ระบบการประเมินและเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมการขยายตัวแบบอัตโนมัติของซอฟต์แวร์ควบคุมคอนเทนเนอร์

วรเทพ อหันทริก

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์
มหาวิทยาลัยธุรกิจบัณฑิตย์
ปีการศึกษา 2565

PERFORMANCE EVALUATION AND COMPARISON OF AUTO-SCALING ALGORITHM IN CONTAINER ORCHESTRATION SOFTWARE

WORATHEP AHANTARIG

A Thematic Paper Submitted in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering
Department of Computer Engineering,
College of Innovative Technology and Engineering
Dhurakij Pundit University
Academic Year 2022

ใบรับรองสารนิพนธ์

วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต หลักสูตรวิศวกรรมคอมพิวเตอร์

หัวข้อสารนิพนธ์	ระบบการประเมินและเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมการขยายตัวแบบอัตโนมัติของซอฟต์แวร์ควบคุมคอนเทนเนอร์
เสนอโดย	นายวรเทพ อหันทริก
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาสารนิพนธ์	ดร.ธนัญ จารุวิทยโกวิท

ได้พิจารณาเห็นชอบโดยคณะกรรมการสอบสารนิพนธ์แล้ว

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณรงค์เดช กีรติพรานนท์)

..... กรรมการและอาจารย์ที่ปรึกษาสารนิพนธ์

(อาจารย์ ดร.ธนัญ จารุวิทยโกวิท)

.....กรรมการ

(อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์

วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์รับรองแล้ว

.....คณบดีวิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์

(อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์)

วันที่ 27 เดือน ก.ค. พ.ศ. 2566

หัวข้อสารนิพนธ์	ระบบการประเมินและเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมการขยายตัวแบบอัตโนมัติของซอฟต์แวร์ควบคุมคอนเทนเนอร์
ชื่อผู้เขียน	วรเทพ อหันทริก
อาจารย์ที่ปรึกษา	ดร. ธนัญ จารุวิทยโกวิท
หลักสูตร	วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์)
ปีการศึกษา	2565

บทคัดย่อ

งานวิจัยฉบับนี้จัดทำขึ้นเพื่อศึกษาและประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพของการขยายตัวของ Service บนคอนเทนเนอร์ 2 แพลตฟอร์ม Kubernetes และ Docker Swarm ซึ่งจะประเมินประสิทธิภาพในด้านของ ซีพียู และเวลาในการขยายตัวของ Service ทั้ง 2 คือ NGINX และ Web Service (VITE) เพื่อลดปัญหาการใช้งานจากผู้ใช้งานที่มีปริมาณมาก

งานวิจัยนี้ได้ใช้ Public Cloud ของ Google Cloud Platform จัดการโครงสร้าง Kubernetes ด้วย Google Kubernetes Engine (GKE) เป็นเครื่องมือช่วยบริหารจัดการ container ทำงานบน Cloud ช่วยในเรื่องของการสร้าง (build) Service, การนำ Service ไปใช้ (deploy) และการรองรับการขยายตัว (scale) ส่วนของ Docker Swarm จะสร้าง VM Instances บน Google Container Engine โดยบนคอนเทนเนอร์ 2 แพลตฟอร์ม จะติดตั้ง Workload 2 ชนิด คือ NGINX เป็น Web Service ทั่วไป และ VITE เป็น Web Service โดยใส่เนื้อหาวีดีโอเป็น Full HD: ความละเอียดของภาพ 1080p เพื่อใช้เปรียบเทียบประสิทธิภาพการใช้งาน CPU ของแต่ละ Service ซึ่งกำหนดสถานะไว้ ถ้าสถานะปกติคือ CPU ต่ำกว่า 80% ให้ Service ทำงานแต่ 1 Service แต่ถ้ามีปริการใช้งาน CPU มากกว่า 80% จะทำการขยายตัวเพื่อให้รองรับปริมาณที่มาใช้มากขึ้น โดยตั้งไว้ให้ขยายได้มากสุดถึง 5 Service เพื่อให้รองรับการทำงานของผู้ใช้ได้อย่างมีประสิทธิภาพ

จากการที่ได้ทดลองเปรียบเทียบทั้ง 2 แพลตฟอร์ม พบว่าการทำงานของ Kubernetes มีความสะดวกสบายสามารถทำการขยายตัว Service แบบอัตโนมัติได้ และยังขยายตัวได้เร็วกว่าถึง 93% ในส่วนของ Docker Swarm ไม่มี Auto scaling ต้องอาศัยการ Monitor จากผู้ดูแลระบบ จำเป็นต้องให้ผู้ดูแลระบบทำ manual scaling ทำให้เกิดความล่าช้ากว่ามาก สรุปคือ Kubernetes สามารถมีการขยายตัวได้เร็วกว่า Docker Swarm

คำสำคัญ: การขยายตัว, คอนเทนเนอร์, เว็บเซอร์วิซ

.....

อาจารย์ที่ปรึกษา

Thematic Paper Title	PERFORMANCE EVALUATION AND COMPARISON OF AUTO-SCALING ALGORITHM IN CONTAINER ORCHESTRATION SOFTWARE
Author	Worathep Ahantarig
Thematic Paper Advisor	Dr. Tanun Jaruvitayakovit
Program	Master of Engineering (Computer Engineering)
Academic Year	2022

ABSTRACT

This research project was conducted to study and evaluate the competence and performance comparison of scaling services on two container platforms, Kubernetes and Docker Swarm. It aims to assess the efficiency in terms of CPU and the time taken for the expansion of two services, namely NGINX and Web Service (VITE). The objective is to address user-related issues that arise due to high user volumes.

This research project utilized the Public Cloud of Google Cloud Platform to manage the Kubernetes infrastructure using Google Kubernetes Engine (GKE) as a tool for container management on the cloud. It facilitated the process of building, deploying, and scaling services. For Docker Swarm, VM instances were created on Google Container Engine. On both container platforms, two types of workloads, NGINX (a general web service) and VITE (a video content web service with a Full HD resolution of 1080p), were installed to compare the CPU performance of each service. The services were set to a standby state, where they would operate as a single service if the CPU usage was below 80%. However, if the CPU usage exceeded 80%, the services would scale up to accommodate the increased workload. The maximum scalability was set to five services to efficiently handle user operations.

After conducting a comparative test on both platforms, it was found that the operation of Kubernetes is more convenient as it allows for the automatic scaling of services and achieves faster scalability, up to 93%. On the other hand, Docker Swarm does not have auto-scaling functionality and relies on system administrators for monitoring. Manual scaling is required, resulting in significant delays. In conclusion, Kubernetes has the advantage of faster scalability compared to Docker Swarm.

Keywords: Scaling, Container, Web Service

.....

