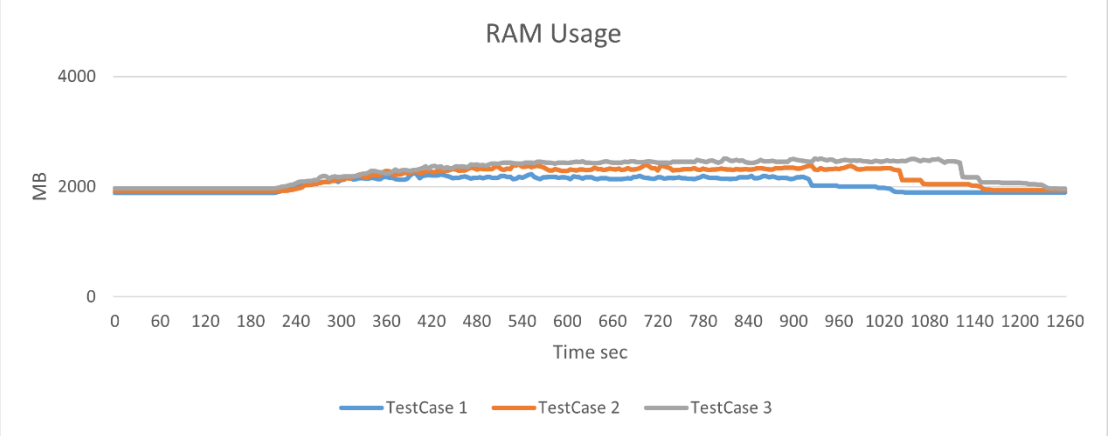
(b)



(c)

ภาพประกอบ 32 การใช้งาน CPU บนเซิร์ฟเวอร์ในสถานการณ์ 3

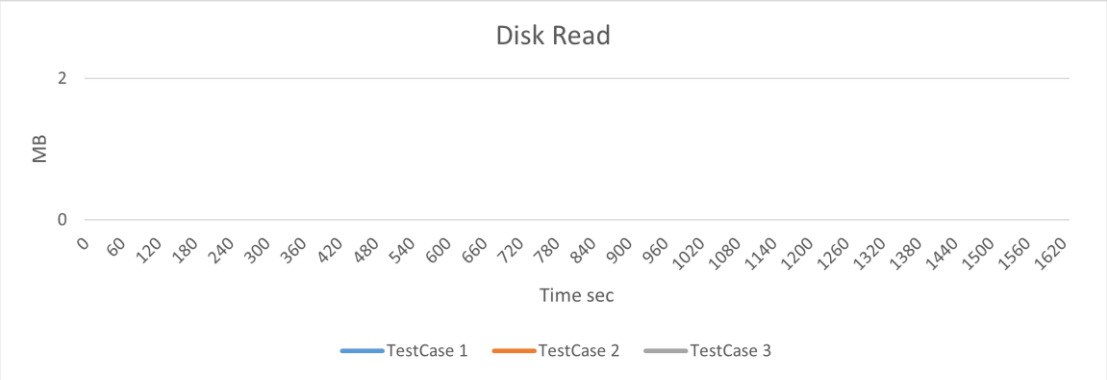
ภาพประกอบ 32 แสดงการใช้งาน CPU ของเซิร์ฟเวอร์พบว่า อัตราการใช้งาน CPU ในทุก TestCase มีความใกล้เคียงกันถึงแม้จะมีปริมาณ Workload ที่เพิ่มขึ้น โดย System อยู่ที่ 18 – 20% User อยู่ที่ 70 – 75% และ IRQs อยู่ที่ 3 – 5%



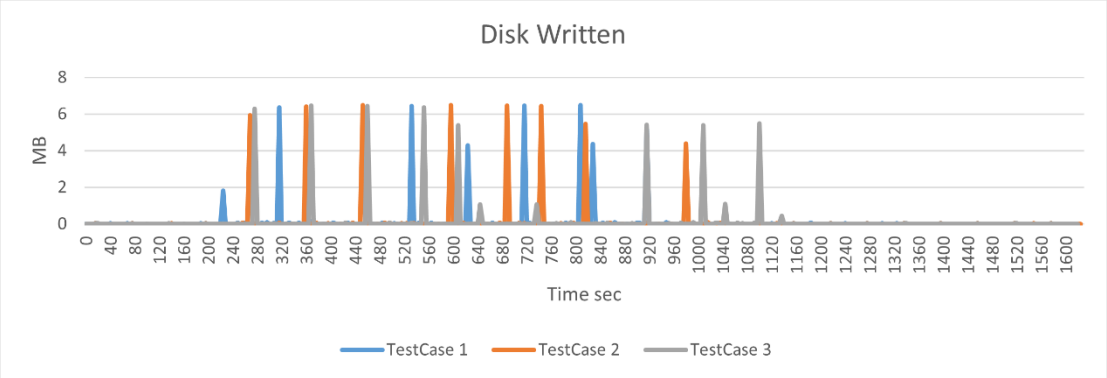
ภาพประกอบ 33 การใช้งาน RAM บนเซิร์ฟเวอร์ในสถานการณ์ 3

ภาพประกอบ 33 แสดงปริมาณการใช้งาน RAM ของเครื่องเซิร์ฟเวอร์ที่มีปริมาณเพิ่มขึ้นตามจำนวน Workload โดย TestCase 3 มีปริมาณการใช้งาน RAM มากที่สุดและระยะเวลาใช้งานยาวนานที่สุด แต่ปริมาณที่เพิ่มขึ้นมีเพียงเล็กน้อย

4.3.2.2 Disk



(a)



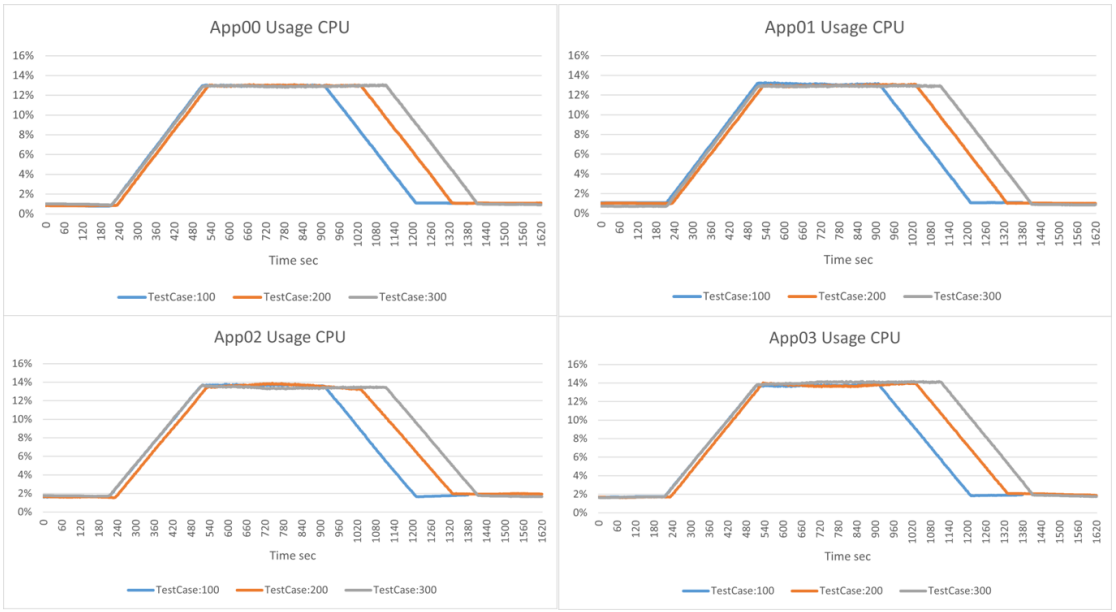
(b)

ภาพประกอบ 34 การใช้งาน Disk บนเซิร์ฟเวอร์ในสถานการณ์ 3

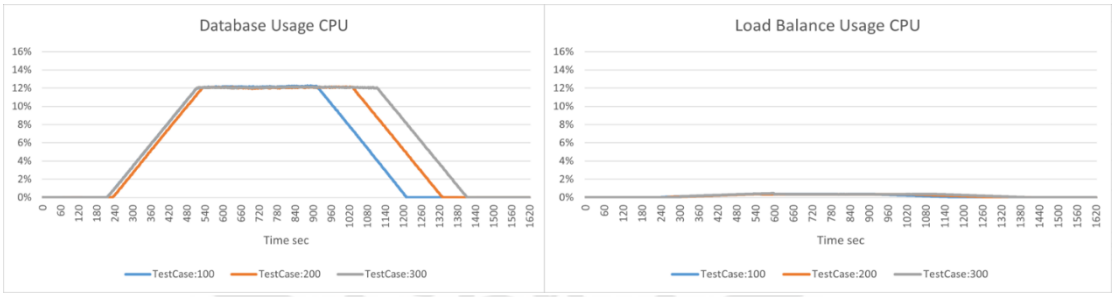
ภาพประกอบ 34(a) แสดงปริมาณการอ่านข้อมูล ซึ่งการอ่านข้อมูลช่วงเกิด Workload ไม่พบการอ่านข้อมูลในทุก TestCase และ 34(b) แสดงปริมาณการเขียนข้อมูลในขณะที่เกิด Workload ซึ่งมีการเขียนข้อมูลเป็นบางช่วงเวลาที่เกิดจากการบันทึก log file ของแอปพลิเคชันในทุก TestCase อย่างไรก็ตาม ปริมาณการเขียนข้อมูลที่เกิดขึ้นนั้นมีปริมาณที่ใกล้เคียงกัน

4.3.3 การใช้งานทรัพยากรของด็อกเกอร์คอนเทนเนอร์

4.3.3.1 CPU



(a)



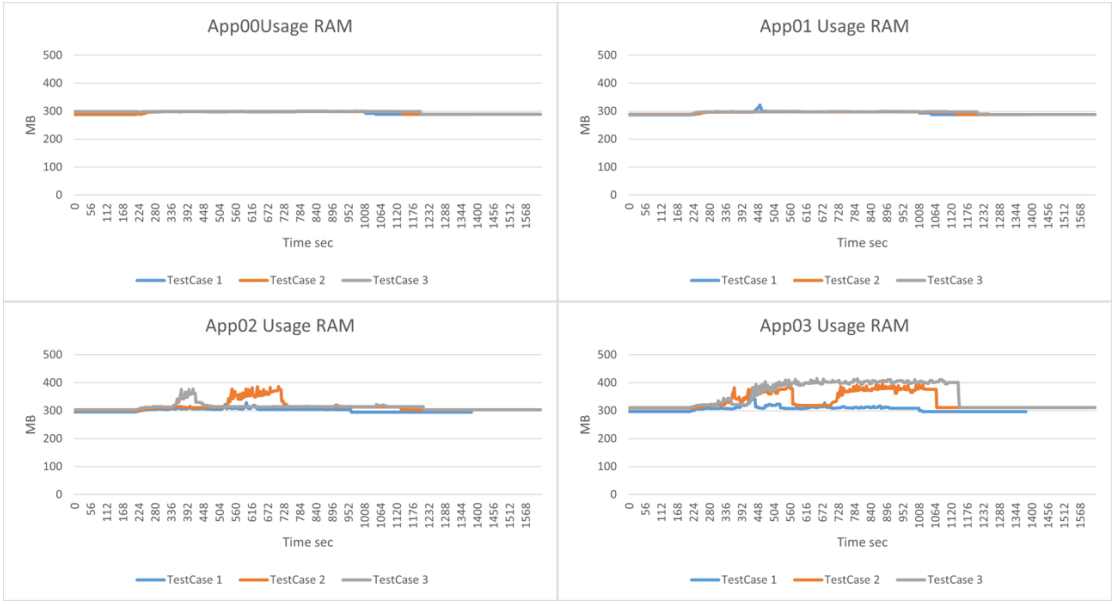
(b)

ภาพประกอบ 35 การใช้งาน CPU ของคอนเทนเนอร์ในสถานการณ์ 3

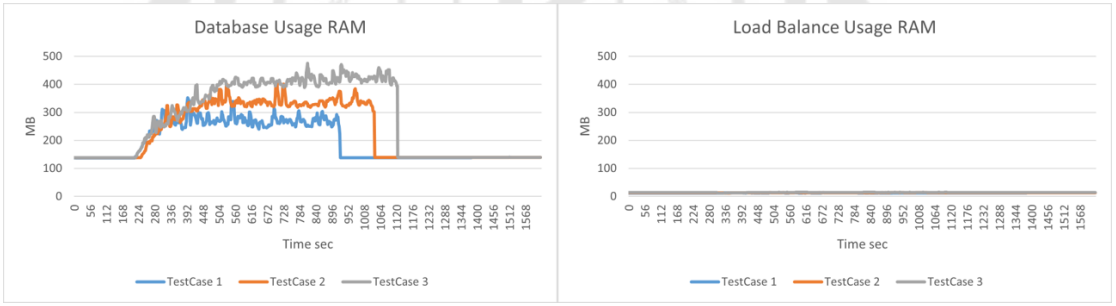
ภาพประกอบ 35(a) แสดงการใช้งาน CPU ของคอนเทนเนอร์แอปพลิเคชัน โดยคอนเทนเนอร์ App00 และ App01 มีอัตราการใช้งาน CPU ใกล้เคียงกันประมาณ 13% แต่คอนเทนเนอร์ App02 และ App03 มีอัตราการใช้งานประมาณ 14%

ภาพประกอบ 35(b) แสดงการใช้งาน CPU ของคอนเทนเนอร์ฐานข้อมูลและคอนเทนเนอร์ Load Balance โดยคอนเทนเนอร์ฐานข้อมูลมีอัตราการใช้งานใกล้เคียงกันในทุก TestCase แต่คอนเทนเนอร์ Load Balance มีอัตราการใช้งานต่ำ

4.3.3.2 RAM



(a)



(b)

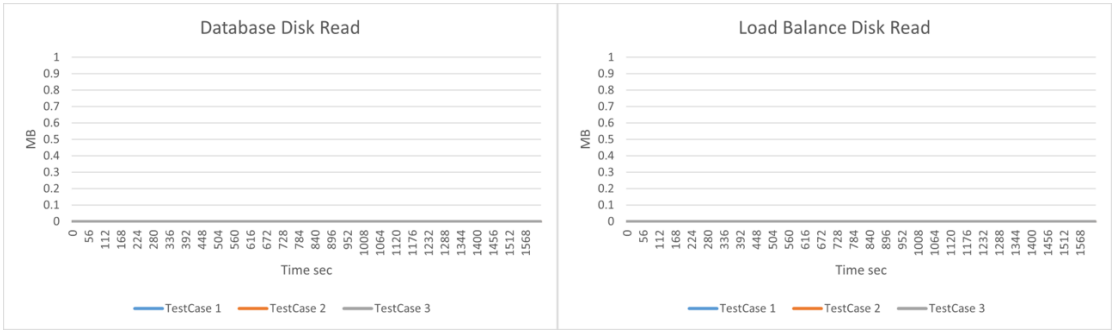
ภาพประกอบ 36 การใช้งาน RAM ของคอนเทนเนอร์ในสถานการณ์ 3

ภาพประกอบ 36(a) แสดงอัตราการใช้งาน RAM ของคอนเทนเนอร์แอปพลิเคชัน พบว่า คอนเทนเนอร์ App00 และ App01 มีอัตราการใช้งานที่น้อย แต่คอนเทนเนอร์ App02 พบการใช้งานที่ TestCase 1 กับ 2 และ App03 พบการใช้งานในทุก TestCase