Advisor

กิตติกรรมประกาศ

สารนิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องจากได้รับความกรุณาอย่างสูงจากอาจารย์ ดร.ธนัญ จารุวิทยโกวิท อาจารย์ที่ปรึกษางานวิจัย ที่กรุณาให้คำแนะนำ แนวคิด ตรวจสอบ แก้ไข ตลอดจนชี้แนะข้อบกพร่องต่างๆ ด้วยความเอาใจใส่อย่างดียิ่งมาโดยตลอด จนงานวิจัยสำเร็จลุล่วงได้ด้วยดี ผู้จัดทำงานวิจัยนี้ตระหนักถึงความตั้งใจจริงและความทุ่มเทของอาจารย์และกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้

ผู้จัดทำงานวิจัยหวังเป็นอย่างยิ่งว่างานวิจัยฉบับนี้จะมีประโยชน์ไม่น้อย จึงขอมอบส่วนดีทั้งหมดให้แก่เหล่าคณาจารย์ที่ได้ประสิทธิประสาทวิชาจนทำให้ผลงานวิจัยนี้เป็นประโยชน์ต่อผู้ที่เกี่ยวข้อง และขอมอบความกตัญญูกตเวทิตาคุณแด่บิดามารดาและผู้มีพระคุณทุกท่าน สำหรับข้อบกพร่องต่าง ๆ ที่อาจเกิดขึ้น ผู้วิจัยขอน้อมรับไว้แต่เพียงผู้เดียวและยินดีที่จะรับฟังคำแนะนำจากทุกท่านที่ได้เข้ามาศึกษาเพื่อเป็นประโยชน์ในการพัฒนางานวิจัยต่อไป

วรเทพ อหันทริก

สารบัญ

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

เทคโนโลยีสารสนเทศ (Information Technology) เป็นเครื่องมือสำคัญประการหนึ่งในการดำเนินธุรกิจของหน่วยงานบริษัท หรือผู้ให้บริการการประยุกต์ใช้เทคโนโลยีสารสนเทศได้อย่างเหมาะสมจะส่งผลให้การดำเนินการมีประสิทธิภาพเพิ่มมากขึ้น สามารถลดระยะเวลาในการทำางานได้อย่างมีนัยยะสำคัญ ลดการใช้ทรัพยากร ลดการใช้พลังงานไฟฟ้า แต่จะสามารถเพิ่ม ความง่าย ความสะดวกในการใช้งานและเพิ่มผลผลิตได้มากยิ่งขึ้น

คอนเทนเนอร์จึงถูกสร้างขึ้นมาเพื่อแก้ปัญหาข้างต้น โดยมีฮาร์ดแวร์และ OS เพียงชุดเดียวกัน ลดความซ้ำซ้อนของการใช้ทรัพยากรลง ส่วนตัวแอพพลิเคชันและซอฟต์แวร์ซึ่งเป็นจุดที่แตกต่างกันไปก็จะมี "container" (เทียบได้กับ VM) มาครอบเพื่อแบ่งส่วนทรัพยากรไว้ไม่ให้ยุ่งกัน จุดเด่นของคอนเทนเนอร์จึงเป็นเรื่องการใช้ทรัพยากรที่น้อยกว่า virtualization มาก อิมเมจของคอนเทนเนอร์อาจมีขนาดเพียงไม่กี่สิบ MB ในขณะที่อิมเมจของ VM ต้องใช้พื้นที่ระดับหลาย GB นอกจากนี้ระยะเวลาที่ใช้บูต, พลังซีพียู และปริมาณแรมที่ต้องใช้ก็ลดลงตามไปด้วย ส่งผลให้เซิร์ฟเวอร์หนึ่งเครื่องสามารถยัดคอนเทนเนอร์จำนวนมากกว่าการรัน VM ที่ให้ผลแบบเดียวกันถึง 2-3 เท่าตัว บางครั้งคอนเทนเนอร์ถูกเรียกชื่อในทางเทคนิคว่า Operating-system-level virtualization หรือการสร้าง VM ที่ระดับ OS โดยเราไม่ต้องสร้างเครื่องคอมพิวเตอร์เสมือนขึ้นมาทั้งตัว

1.2 ปัญหาและแรงจูงใจ

ปัจจุบันองค์กรใดที่มีการใช้งานเครื่อง Server ก็จะต้องมีการลงทุนในระบบนี้ไปไม่น้อย และเครื่อง Server ก็เป็นสิ่งหนึ่งที่ทำหน้าที่เป็นศูนย์ข้อมูลขององค์กรทั้งหมด ทั้งการเก็บข้อมูลเว็บไซต์ เก็บข้อมูลสำคัญต่าง ๆ ในการทำงาน รวมถึงจะต้องมีการใช้งานเป็นศูนย์กลางถ่ายโอนข้อมูลทั้งภายในและภายนอกอยู่เสมอ จึงเป็นสิ่งสำคัญมากในระบบการทำงานในปัจจุบัน

ปัญหาที่พบบ่อยในการใช้งาน

1.2.1 ปัญหาการทำ Web Application แต่การออกแบบระบบในตอนแรกไม่ได้ออกแบบไว้ให้ผู้ใช้ Request เข้ามาใช้งานพร้อม ๆ กันเป็นจำนวนมาก พอระบบเริ่มขยายมีคนเข้ามาใช้งานจำนวนมาก ๆ Web Server จึงรับไม่ไหว จึงทำให้ Server ล่มได้

1.2.2 ปัญหา Internet ช้าเป็นเรื่องที่พบเจอ โดยเกิดจากระบบเครือข่ายโดยตรง ไม่ว่าจะเป็นระบบที่ออกแบบมาไม่ดี การใช้อุปกรณ์ไม่มีประสิทธิภาพเนื่องจากงบประมาณหรือการประเมินสเปคผิดพลาด หรือมีการดาวน์โหลดไฟล์เยอะเกินไป

1.2.3 การเลือกใช้ Server แบบประกอบเองทำให้อุปกรณ์ที่ใช้ไม่ได้มาตรฐานที่ดี

1.2.4 ปัญหาการจัดทำระบบ Database ไม่สอดคล้อง เพราะเมื่อระบบ Input และ Output ประมวลผลไม่ทันก็เกิดการจัดการระบบข้อมูลไม่ได้ ซึ่งเป็นอีกสาเหตุที่ทำให้ระบบล่ม

1.2.5 ปัญหาระบบภายในเครื่อง Server จะใช้เวลานานในการตรวจสอบสถานะของ server และการตรวจพบปัญหา

เพื่อลดปัญหาการใช้งานจึงได้เลือกใช้คอนเทนเนอร์เพื่อประโยชน์ดังนี้

1.2.6 ใช้ Resource น้อยกว่า เนื่องจากโครงสร้างของ Containers ไม่จำเป็นต้องมี Operating System image เหมือนกับ VM ทำให้ใช้งาน Resources ไม่ได้มากเท่า

1.2.7 เพิ่มช่องทางในการ Run Application เนื่องจาก Application ที่ Run อยู่บน Containers สามารถ Deploy ไปบน OS และ Hardware อะไรก็ได้ (อาจไม่ Support ทั้งหมด แต่ Support เป็นส่วนใหญ่)

1.2.8 Operation ทำงานได้อย่างราบรื่น เพราะ Application ที่อยู่บน Containers จะทำงานได้เหมือนเดิม ไม่ว่าจะ Deploy ไปไว้บนไหนก็ตาม