ภาพประกอบ 36(b) แสดงอัตราการใช้งาน RAM ของคอนเทนเนอร์ฐานข้อมูลและคอนเทนเนอร์ Load Balance พบว่า คอนเทนเนอร์ฐานข้อมูลมีอัตราการใช้งาน RAM ที่เพิ่มขึ้นโดย TestCase 3 มีอัตราการใช้งานมากที่สุด แต่คอนเทนเนอร์ Load Balance มีอัตราการใช้งานต่ำ

4.3.3.3 Disk

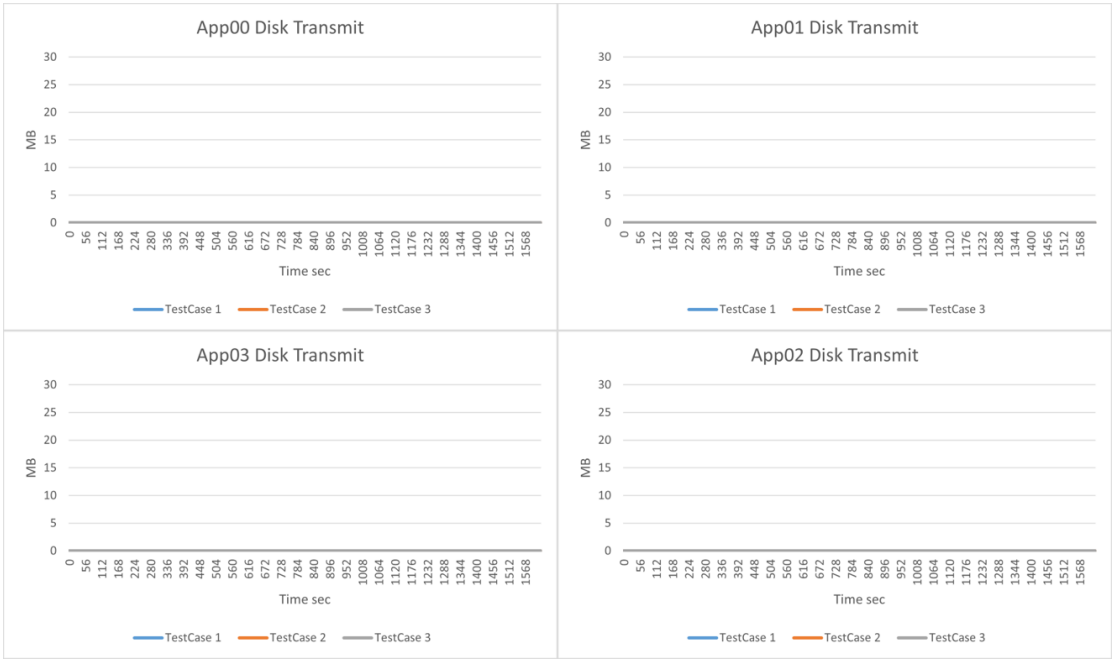
(a)



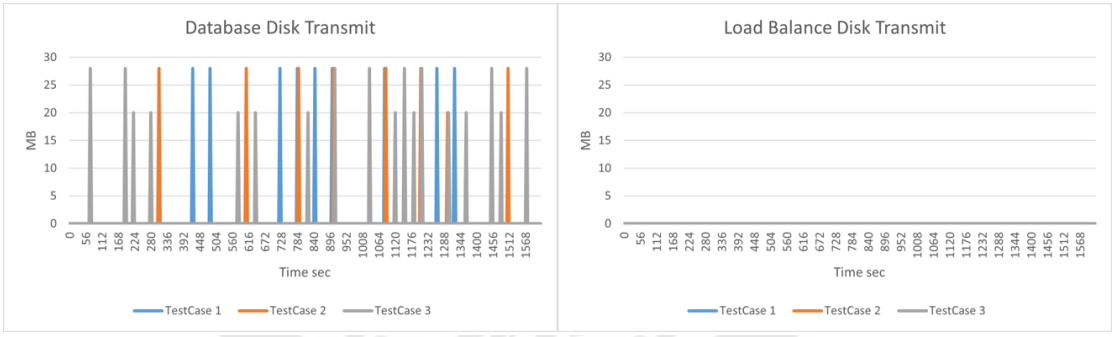
(b)

ภาพประกอบ 37 การใช้งาน Disk Read ของคอนเทนเนอร์ในสถานการณ์ 3

ภาพประกอบ 37 แสดงการอ่านข้อมูลของแอปพลิเคชัน ฐานข้อมูล และ Load Balance ซึ่งไม่พบการอ่านข้อมูลจาก Disk ในทุก TestCase ระหว่างการทดสอบ



(a)



(b)

ภาพประกอบ 38 การใช้งาน Disk Transmit ของคอนเทนเนอร์ในสถานการณ์ 3

ภาพประกอบ 38 แสดงการส่งข้อมูลของคอนเทนเนอร์ โดยคอนเทนเนอร์แอปพลิเคชันและ Load Balance ไม่พบการส่งข้อมูลในทุก TestCase แต่คอนเทนเนอร์ฐานข้อมูลมีการส่งข้อมูลเป็นบางช่วงเวลาด้วยปริมาณที่ใกล้เคียงกันในทุก TestCase ดังแสดงในภาพประกอบ 40(b) ที่ DB Disk I/O Transmit

4.4 รวมผลการใช้งานทรัพยากร

เพื่อให้เห็นข้อมูลการใช้งานทรัพยากรได้อย่างชัดเจนทั้งเซิร์ฟเวอร์และคอนเทนเนอร์ ผู้วิจัยได้ทำการรวมผลข้อมูลการใช้งานทรัพยากรซึ่งมีรายละเอียดดังนี้

4.4.1 ผลรวมการใช้งาน CPU

ตาราง 5 อัตราเฉลี่ยการใช้งาน CPU ของเซิร์ฟเวอร์

TestCase	สถานการณ์	System%	IRQs%	User%
TestCase 1	สถานการณ์ 1	4.536	0.495	29.309
	สถานการณ์ 2	10.513	2.158	51.490
	สถานการณ์ 3	19.791	2.484	71.509
TestCase 2	สถานการณ์ 1	4.324	0.444	28.863
	สถานการณ์ 2	10.357	2.199	49.714
	สถานการณ์ 3	19.531	2.504	71.286
TestCase 3	สถานการณ์ 1	4.425	0.425	28.725
	สถานการณ์ 2	10.468	2.230	50.197
	สถานการณ์ 3	19.388	2.536	71.280

ตาราง 5 แสดงอัตราเฉลี่ยการใช้งาน CPU ของเซิร์ฟเวอร์ โดยตารางจำแนกตาม TestCase ซึ่งมีอัตราการใช้งานที่เพิ่มขึ้นตามจำนวนคอนเทนเนอร์แอปพลิเคชัน

ตาราง 6 อัตราเฉลี่ยการใช้งาน CPU ของเซิร์ฟเวอร์

Test Case	สถานการณ์	App00	App01	App02	App03	Database	Load Balance
TestCase 1	สถานการณ์ 1	17.845				7.991	
	สถานการณ์ 2	16.738	16.754			13.047	0.256
	สถานการณ์ 3	12.567	12.706	13.249	13.331	11.728	0.348
TestCase 2	สถานการณ์ 1	18.348				7.318	
	สถานการณ์ 2	16.207	17.398			11.405	0.245
	สถานการณ์ 3	12.634	12.633	13.210	13.475	11.718	0.337
TestCase 3	สถานการณ์ 1	18.578				7.067	
	สถานการณ์ 2	16.165	17.403			11.384	0.249
	สถานการณ์ 3	12.637	12.604	13.160	13.725	11.770	0.351

ตาราง 6 แสดงอัตราเฉลี่ยการใช้งาน CPU ของเซิร์ฟเวอร์ โดยตารางจำแนกตาม TestCase เพื่อเป็นการเปรียบเทียบการใช้งาน CPU ในแต่ละสถานการณ์ ซึ่งแยกตามการใช้งานของคอนเทนเนอร์ในแต่ละสถานการณ์ช่องว่างในตารางหมายถึงไม่มีคอนเทนเนอร์อยู่ในสถานการณ์นั้น ในการใช้งาน CPU ในสถานการณ์ 2 และ 3 คอนเทนเนอร์แอปพลิเคชันมีอัตราการใช้งานที่ใกล้เคียงกัน แต่มีอัตราการใช้งานลดลงเมื่อเปรียบเทียบกับสถานการณ์ 1

4.4.2 **ผลรวมการใช้งาน** RAM

ตาราง 7 อัตราเฉลี่ยการใช้งาน RAM ของเซิร์ฟเวอร์ในทุกสถานการณ์

TestCase	สถานการณ์	RAM Uasge / MB
TestCase 1	สถานการณ์ 1	984.442
	สถานการณ์ 2	985.574
	สถานการณ์ 3	2125.105
TestCase2	สถานการณ์ 1	1202.365
	สถานการณ์ 2	1202.365
	สถานการณ์ 3	2267.371
TestCase3	สถานการณ์ 1	1342.469
	สถานการณ์ 2	1361.462
	สถานการณ์ 3	2427.883

ตาราง 7 แสดงอัตราเฉลี่ยการใช้งาน RAM ของเซิร์ฟเวอร์ในทุกสถานการณ์ ซึ่งจากข้อมูลการใช้งานมีอัตราเพิ่มขึ้นมากเมื่อดูใน TestCase เดียวกัน แต่ถ้าสังเกตุในสถานการณ์เดียวกันมีอัตราการใช้งานเพิ่มขึ้นเพียงเล็กน้อยเท่านั้น อย่างไรก็ดีอัตราที่เพิ่มขึ้นก็ยังใช้งานเพียงเล็กน้อยถ้าเปรียบเทียบกับทั้งหมด

ตาราง 8 อัตราเฉลี่ยการใช้งาน RAM ของคอนเทนเนอร์ในทุกสถานการณ์

TestCase	สถานการณ์	App 1	App 2	App 3	App 4	Database	Load Balance
TestCase 1	สถานการณ์ 1	327.431				237.121	
	สถานการณ์ 2	308.808	310.813			193.222	12.383
	สถานการณ์ 3	298.157	297.806	304.750	311.361	246.736	12.177
TestCase 2	สถานการณ์ 1	396.179				368.649	
	สถานการณ์ 2	317.563	374.001			249.308	13.334
	สถานการณ์ 3	298.922	297.392	322.433	351.364	300.644	13.125
TestCase 3	สถานการณ์ 1	413.003				502.943	
	สถานการณ์ 2	321.665	388.461			309.091	14.241
	สถานการณ์ 3	298.990	296.287	314.983	365.522	318.233	14.096

ตาราง 8 แสดงอัตราเฉลี่ยการใช้งาน RAM ของคอนเทนเนอร์ทุกสถานการณ์ โดยแยกตามสถานการณ์และคอนเทนเนอร์ ซึ่งแต่ละคอนเทนเนอร์มีอัตราใช้งาน RAM มีความแตกต่างเล็กน้อย

ตาราง 9 อัตราเฉลี่ยการใช้งาน Disk ของเซิร์ฟเวอร์ในทุกสถานการณ์

TestCase	สถานการณ์	Read	Written
TestCase 1	สถานการณ์ 1	0.00	0.05
	สถานการณ์ 2	0.00	0.12
	สถานการณ์ 3	0.00	0.13
TestCase 2	สถานการณ์ 1	0.00	0.05
	สถานการณ์ 2	0.00	0.10
	สถานการณ์ 3	0.00	0.12
TestCase 3	สถานการณ์ 1	0.00	0.06
	สถานการณ์ 2	0.00	0.11
	สถานการณ์ 3	0.00	0.13

ตาราง 9 แสดงอัตราเฉลี่ยการใช้งาน Disk ซึ่งจากตารางไม่พบการอ่านข้อมูลในทุกสถานการณ์ในการทดสอบแสดงในคอลัมน์ Read แต่พบการเขียนข้อมูลของ Disk ในการทดสอบซึ่งมีอัตราการเขียนที่น้อยในคอลัมน์ Written และจากกราฟการใช้งานมีอัตราการใช้งานเป็นช่วงเวลา

ตาราง 10 อัตราเฉลี่ยการใช้งาน Disk การอ่านข้อมูลของคอนเทนเนอร์ในทุกสถานการณ์

TestCase	สถานการณ์	App 01	App 02	App 03	App 04	Database	Load Balance
TestCase 1	สถานการณ์ 1	0				0	
	สถานการณ์ 2	0	0			0	0
	สถานการณ์ 3	0	0	0	0	0	0
TestCase 2	สถานการณ์ 1	0				0	
	สถานการณ์ 2	0	0			0	
	สถานการณ์ 3	0	0	0	0	0	
TestCase 3	สถานการณ์ 1	0				0	
	สถานการณ์ 2	0	0			0	0
	สถานการณ์ 3	0	0	0	0	0	0

ตาราง 10 แสดงอัตราเฉลี่ยการใช้งาน Disk การอ่านข้อมูลของคอนเทนเนอร์ โดยไม่พบการอ่านข้อมูลในทุกคอนเทนเนอร์และในทุกสถานการณ์ ซึ่งสัมพันธ์กับการอ่านข้อมูลของ Disk ของเซิร์ฟเวอร์

ตาราง 11 อัตราเฉลี่ยการใช้งาน Disk การส่งข้อมูลของคอนเทนเนอร์ในทุกสถานการณ์

TestCase	สถานการณ์	App01	App02	App03	App04	Database	Load Balance
TestCase 1	สถานการณ์ 1	0				0.809	0
	สถานการณ์ 2	0	0			0.001	0
	สถานการณ์ 3	0	0	0	0	0.728	0
TestCase 2	สถานการณ์ 1	0				0.872	0
	สถานการณ์ 2	0	0			0.000	0
	สถานการณ์ 3	0	0	0	0	0.532	0
TestCase 3	สถานการณ์ 1	0				0.722	0
	สถานการณ์ 2	0	0			0.001	0
	สถานการณ์ 3	0	0	0	0	1.044	0

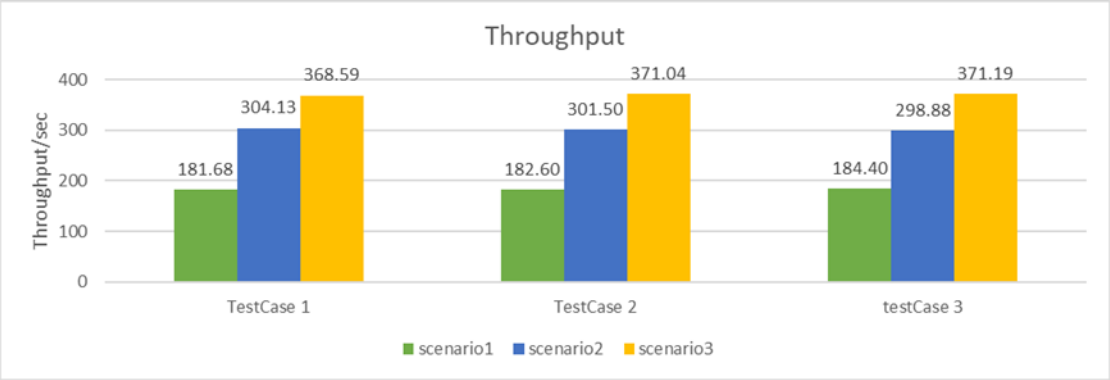
ตาราง 11 แสดงอัตราเฉลี่ยการใช้งาน Disk การส่งข้อมูลของคอนเทนเนอร์ โดยไม่พบการส่งข้อมูลในคอนเทนเนอร์แอปพลิเคชันทั้งหมดในทุกสถานการณ์ แต่พบการส่งข้อมูลบนคอนเทนเนอร์ฐานข้อมูลในคอลัมน์ Database