1.2.9 ให้ประสิทธิภาพการทำงานดี เพราะว่าสามารถ Deploy ได้ไว Update Patch ได้ง่ายไม่กระทบการทำงานส่วนอื่นและยัง Scale ได้ง่าย หาก Resource ไม่เพียงพอ

1.3 วัตถุประสงค์ของการวิจัย

1.3.1 เพื่อทดสอบและเปรียบเทียบประสิทธิภาพของการทำงานของการขยายตัวของคอนเทนเนอร์ 2 แพลตฟอร์ม

(1) Kubernetes

(2) Docker Swarm

1.3.2 เพื่อเปรียบเทียบความแตกต่างระหว่าง CPU load ของ 2 Service คือ NGINX และ Web test ที่เขียนด้วย React VITE

1.4 ขอบเขตของการวิจัย

งานวิจัยนี้จะทำการทดสอบบนระบบคอนเทนเนอร์คลัสเตอร์ Kubernetes และ Docker Swarm ที่ทำงานบน Public Cloud (Google Cloud Platform) โดยจะเปรียบเทียบอุปกรณ์หลักของระบบ เช่น CPU และการขยายตัวของ Web Service NGINX และ Web test ที่เขียนด้วย React VITE โดยมีรายละเอียดขอบเขตงานวิจัยดังนี้

1.4.1 ใช้ระบบ Kubernetes รุ่นเสถียร รหัสรุ่น 1.24.12 (GKE)

1.4.2 ใช้ระบบ Docker Engines รุ่น 24.0.2

1.4.3 ใช้ระบบ Container Runtime เป็น Docker

1.4.4 Public Cloud ที่ใช้ในการทดสอบใช้บริการจาก Google Cloud Platform (GCP) โซนของประเทศสิงคโปร์ รหัสโซน asia-southeast1-a และ asia-southeast2-a

1.4.5 คอมพิวเตอร์ Workstation ติดตั้งระบบปฏิบัติการ Ubuntu Linux Server 18.04.5

1.4.6 ซอฟต์แวร์ที่ใช้ในการทดสอบใช้โปรแกรม wrk

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ได้ทราบถึงวิธีการทำงานและประสิทธิภาพของการทำงานของการขยายตัวบนคอนเทนเนอร์

1.5.2 สามารถนำข้อมูลผลการทดสอบไปประกอบการตัดสินใจเลือกใช้การขยายตัวของคอนเทนเนอร์ได้อย่างเหมาะสมกับระบบที่จะให้บริการ

1.5.3 สามารถนำความรู้ทั้งหมดจากงานวิจัยชิ้นนี้มาประยุกต์ใช้งานจริง

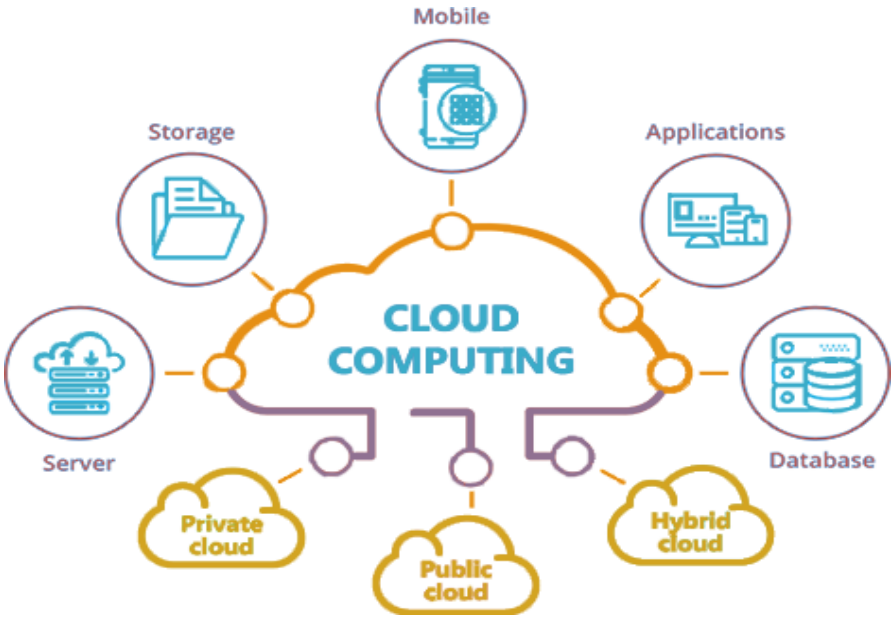
บทที่ 2
แนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องและนำมาประยุกต์ใช้ในงานวิจัย ซึ่งประกอบด้วย การประมวลผลแบบกลุ่มเมฆ (Cloud Computing), การประมวลผลแบบคอนเทนเนอร์ (Container), ระบบจัดการคอนเทนเนอร์คูเบอร์เนทิส (Kubernetes), ระบบจัดการคอนเทนเนอร์ด็อกเกอร์ สวอม (Kubernetes), การสเกล (Scale Up/Down)

2.1 ความรู้พื้นฐาน

2.1.1 การประมวลผลแบบกลุ่มเมฆ (Cloud computing) [1]

Cloud Computing เป็นเสมือนคอมพิวเตอร์เครื่องหนึ่งที่เราใช้งานกันอยู่คือ มีทั้งฮาร์ดแวร์และซอฟต์แวร์ที่ใช้ในการประมวลผลและเก็บข้อมูล แต่ Cloud Computing เป็นระบบคอมพิวเตอร์ที่ใหญ่มากๆ รองรับการใช้งาน การประมวลผล ตลอดจนการจัดเก็บข้อมูลได้อย่างมหาศาลตามภาพที่ 2.1



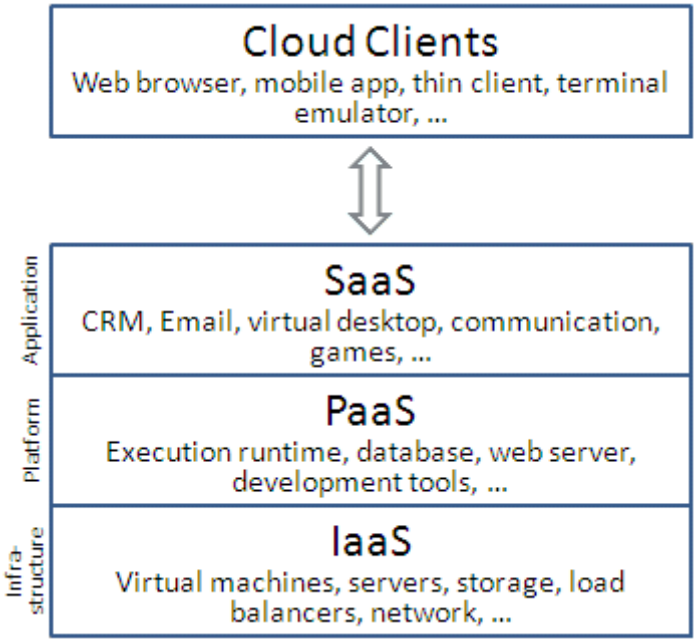
ภาพที่ 2.1 การประมวลผลแบบกลุ่มเมฆ [1]