การใช้งาน Disk ในการอ่านข้อมูลไม่พบอัตราการใช้งานแม้แอปพลิเคชันจะเป็นการเรียกข้อมูลจากฐานข้อมูล ซึ่งอาจจะมาจากการทำงานของแอปพลิเคชันและระบบปฏิบัติการด้วยขนาดข้อมูลในฐานข้อมูลมีขนาดไม่ใหญ่มาก ซึ่งสามารถเก็บไว้บน RAM อีกทั้ง Disk ที่ใช้เป็น SSD จึงส่งผลให้ไม่พบการอ่านข้อมูลบน Disk

4.5 เปรียบเทียบผลการทดลอง

ผลจากการทดสอบด้วยโหลดเทสและผลการใช้งานทรัพยากรที่ได้รวบรวมข้อมูลนำมาเปรียบเทียบเพื่อประเมินสมรรถนะการใช้งานฮาร์ดแวร์ของแอปพลิเคชันบนด็อกเกอร์คอนเทนเนอร์และเพื่อศึกษา Overhead ที่เกิดขึ้นดังผลต่อไปนี้

4.4.1 Throughput



ภาพประกอบ 39 อัตรา Throughput ในแต่ละสถานการณ์จำลอง

ภาพประกอบ 39 แสดงอัตรา Throughput ในแต่ละสถานการณ์ พบว่ามีอัตรา Throughput เพิ่มขึ้นตามจำนวนคอนเทนเนอร์แอปพลิเคชันและในแต่ละ TestCase มีอัตราที่ใกล้เคียงกัน ซึ่งสถานนการณ์ 3 มีอัตรา Throughput มากที่สุด เพื่อเปรียบเทียบอัตรา Throughput ที่เพิ่มขึ้นระหว่างสถานการณ์ โดยสามารถคิดเป็น % จากการคำนวณดังนี้

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

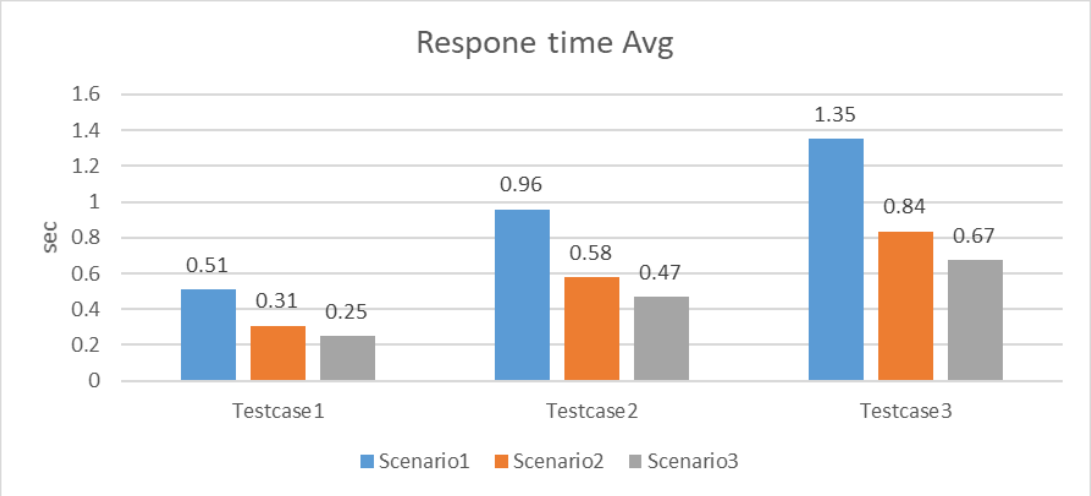
โดยกำหนด Ty เป็น Throughput ของสถานการณ์ที่กำลังเปรียบเทียบ และ Tx เป็น Throughput ที่เพิ่มขึ้นของสถานการณ์ที่มีการเพิ่มจำนวนคอนเทนเนอร์ ซึ่งผลของการคำนวณแสดงในตาราง 12

ตาราง 12 ประสิทธิภาพ Throughput ที่เพิ่มขึ้นระหว่างสถานการณ์

TestCase	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%

ตาราง 12 แสดงการเปรียบเทียบประสิทธิภาพ Throughput ที่เพิ่มขึ้น โดยสถานการณ์ 1 เปรียบเทียบกับสถานการณ์ 2 มีประสิทธิภาพเพิ่มขึ้น 67.40% และสถานการณ์ 2 เปรียบเทียบกับสถานการณ์ 3 มีประสิทธิภาพเพิ่มขึ้น 21.20% ใน TestCase 1 ซึ่งในอุดมคติอัตรา Throughput ควรมีอัตราเพิ่มขึ้นเป็นสองเท่า เมื่อเพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชันเป็นสองเท่า โดยระหว่างสถานการณ์ 2 กับ 3 มีประสิทธิภาพลดลงเมื่อเปรียบเทียบระหว่างสถานการณ์ 1 กับ 2

4.4.2 Response Time



ภาพประกอบ 40 Response Time ในแต่ละสถานการณ์

ภาพประกอบ 40 แสดงผล Response Time พบว่าแต่ละสถานการณ์มี Response Time ที่ลดลงและในแต่ละ TestCase ลดลงใกล้เคียงกัน ซึ่งการเพิ่มจำนวนแอปพลิเคชันคอนเทนเนอร์สามารถเพิ่มความเร็วในการประมวลผลส่งผลให้ Response Time ลดลง เพื่อเปรียบเทียบ Response Time ที่ลดลงในแต่ละสถานการณ์ โดยสามารถคิดเป็น % ด้วยการคำนวณตามสูตร

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

โดยที่ Ry เป็น Response Time ของสถานการณ์ที่กำลังเปรียบเทียบและ Rx เป็น Response Time ที่เพิ่มขึ้นของสถานการณ์ที่มีการเพิ่มจำนวนคอนเทนเนอร์ผลของการคำนวณแสดงในตาราง 13

ตาราง 13 ประสิทธิภาพ Response Time ที่เพิ่มขึ้น

TestCase	สถานการณ์ 2 กับ สถานการณ์ 1	สถานการณ์ 3 กับ สถานการณ์ 2
TestCase 1	66.09%	22.15%
TestCase 2	65.15%	23.07%
TestCase 3	61.55%	24.20%

ตาราง 13 แสดงการเปรียบเทียบประสิทธิภาพ Response Time ที่เพิ่มขึ้น โดยสถานการณ์ 1 เปรียบเทียบกับสถานการณ์ 2 มีเวลาเฉลี่ยลดลง 66.09% และสถานการณ์ 2 เปรียบเทียบกับสถานการณ์ 3 มีเวลาเฉลี่ยนลดลง 22.15% ใน TestCase 1 ซึ่งในอุดมคติ Response Time ควรลดลงเป็นสองเท่าเมื่อเพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชันเป็นสองเท่าในสถานการณ์ 2 กับ 3 และ % Response Time ระหว่างสถานการณ์มีเปอร์เซนต์ที่ลดลงเมื่อเปรียบเทียบกับสถานการณ์ 1 กับ 2

4.4.3 การใช้งานทรัพยากร

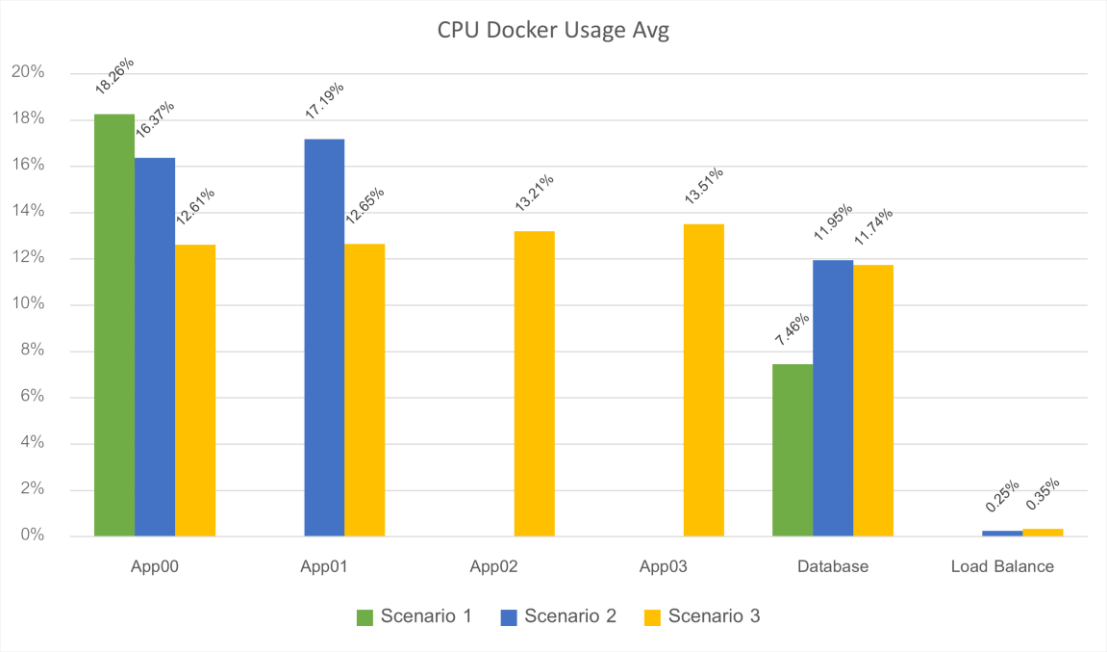
การตรวจสอบการใช้งานทรัพยากรของเซิร์ฟเวอร์พบว่า การใช้งานทรัพยากรส่วนใหญ่อยู่ที่ CPU เพื่อให้เห็นถึงการใช้งานทรัพยากรและเปรียบเทียบการทำงานที่เกิดขึ้น โดยการใช้งาน CPU ของ User มีอัตราการใช้งานมากที่สุด ซึ่งมากจากการทำงานของแอปพลิเคชันและแต่ละสถานการณ์ยังมีอัตราการใช้งานที่เพิ่มขึ้น เช่นเดียวกับ System และ RIQs ดังตาราง 14

ตาราง 14 ค่าเฉลี่ยอัตราการใช้งาน CPU บนเซิร์ฟเวอร์

สถานการณ์	System	IRQs	User
สถานการณ์ 1	4.43%	0.45%	28.97%
สถานการณ์ 2	10.45%	2.20%	50.47%
สถานการณ์ 3	19.57%	2.51%	71.36%

การการใช้งาน CPU ที่เพิ่มขึ้นของ System สัมพันธ์กับจำนวนคอนเทนเนอร์ที่เพิ่มขึ้น โดยสถานการณ์ 1 เป็นสถานการณ์ตั้งต้น System มีอัตราการใช้งานที่ 4.43% ในสถานการณ์ 2 System มีอัตราการใช้งานเพิ่มขึ้นเป็น 10.45% และสถานการณ์ 3 เพิ่มขึ้นเป็น 19.57% ซึ่งการเพิ่มขึ้นของคอนเทนเนอร์นอกจากจะเพิ่มการใช้งาน CPU ของ User หรือคอนเทนเนอร์ยังส่งผลให้เพิ่มการใช้งานของ System หรือระบบปฏิบัติการ เนื่องจากระบบปฏิบัติต้องใช้ CPU ในการจัดการกับคอนเทนเนอร์ที่เพิ่มขึ้นและด้วยคอนเทนเนอร์ที่เพิ่มขึ้นทำงานในลักษณะเดียวกัน ส่งผลให้ IRQs มีอัตราที่เพิ่มขึ้น

การตรวจสอบการใช้งาน CPU ของคอนเทนเนอร์ โดยทุกสถานการณ์ในแต่ละคอนเทนเนอร์แอปพลิเคชันมีอัตราใช้งาน CPU ใกล้เคียงกัน แต่เมื่อจำนวนคอนเทนเนอร์แอปพลิเคชันเพิ่มขึ้นจะมีอัตราใช้งาน CPU ลดลงเพื่อแบ่งการทำงานให้กับคอนเทนเนอร์ที่เพิ่มขึ้น ดังภาพประกอบ 41



ภาพประกอบ 41 การใช้งาน CPU บนคอนเทนเนอร์ทุกสถานการณ์

นอกจากนี้การเพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชันยังส่งผลให้การใช้งาน CPU ของคอนเทนเนอร์ฐานข้อมูลเพิ่มขึ้น เนื่องจากการทดลองกำหนดให้ทุกสถานการณ์มีเพียงหนึ่งคอนเทนเนอร์ฐานข้อมูล ส่วนคอนเทนเนอร์ Load Balance มีอัตราการใช้งานต่ำเนื่องด้วยแอปพลิเคชันทำงานบน HTTP แต่ยังสามารถกระจายงานได้อย่างมีประสิทธิภาพ

ตาราง 15 อัตราเฉลี่ยการใช้งาน CPU ต่อ 1 Throughput/s

สถานการณ์	System: 1 TPS	IRQs: 1 TPS	User: 1 TPS
สถานการณ์ 1	0.024%	0.002%	0.158%
สถานการณ์ 2	0.035%	0.007%	0.167%
สถานการณ์ 3	0.053%	0.007%	0.193%

ตาราง 15 แสดงอัตราเฉลี่ยการใช้งาน CPU ต่อ 1 Throughput ต่อ 1 วินาที โดยในทุกสถานการณ์มีอัตราการใช้งานเพิ่มขึ้น แม้จะมีอัตรา Workload ที่เท่ากัน แสดงให้เห็นว่าการเพิ่มจำนวนคอนเทนเนอร์ส่งผลให้ในการประมวลผลงาน 1 TPS เพิ่มอัตราการใช้งาน CPU โดยในสถานการณ์ 3 ที่ User ใช้งาน CPU เพิ่มขึ้น 0.193%

บทที่ 5
สรุปผลการวิจัย

งานวิจัยการประเมินสมรรถนะการใช้งานฮาร์แวร์ของแอปพลิเคชันบนด็อกเกอร์คอนเทนเนอร์ โดยการสร้างสถานการณ์จำลองด้วยการเพิ่มจำนวนคอนเทนเนอร์ของแอปพลิเคชัน ซึ่งทดสอบด้วยการโหลดเทส เพื่อเก็บข้อมูลการใช้งานทรัพยากรและประสิทธิภาพ การประมวลผลและนำมาเปรียบเทียบ สามารถสรุปผลการวิจัยได้ดังนี้