Cloud Computing สามารถจำแนกตามการใช้งานได้ 3 ประเภท

- Infrastructure as a Service (IaaS) เป็นโครงสร้างพื้นฐานเหมือนกับระบบคอมพิวเตอร์คือ มีทั้งหน่วยประมวลผล ระบบเครือข่าย และพื้นที่จัดเก็บข้อมูลให้ใช้งาน

- Platform as a Service (PaaS) เป็นการใช้งานเกี่ยวกับแพลตฟอรม์ต่าง ๆ เช่น การพัฒนาเว็บแอปพลิเคชั่นหรือโมบายแอปพลิเคชั่น เป็นต้น

- Software as a Service (SaaS) เป็นการใช้งานทางด้านซอฟต์แวร์ ซึ่งภายใต้ระบบหรือเทคโนโลยี Cloud Computing จะมีซอฟต์แวร์เฉพาะด้าน เช่น ซอฟต์แวร์ทางบัญชี ซอฟต์แวร์การบริหารจัดการโรงแรม และอื่นๆ ซึ่งจำเป็นสำหรับธุรกิจประเภทต่างๆ ให้ใช้งาน



ภาพที่ 2.2 ประเภทของการประมวลผลแบบกลุ่มเมฆ [1]

สำหรับงานวิจัยนี้ได้นำ Cloud Computing มาใช้ทำหน้าที่เผื่อจัดการโครงสร้างพื้นฐานของประมวลผล ระบบเครือข่าย และพื้นที่จัดเก็บข้อมูล

2.1.2 การประมวลผลแบบคอนเทนเนอร์ (Container) [2]

Containers คือการจัดเก็บแอปพลิเคชันให้อยู่ในรูปแบบที่ง่ายต่อการโยกย้ายและนำขึ้นระบบการปรับใช้โดยประกอบไปด้วยตัวแอปพลิเคชัน library หรือ binary จำเป็นในการรันแอปพลิเคชันนั้นและการกำหนดค่าของแอปพลิเคชันนั้น ไม่ว่าจะย้ายตัวแอปพลิเคชันที่ถูกจัดเก็บเป็น container ไปรันที่ไหนจะสามารถทำงานได้เหมือนเดิมโดยไม่คำนึงถึงระบบปฏิบัติการ Container เป็นเทคโนโลยีการจำลองเสมือนของระบบปฏิบัติการคอมพิวเตอร์ เช่น CPU Memory แต่ละครเทนเนอร์ทำหน้าที่เป็นแพ็กเกจแยกต่างหากซึ่งสามารถเรียกใช้แอปพลิเคชันหรือบริการได้โดยไม่ต้องคำนึงถึงสภาพแวดล้อมการประมวลผลพื้นฐานภายในหนึ่ง container สามารถรันแอปพลิเคชันทั้งตัวหากเราแบ่งแบบ microservice ก็ได้แอปพลิเคชันในรูปแบบ container จะถูกรันผ่าน software ที่เรียกว่า container engine/runtime

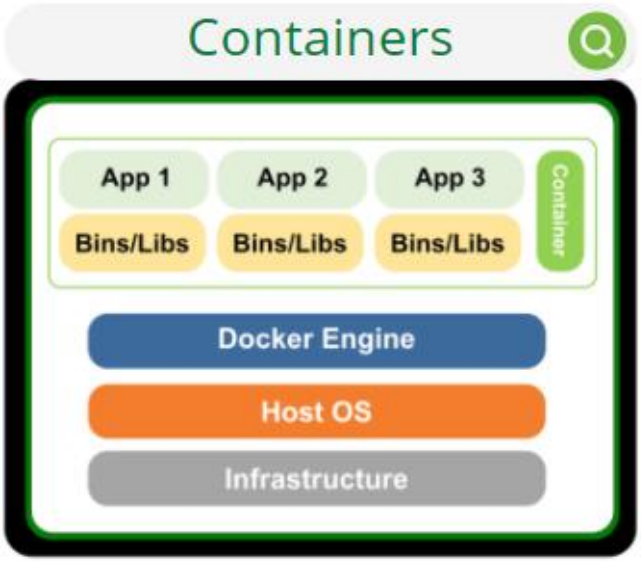
Containers จะมีชั้นของแอปพลิเคชันที่เก็บอยู่รวมกันมีแพคเกจหรือชุดคำสั่งที่ใช้ทำงานร่วมกัน

Containers ตั้งแต่หนึ่งตัวขึ้นไปสามารถรันบนคอมพิวเตอร์เครื่องเดียวกันและใช้เคอร์เนลของระบบปฏิบัติการร่วมกันกับ container ตัวอื่น ๆ

Containers แต่ละตัวทำงานหรือประมวลผลแยกในพื้นที่ของตัวเอง

Containers ใช้พื้นที่น้อยกว่า Virtual Machine ไฟล์อิมเมจมีขนาดเล็ก หลัก เมกกะไบต์

Containers สามารถจัดการแอปพลิเคชันได้มากขึ้น ใช้จำนวน Virtual Machine และจำนวนระบบปฏิบัติการน้อยลงดังภาพที่ 2.3



ภาพที่ 2.3 การประมวลผลแบบคอนเทนเนอร์ [2]

สำหรับงานวิจัยนี้ได้นำ Container มาใช้ทำหน้าที่เผื่อจัดการ library ของ Application

2.1.3 Docker Swarm [3]

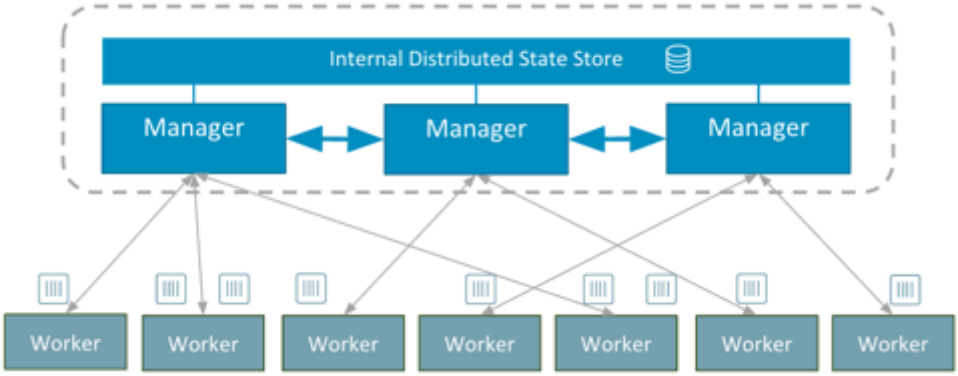
Docker swarm เป็น Native Clustering ของ Docker คือเป็นเครื่องมือช่วยจัดการเครื่อง server ที่รัน Docker หลายๆ เครื่องให้อยู่ในสภาพแวดล้อมเดียวกัน พูดง่ายๆ ก็คือ การนำเอาเครื่อง server หลายๆ เครื่อง (Worker) มาช่วยกันทำงาน โดยจะถูกควบคุมการทำงานโดย Manager และยังมีระบบ IPVS ที่เป็น Load-balance ซึ่งจะทำให้เราสามารถเข้าถึง Website หรือ Application ต่างๆ ที่เรารันอยู่ได้จากเครื่องไหนก็ได้ใน Swarm โดย IPVS จะช่วยจัดการให้เราเองโดยอัตโนมัติ ศัพท์ที่เกี่ยวกับ Docker swarm ที่ควรรู้มีดังนี้

Manager คือ เครื่อง Server ที่ทำหน้าที่เป็นตัวกลางที่คอยควบคุมให้ เครื่อง Server ที่รัน Docker เครื่องอื่นๆ ทำงานร่วมกันได้ และเป็นตัวกระจายงานให้กับเครื่องอื่นๆ Manager สามารถมีได้หลายเครื่อง (แนะนำให้มีเป็นเลขคี่)

Worker คือ เครื่อง server ที่จะคอยรับงานจาก Manager

Service คือ Application ที่เราต้องการให้ทำงาน โดยภายในจะถูกแบ่งเป็น task (อาจจะเรียกว่า Container ก็ได้) ตามจำนวนที่เรากำหนด

Task คือ พูดง่ายๆ ก็คือ Container นั่นแหละครับ Task นี้จะถูกกระจายไปยัง Worker แต่ละเครื่อง ตามโครงสร้าง Swarm Model ดังภาพที่ 2.4



ภาพที่ 2.4 โครงสร้างด็อกเกอร์ สวอม [3]

สำหรับงานวิจัยนี้ได้นำ Docker Swarm มาใช้ในการจัดการเครื่อง Server ที่รัน Docker หลายเครื่อง และใช้ในการ Scale ของ Service

2.1.4 Kubernetes [4]

Kubernetes คือ container orchestration engine จุดประสงค์หลักของ Kubernetes คือการจัดการ container deployments ในปัจจุบันทุกๆ application กำลังจะใช้ Microservices architecture มากกว่า Monolithic architecture และวิธีการใช้งาน microservices architecture เหล่านั้นจะออกแบบโดย containerization technology ตามที่เรารู้ Docker ปฏิวัติ container technology ทุกคนใช้ containers ในการ build their applications ในตอนนี้ และวิธีการ manage containers at a large scale, คือเราใช้ Kubernetes

Kubernetes ถูกออกแบบและพัฒนาโดย Google และปัจจุบัน Project ถูกดูแลโดย Cloud Native Computing Foundation

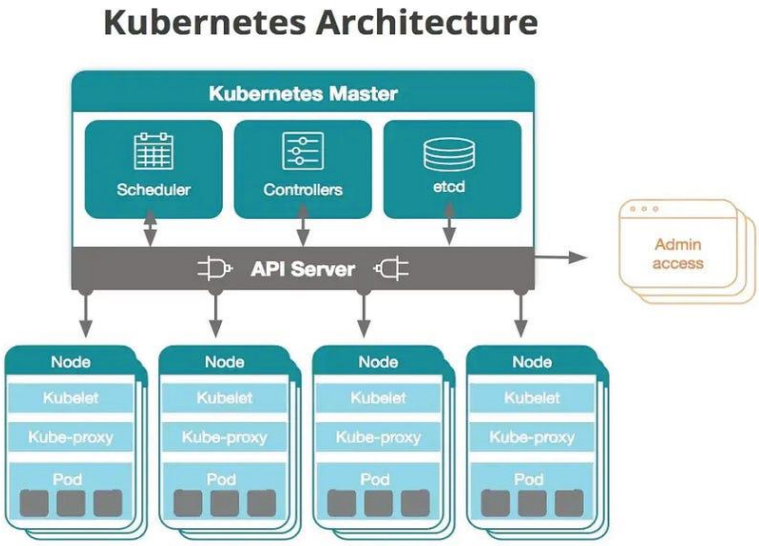
หน้าที่หลักๆ ของ Kubernetes

Scaling Deployments

Load Balancing

Monitoring

Automated rollouts และ rollbacks



ภาพที่ 2.5 โครงสร้างของคูเบอร์เนเตส [4]

จากภาพที่ 2.5 จะประกอบด้วย Master และ Worker Node โดย Master Node จะมี

1. Kube-API-Server เป็น API กลางของ Kubernetes Cluster เพื่อรับคำสั่งจาก Developers ว่าจะ Deploy apps อะไร ทำอะไรกับ apps นั้นๆ

2. Kube-Scheduler ทำหน้าที่จัดการว่า apps ควรจะไป Deploy ที่ Worker Nodes, Node ไหน

3. ETCD ทำหน้าที่เก็บ State ของ Kubernetes Cluster ว่า Cluster ทำงานเป็นอย่างไร มี Apps อะไรอยู่บ้าง

4. Kube Controller Manager ทำหน้าที่ควบคุม State ของ Kubernetes Cluster

5. Cloud Controller Manager ใช้ติดต่อกับ Cloud Providers เพื่อจัดการ Worker Nodes / Load Balancers Worker Node จะมี

- Kube Proxy (จะอยู่ใน Master Node ด้วย) หน้าที่เขียน iptables rules หรือ routing tables ให้ apps ที่ Deploy ลงบน Kubernetes คุยกันเองได้

- Kubelet ทำหน้าที่รับคำสั่งจาก Kube-API-Server เพื่อสั่ง App Deployment บน Worker Node

Kubernetes Resources หลักๆ ที่ใช้งานกันมีอะไรบ้าง:

1. Pods เป็นหน่วยเล็กที่สุดที่เราสามารถนำ Apps มาใส่ และ Deploy as container บน Kubernetes ได้

2. Deployments ทำหน้าที่ Make sure ว่า Pod ของ App นั้นๆ มี Replicas ตามที่เรากำหนดไว้แน่ๆ และช่วยในการทำให้ Pod ของ App นั้นๆ สามารถ Rolling Update ได้ เพื่อลดการ Downtime ของ App

3. Stateful Sets ช่วยให้เราสามารถ Deploy app แบบเป็น order ได้ เช่น app01, app02, app03 และ Storage ก็จะมีการ Mounting แบบ Redundancy

4. Jobs ช่วยในการ Run Job สั้นๆ บน Kubernetes เช่น DB Migration (Flyway), Golang-migrate

5. HPA ช่วยในการทำ Auto scale Replicas ของ Pods ตาม Metrics ที่เรากำหนดไว้ ซึ่งเราอาจจะนำ Metrics มาจาก Prometheus ได้เช่นกัน โดยใช้ Prometheus Adapter

สำหรับงานวิจัยนี้ได้นำ Kubernetes มาใช้ในการจัดการเครื่อง Server ที่รัน Worker กับ Master หลายเครื่อง และใช้ในการ Auto-Scale ของ Service