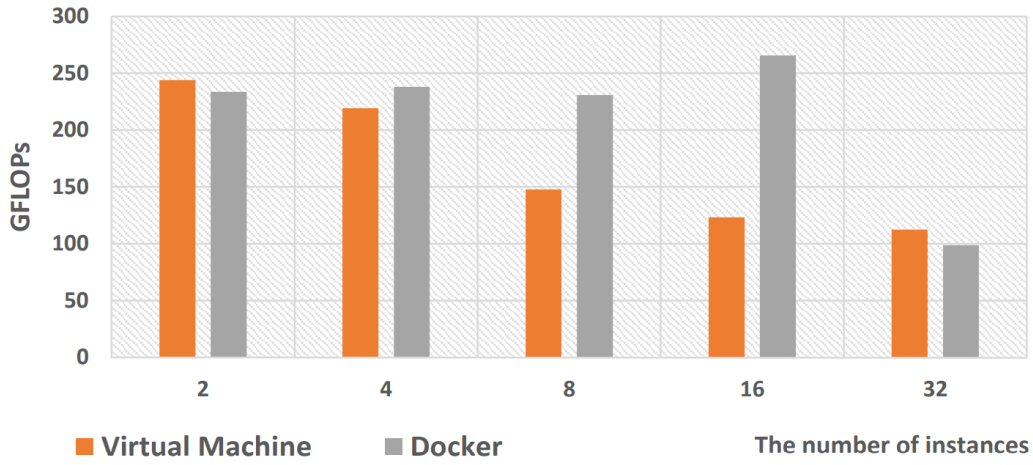
5.1 สรุปผลงานวิจัย

การทดลองพบว่า การใช้งานฮาร์ดแวร์ของด็อกเกอร์คอนเทนเนอร์ โดยการใช้งานส่วนใหญ่เกิดขึ้นที่ CPU ซึ่งบนเซิร์ฟเวอร์มีอัตราการใช้งาน CPU เพิ่มขึ้นเมื่อเพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชัน โดยในสถานการณ์ 1 มีอัตราการใช้งานสูงสูดใกล้เคียงกันที่ 30% แม้จะมีอัตรา Work load เพิ่มขึ้นเช่นเดียวกับในสถานการณ์ 2 และ 3 ซึ่งเกิดจากแอปพลิเคชันที่ใช้ในการทดสอบที่เป็นเพียงการเรียกข้อมูลเท่านั้น ส่วนการใช้งาน RAM มีอัตราการใช้งานเพิ่มขึ้นเพียงเล็กน้อยเช่นเดียวกับการใช้งาน Disk ซึ่งการใช้งานการอ่านของ Disk ไม่พบการใช้งานในทุกสถานการณ์แม้การทำงานแอปพลิเคชันเป็นการอ่านข้อมูล ส่วนการใช้งานการเขียนของ Disk พบการใช้งานเป็นช่วงจังหวะ ซึ่งเกิดจาการเขียน log file ของระบบปฏิบัติการ โดยการใช้งานทรัพยากรของคอนเทนเนอร์สัมพันธ์กับการใช้งานของเซิร์ฟเวอร์ ซึ่งการใช้งานส่วนใหญ่เกิดขึ้นที่ CPU ในลักษณะเดียวกันกับเซิร์ฟเวอร์ โดยในสถานการณ์ 1 ที่คอนเทนเนอร์แอปพลิเคชันมีอัตราการใช้งาน CPU สูงสุด ยังคงมีอัตราที่ใกล้เคียงกัน ซึ่งในสถานการณ์ 2 และ 3 ที่ทำการเพิ่มจำนวนแอปพลิเคชันคอนเทนเนอร์ก็ยังคงมีอัตราการใช้งานสูงสูดใกล้เคียงกันแม้จะมีอัตรา Work load เพิ่มขึ้น แต่อัตราการใช้งานสูงสุดลดต่ำลงเมื่อเปรียบเทียบกับสถานการณ์ 1 เนื่องจากการกระจายงานของ Load Balance และการใช้งาน CPU ของทุกคอนเทนเนอร์แอปพลิเคชันในสถานการณ์ 2 และ 3 มีอัตราเฉลี่ยใกล้เคียงกันอย่างมาก แสดงให้เห็นว่าการทำงาน Load Balance สามารถกระจายงานไปยังคอนเทนเนอร์แอปพลิเคชันมีประสิทธิภาพและแสดงให้เห็นถึงการบริหารจัดการทรัพยากรของด็อกเกอร์ให้กับคอนเทนเนอร์สามารถปรับเพิ่มลดการใช้งานทรัพยากร โดยสังเกตุได้ที่คอนเทนเนอร์ฐานข้อมูลที่มีอัตราการใช้งาน CPU และ RAM เพิ่มขึ้นในสถานการณ์ 2 และ 3 เนื่องด้วยในการทดลองนี้มีคอนเทนเนอร์ฐานข้อมูลเพียง 1 คอนเทนเนอร์ ซึ่งจะต้องรองรับการเรียกข้อมูลจากคอนเทนเนอร์แอปพลิเคชัน

การเพิ่มจำนวนของคอนเทนเนอร์แอปพลิเคชันสามารถเพิ่มประสิทธิภาพในการประมวลผล (Throughput) ได้มากว่า 60% และยังลดเวลาในการประมวลผล (Response Time) ได้มากกว่า 60 % ในการเปรียบเทียบระหว่างสถานการณ์ 1 กับสถานการณ์ 2 ซึ่งแสดงให้เห็นว่าการเพิ่มจำนวนคอนเทนเนอร์แอปพลิเคชันไม่สามารถเพิ่ม Throughput และ Response Time เป็น 1 เท่า และเมื่อเปรียบเทียบระหว่างสถานการณ์ 2 กับสถานการณ์ 3 Throughput เพิ่มขึ้นเพียง 23% เท่านั้น ซึ่งสถานการณ์ 2 และ 3 จำนวนคอนเทนเนอร์แอปพลิเคชันเพิ่มขึ้นเป็นเท่าตัว

ค่าดำเนินการ (Overhead) ที่เกิดขึ้นจากการเพิ่มจำนวนแอปพลิเคชันคอนเทนเนอร์ โดยเมื่อพิจารณาจากข้อมูลการใช้งาน CPU ของทั้ง 3 สถานการณ์ในตาราง 7 อัตราการใช้งานมากที่สุดเกิดขึ้นที่ User ซึ่งเป็นการใช้งาน CPU ด้วย User ของระบบปฏิบัติการหรือการทำงานของด็อกเกอร์มีอัตราการใช้งาน CPU เพิ่มขึ้นด้วยการเพิ่มจำนวนแอปพลิเคชันคอนเทนเนอร์ ซึ่งถือเป็นปกติเมื่อเพิ่มจำนวนคอนเทนเนอร์จะต้องเกิดการใช้งาน CPU เพิ่มขึ้น แต่นอกจากการเพิ่มขึ้นของ CPU ที่ใช้โดยคอนเทนเนอร์ยังมีการใช้งาน CPU ด้วยระบบปฏิบัติการหรือ System เพื่อใช้งานการจัดการกับคอนเทนเนอร์ซึ่งมีอัตราการใช้งานที่เพิ่มสูงขึ้นอย่างมาก โดยสถานการณ์ 1 System มีอัตราการใช้งาน CPU เฉลี่ยอยู่ที่ 4.43% ในสถานการณ์ 2 System มีอัตราการใช้งานเฉลี่ยที่ 10.45% ซึ่งอัตราการใช้งานเพิ่มขึ้นถึง 1 เท่า ถึงแม้จะเป็นเพียงการเพิ่มคอนเทนเนอร์แอปพลิเคชัน 1 คอนเทนเนอร์เท่านั้น และสถานการณ์ 3 System มีอัตราการใช้งานเฉลี่ยที่ 19.57% ซึ่งอัตราการใช้งานเพิ่มขึ้นเกือบ 1 เท่า โดยสถานการณ์ 3 เพิ่มคอนเทนเนอร์แอปพลิเคชัน 2 คอนเทนเนอร์จากสถานการณ์ 2 แสดงให้เห็นว่าการเพิ่มคอนเทนเนอร์แอปพลิเคชันยิ่งจำนวนมากขึ้นค่าดำเนินการจะยิ่งเพิ่มอัตราการใช้งานอย่างมาก ซึ่งจะส่งผลให้คอนเทนเนอร์แอปพลิเคชันไม่สามารถใช้งานทรัพยากรได้อย่างเต็มประสิทธิภาพ นอกจากนี้ยังส่งผลให้อัตราการใช้งาน CPU โดย IRQs เพิ่มขึ้นเนื่องด้วยคอนเทนเนอร์แอปพลิเคชันที่เพิ่มจำนวนและคอนเทนเนอร์ฐานข้อมูลมีเพียงคอนเทนเนอร์เดียว อีกทั้งแอปพลิเคชันยังทำงานบน Node.js ที่ทำงานในลักษาณะ Single Thread โดยในงานวิจัย Using Docker in High Performance Computing Applications มีลักษณะการทดลองที่คล้ายคลึงกัน ซึ่งทำการทดลองด้วยการเพิ่มจำนวนคอนเทนเนอร์และ VMs บนเครื่อง HPC เพื่อเปรียบเทียบประสิทธิภาพการทำงาน โดยผลลัพธ์การทดลองคอนเทนเนอร์สามารถทำงานได้ดีกว่าที่ 16 คอนเทนเนอร์ เมื่อพิจารณาการทำงานของคอนเทนอร์มีอัตราการประมวลผลเพิ่มขึ้นตามจำนวนคอนเทนเนอร์แต่เมื่อเพิ่มถึง 32 คอนเทนเนอร์กลับมีอัตราที่ต่ำลงและมีประสิทธิภาพการประมวลผลต่ำกว่า VMs