2.1.5 NGINX [5]

NGINX คือ ซอฟต์แวร์โอเพนซอร์สสำหรับ Web service แบบพร็อกซีย้อนกลับ การโหลดบาลานซ์สตรีมมิ่งสื่อและอื่น ๆ NGINX เป็นเว็บเซิร์ฟเวอร์ที่ออกแบบมาเพื่อประสิทธิภาพ และความเสถียรสูงสุด นอกเหนือจากความสามารถของเซิร์ฟเวอร์ HTTP แล้ว NGINX ยังสามารถทำหน้าที่เป็นพร็อกซีเซิร์ฟเวอร์สำหรับอีเมล (IMAP, POP3 และ SMTP) และพร็อกซีแบบย้อนกลับและ balancer โหลดสำหรับเซิร์ฟเวอร์ HTTP, TCP และ UDPIgor Sysoev ได้เขียน NGINX เพื่อแก้ปัญหา C10K ซึ่งเป็นคำจำกัดความในปี 1999 เพื่ออธิบายความยากลำบากที่เว็บเซิร์ฟเวอร์ที่มีอยู่มีประสบการณ์ในการจัดการกับจำนวนมาก (10K) ของการเชื่อมต่อที่เกิดขึ้นพร้อมกัน (C) ด้วยสถาปัตยกรรมแบบอะซิงโครนัสที่อิงกับเหตุการณ์ NGINX ปฏิวัติวิธีที่เซิร์ฟเวอร์ทำงานในบริบทที่มีประสิทธิภาพสูงและกลายเป็นเว็บเซิร์ฟเวอร์ที่เร็วที่สุดที่มีอยู่

หลังจากเปิดโครงการในปี พ.ศ. 2547 และเฝ้าดูการใช้งานที่เพิ่มขึ้นเรื่อย ๆ Sysoev ร่วมก่อตั้ง NGINX, Inc. เพื่อสนับสนุนการพัฒนา NGINX อย่างต่อเนื่อง และเข้าสู่ NGINX Plusในฐานะผลิตภัณฑ์เชิงพาณิชย์ที่มีคุณลักษณะเพิ่มเติมที่ออกแบบมาสำหรับลูกค้าองค์กร วันนี้ NGINX และ NGINX Plus สามารถจัดการการเชื่อมต่อพร้อมกันได้นับแสนครั้งและใช้พลังงานมากกว่า 50% ของไซต์ที่คึกคักที่สุดบนเว็บ

สำหรับงานวิจัยนี้ได้นำ Nginx มาใช้ในเป็น Web service บน container เผื่อเปรียบเทียบประสิทธิภาพการ Scale

2.1.6 React VITE [6]

React เป็น JavaScript Library หรือจะเรียกว่าเป็น JavaScript Framework ก็ได้ที่เราใช้สำหรับสร้างหน้าเว็บของเราให้ออกมาเจ๋งและแจ๋ว พร้อมด้วย action ต่างๆ ที่ทำให้เว็บของเราดูน่าสนใจนั่นเองครับ

จุดเด่นของ React ที่ทำให้มันน่านำมาใช้งานนั้นก็คือ มันมีระบบแคชในตัวทำให้หน้าเว็บของเรามีการตอบสนองที่เร็ว เหมาะแก่การนำไปทำ SPA เป็นอย่างยิ่งนั่นเอง การเขียน React เรายังสามารถแยก

องค์ประกอบของหน้าเว็บเราออกเป็นส่วนๆเรียกว่าเป็น component แล้วนำมาประกอบกันเป็นหน้าเว็บได้ ซึ่งทำให้เราสามารถนำ component ของเราไปใช้ซ้ำที่อื่นได้ ไม่ต้องเสียเวลาเขียนใหม่นั่นเองครับ

สำหรับงานวิจัยนี้ได้นำ React มาใช้เจียนหน้าเว็บออกมาเผื่อเปรียบเทียบ Web service กับ Nginx บน container เผื่อเปรียบเทียบประสิทธิภาพการ Scale

2.2 งานวิจัยที่เกี่ยวข้อง (Related Work)

2.2.1 การสเกลพอดแกนแนวนอนด้วยปฏิทินออนไลน์สำหรับคูเบอร์เนติส [7]

การพัฒนาระบบคูเบอร์เนติสให้มีคุณสมบัติจัดเตรียมพอดได้ล่วงหน้าผ่านระบบปฏิทินออนไลน์ ซอฟต์แวร์มีคุณสมบัติอ่านกำหนดการจากปฏิทินออนไลน์และส่งข้อมูลควบคุมทรัพยากรคูเบอร์เนติสผ่านโพรโทคอลเอ็มคิวทีที อันประกอบด้วยจำนวนผู้ใช้และกำหนดวันนัดหมาย เมื่อระบบได้รับข้อมูลจะทำการขยายพอดโดยอัตโนมัติตามกำหนดนัดหมายของปฏิทินออนไลน์จากการทดลองพบว่าเมื่อถึงกำหนดนัดหมายแล้ว ระบบจะเริ่มสร้างพอดโดยอัตโนมัติภายในระยะเวลาน้อยกว่า 10 นาที

ข้อดี ระบบมีการขยายพอดโดยอัตโนมัติตามกำหนดนัดหมายของปฏิทินออนไลน์

2.2.2 การขยายขนาดการให้บริการแบบอัตโนมัติในระบบด็อกเกอร์ [8]

การพัฒนาสถาปัตยกรรมการขยายขนาดการให้บริการอัตโนมัติในระบบด็อกเกอร์ที่ประกอบด้วยส่วนการเฝ้าระวัง ส่วนการตัดสินใจ และส่วนการขยายขนาดการให้บริการโดยใช้ค่าของการใช้งานหน่วยประมวลผลมาเป็นค่าที่ใช้ในการตัดสินใจว่าจะขยายขนาดหรือลดขนาดและมีการทดสอบการทำงานและทดสอบความพร้อมในการให้บริการของกลไกที่ใช้ในงานวิจัยนี้ โดยกลไกสามารถลดและขยายขนาดการให้บริการได้ตามที่ผู้ใช้งานตั้งค่าไว้โดยมีข้อผิดพลาดจากการทำงานเล็กน้อย

ข้อดี การขยายขนาดการให้บริการโดยใช้ค่าของการใช้งานหน่วยประมวลผลมาเป็นค่าที่ใช้ในการตัดสินใจว่าจะขยายขนาดหรือลดขนาด

ข้อจำกัด กลไกการขยายขนาดการให้บริการอัตโนมัติในระบบด็อกเกอร์นั้น มีความสามารถที่จะให้บริการได้โดยมีความผิดพลาดเฉพาะระหว่างการลดขนาดของด็อกเกอร์คอนเทนเนอร์เพียงเล็กน้อย

2.2.3 ระบบการปรับขนาดอัตโนมัติสำหรับเกตเวย์ API ที่อิงตาม Kubernetes [9]

ระบบเอพีไอเกตเวย์เป็นทางเข้าสู่บริการแบ็กเอนด์สามารถลดปริมาณการติดต่อระยะไกลระหว่างแอปพลิเคชันและบริการแบ็กเอนด์ได้อย่างมีประสิทธิภาพ และลดความซับซ้อนของบริการภายในที่ติดต่อหากัน โดยออกแบบระบบปรับขนาดอัตโนมัติสำหรับระบบเอพีไอเกตเวย์บนคูเบอร์เนติสและPrometheus ซึ่งสามารถปรับขนาดจำนวนแอปพลิเคชันได้ตลอดเวลา ปรับปรุงการใช้ทรัพยากรระบบในขณะที่บริการของแอปพลิเคชันมีคุณภาพและมีความพร้อมให้บริการ

ข้อดี สามารถปรับขนาดจำนวนแอปพลิเคชันได้ตลอดเวลา ปรับปรุงการใช้ทรัพยากรระบบในขณะที่บริการของแอปพลิเคชันมีคุณภาพและมีความพร้อมให้บริการ

ตารางที่ 2.1 ตารางเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับงานวิจัยที่นำเสนอ

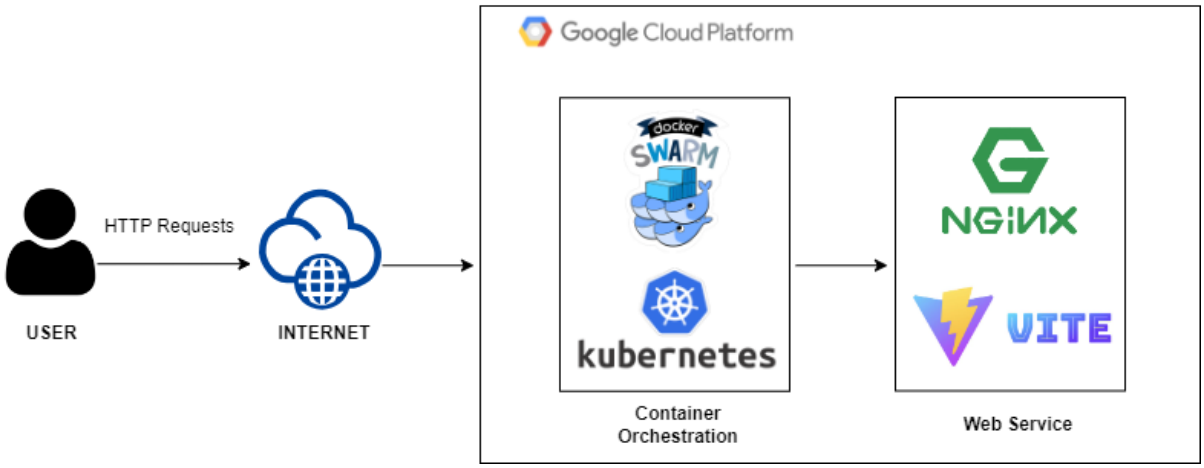
ลำดับ	รายละเอียดของระบบ	งานวิจัยที่ 1	งานวิจัยที่ 2	งานวิจัยที่ 3	งานวิจัยที่นำเสนอ
1	การขยายพอดโดยอัตโนมัติ โดย Kubernetes	✓	X	✓	✓
2	การกระจายงานหรือเรียกว่าโหลดบาลานซ์เซอร์	✓	✓	✓	✓
3	ใช้งานหน่วยประมวลผลมาเป็นค่าที่ใช้ในการตัดสินใจว่าจะขยายขนาดหรือลดขนาด	✓	✓	X	✓
4	การใช้ Docker Swarm ขยายอัตโนมัติ	X	✓	X	X
5	มีการเปรียบเทียบโดยใช้ Web Service และเปรียบเทียบใช้ 2 คอนเทนเนอร์	X	X	X	✓

บทที่ 3
ระเบียบวิธีวิจัย

งานวิจัยนี้จะทำการทดสอบบนระบบคอนเทนเนอร์คลัสเตอร์ Kubernetes และ Docker Swarm ที่ทำงานบน Public Cloud (Google Cloud Platform) โดยจะเปรียบเทียบอุปกรณ์หลักของระบบ เช่น CPU และการขยายตัวของ Web Service NGINX และ Web test ที่เขียนด้วย React VITE เพื่อศึกษาและเป็นแนวทางในการพัฒนาระบบดังกล่าว จะช่วยอำนวยความสะดวกให้กับผู้ที่ดูแลระบบ

3.1 การวิเคราะห์และออกแบบระบบ

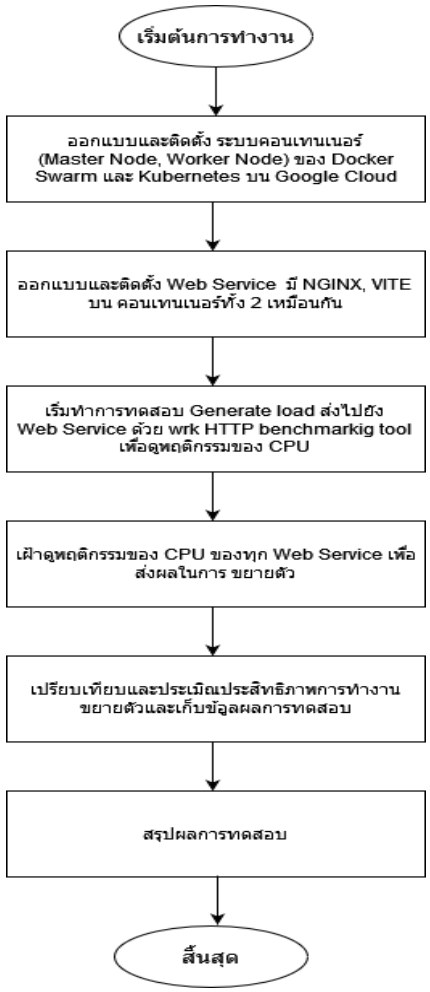
องค์ประกอบของการทำงานของการเปรียบเทียบการขยายตัวจะมีภาพรวมการทำงานตั้งแต่มีการเรียกใช้งานจากผู้ใช้ด้วย HTTP Request ผ่าน Internet เข้าไปที่คอนเทนเนอร์ที่อยู่บน Public Cloud และไปถึง Web Service ที่อยู่บนคอนเทนเนอร์แสดงได้ดังภาพที่ 3.1



ภาพที่ 3.1 แสดงโครงสร้างการทำงานของระบบ

3.2 ขั้นตอนและวิธีการดำเนินงาน

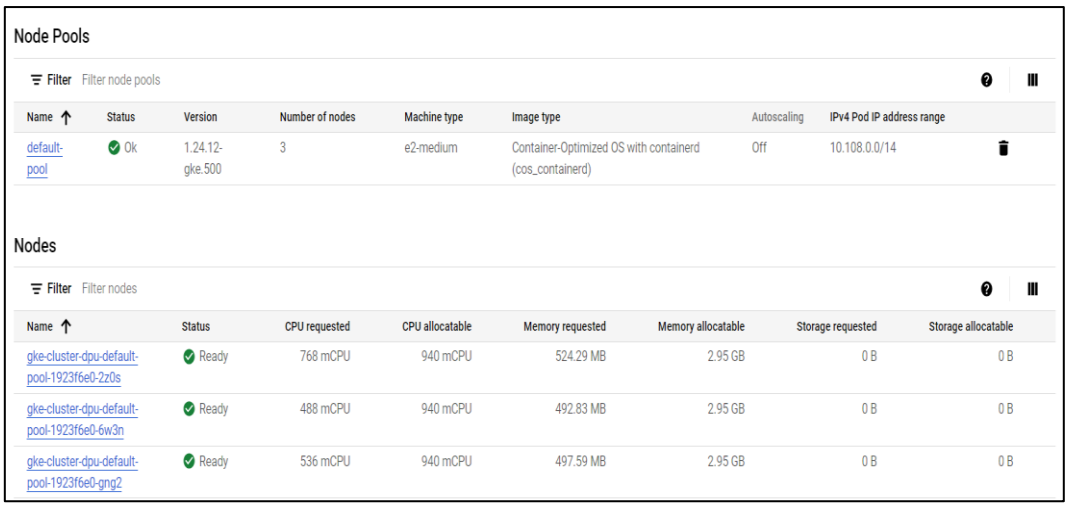
รายละเอียดรูปแบบการทดสอบจะมีอธิบายให้เห็นในรูปแบบ Flow Chart เพื่อแสดงกระบวนการของระบบดังแสดงในภาพที่ 3.2