ดังภาพประกอบ 42 โดยในงานวิจัยวิเคราะห์ไว้ว่า ด็อกเกอร์มีค่าดำเนินการเพิ่มขึ้นตามจำนวนคอนเทนเนอร์และการแชร์ทรัพยากร เมื่อเปรียบเทียบกับผลงานวิจัยนี้ที่อัตราค่าดำเนินการและประสิทธิภาพการทำงานที่เพิ่มขึ้นมีลักษณะที่ใกล้เคียงกัน โดยอัตราการเพิ่มขึ้นของจำนวนคอนเทนเนอร์เพิ่มขึ้นเท่าตัวแต่ประสิทธิภาพไม่สามารถเพิ่งขึ้นได้เท่าตัวเช่นกันและการเพิ่มจำนวนคอนเทนเนอร์ส่งผลต่อค่าดำเนินการอย่างมากอีกเช่นกันแต่ในงานวิจัยที่นำมาเปรียบเทียบไม่มีผลค่าดำเนินการอย่างชัดเจนแต่สังเกตได้จากภาพประกอบ 42 ที่ 32 คอนเทนเนอร์มีประสิทธิภาพลดลงอย่างมาก



ภาพประกอบ 42 ผลการทดสอบการประมวล CPU ของงานวิจัย Using Docker in High Performance Computing Applications

การเพิ่มจำนวนคอนเทนเนอร์บนเซิร์ฟเวอร์เครื่องเดียวจำเป็นจะต้องคำนึงถึง Overhead เพราะจำนวนคอนเทนเนอร์ยิ่งมีจำนวนมากขึ้นซึ่งส่งผลโดยตรงต่อ Overhead แต่อย่างไรก็ตามในการทดลองเป็นเพียงการศึกษาบนเซิร์ฟเวอร์เครื่องเดียว เนื่องจากข้อจำกัดที่ไม่สามารถหาเซิร์ฟเวอร์เพียงพอในการทำคลัสเตอร์(Cluster) จึงไม่สามารถเปรียบเทียบระหว่างเซิร์ฟเวอร์เครื่องเดียวกับเซิร์เวอร์ที่เชื่อมต่อเป็นคัสเตอร์ได้

5.2 ข้อเสนอแนะ

5.2.1 เนื่องจากงานวิจัยนี้ใช้โครงสร้างซอฟต์แวร์ที่ไม่ซับซ้อนมีการประมวลน้อยซึ่งอาจจะทำให้ไม่ใกล้เคียงกับซอฟต์แวร์ในการทำงานจริง ดังนั้นหากสามารถนำซอฟต์แวร์ที่ให้บริการจริงจะช่วยให้เห็นการใช้ทรัพยากรที่ชัดเจนขึ้น

5.2.2 ในงานวิจัยนี้การเพิ่มคอนเทนเนอร์แอปพลิเคชันเป็นการเพิ่มโดยการกำหนดจำนวนคอนเทนเนอร์แอปพลิเคชันบน Dockerfile ซึ่งไม่ได้ใช้การเพิ่มแบบอัตโนมัติ (auto scaling) ถ้านำ Docker Swarm หรือ Kubernetes มาใช้จะทำให้เห็นพฤติกรรมการใช้ทรัพยากรในช่วงที่มีการเพิ่มจำนวนคอนเทนเนอร์

5.2.3 เพื่อให้เห็นปัจจัยในค่านำเนินการควรทำการเปรียบเทียบการทำงานบนเซิร์ฟเวอร์เครื่องกับการทำงานบนเครื่องเซิร์ฟเวอร์แบบคลัสเตอร์(Cluster)

5.2.4 กำหนดขนาดของทรัพยากรให้กับคอนเทนเนอร์ที่ใช้ในการทดสอบ

บรรณานุกรม

Casalicchio, E., และ 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.

Chung, M. T., Quang-Hung, N., Nguyen, M.-T., และ Thoai, N. (2016). *Using docker in high performance computing applications.* Paper presented at the 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE).

Docker. (2015). Docker. https://docs.docker.com/get-started/overview/

Herrera-Izquierdo, L., และ Grob, M. (2017). A performance evaluation between Docker container and Virtual Machines in cloud computing architectures. *Maskana, 8*, 127-133.

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).

K6. (2019). k6 documentation. https://k6.io/docs/

Kwon, S., และ Lee, J.-H. (2020). DIVDS: docker image vulnerability diagnostic system. *IEEE Access, 8*, 42666-42673.

linuxcontainers. (2018). What's LXC. https://linuxcontainers.org/lxc/introduction/

Mavridis, I., และ Karatza, H. (2017, 24-27 July 2017). *Performance and Overhead Study of Containers Running on Top of Virtual Machines.* Paper presented at the 2017 IEEE 19th Conference on Business Informatics (CBI).

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., และ Al-Hammadi, Y. (2017, 7-9 March 2017). *Performance comparison between container-based and VM-based services.* Paper presented at the 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN).

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 นภวิชญ์, ท. (2020). PERFORMANCE COMPARISON BETWEEN MONOLITH AND MICROSERVICES USING DOCKER AND KUBERNETES : การเปรียบเทียบประสิทธิภาพระหว่างสถาปัตยกรรมแบบโมโนลิธ และไมโครเซอร์วิสโดยใช้ดอคเกอร์และคูเบอร์เนตส์. In: Srinakharinwirot University.

Yasrab, R. (2018). Mitigating docker security issues. *arXiv preprint arXiv:1804.05039*.

ขุนเพร็ชร์, ธ. (2015). การขยายขนาดการให้บริการแบบอัตโนมัติในระบบด็อกเกอร์. (ปริญญานิพนธ์ปริญญามหาบัณฑิต). มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ, กรุงเทพฯ. https://tdc.thailis.or.th/tdc/browse.php?option=show&browse_type=title&titleid=493457&query=%A1%D2%C3%A2%C2%D2%C2%A2%B9%D2%B4%A1%D2%C3%E3%CB%E9%BA%C3%D4%A1%D2%C3%E1%BA%BA%CD%D1%B5%E2%B9%C1%D1%B5%D4%E3%B9%C3%D0%BA%BA%B4%E7%CD%A1%E0%A1%CD%C3%EC&s_mode=any&d_field=&d_start=0000-00-00&d_end=2564-06-25&limit_lang=&limited_lang_code=&order=&order_by=&order_type=&result_id=2&maxid=2#

ประวัติผู้เขียน

ชื่อ-สกุล	ธนู คะชา
วัน เดือน ปี เกิด	11 กุมภาพันธ์ 2535
สถานที่เกิด	ตราด
วุฒิการศึกษา	พ.ศ.2559 วิทยาศาสตรบัณฑิต สาขาเทคโนโลยีสารสนเทศ จาก มหาวิทยาลัยราชภัฏสวนสุนันทา
ที่อยู่ปัจจุบัน	568 หมู่ 1 ตำบลประณีต อำเภอเขาสมิง จังหวัดตราด