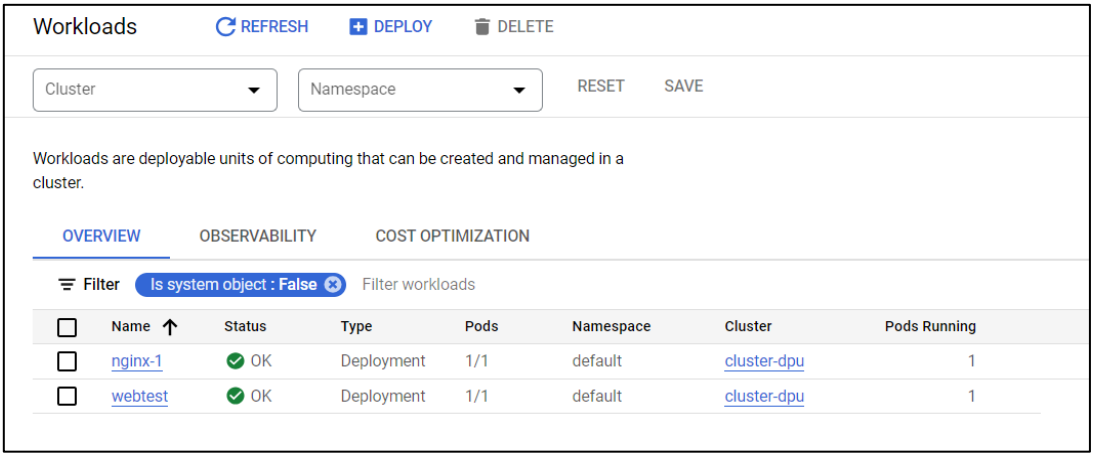
ภาพที่ 3.2 แผนผังขั้นตอนการดำเนินงานทดสอบระบบ

จากภาพที่ 3.2 จะมีรายละเอียดดังนี้

การดำเนินการออกแบบโดยคอนเทนเนอร์ที่ 1 จะสร้างระบบคอนเทนเนอร์ Kubernetes ขึ้น โดยใช้อยู่บน Google cloud platform ประกอบไปด้วยคลัสเตอร์ที่มี Master Node และ Worker Node ดังภาพที่ 3.3

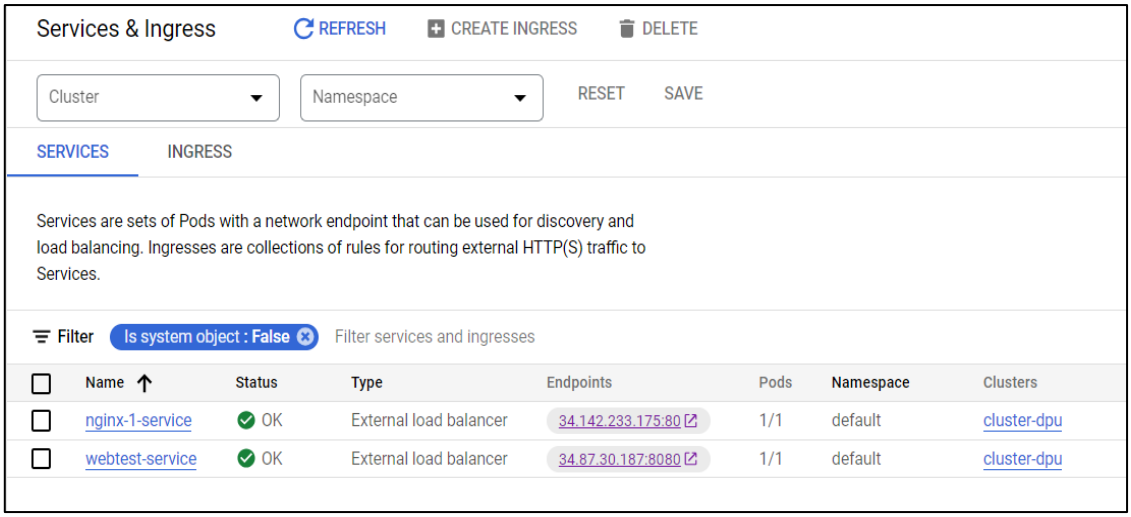


ภาพที่ 3.3 แสดงโหนดของระบบคูเบอร์เนทิส



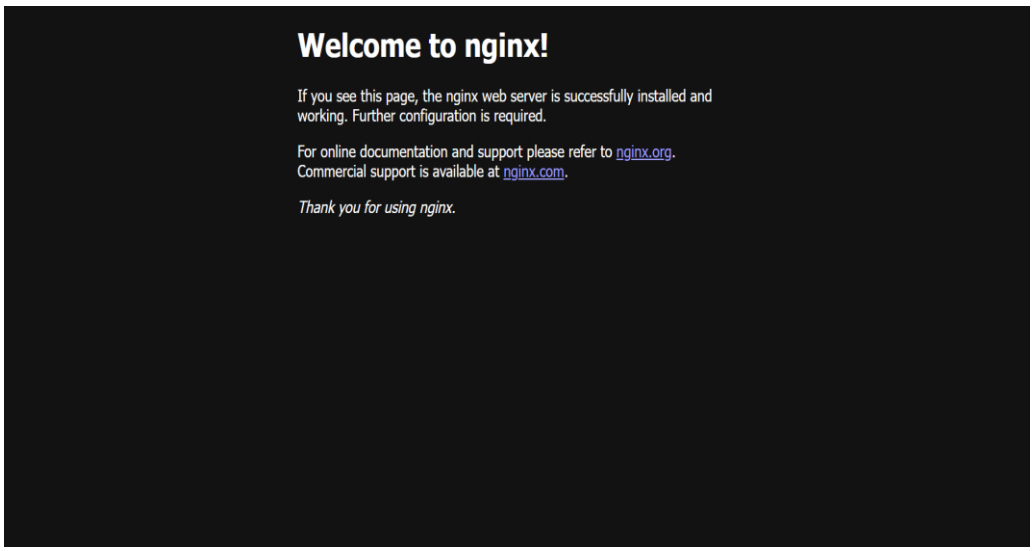
ภาพที่ 3.4 แสดงเว็บแอปพลิเคชันที่ติดตั้งบนคูเบอร์เนทิส

จากภาพที่ 3.4 การติดตั้งเว็บแอปพลิเคชันเพื่อใช้ทดสอบทั้ง 2 ตัวบน Kubernetes คือ เว็บแอปพลิเคชันโดยใช้ NGINX และเว็บแอปพลิเคชันโดยใช้ React VITE

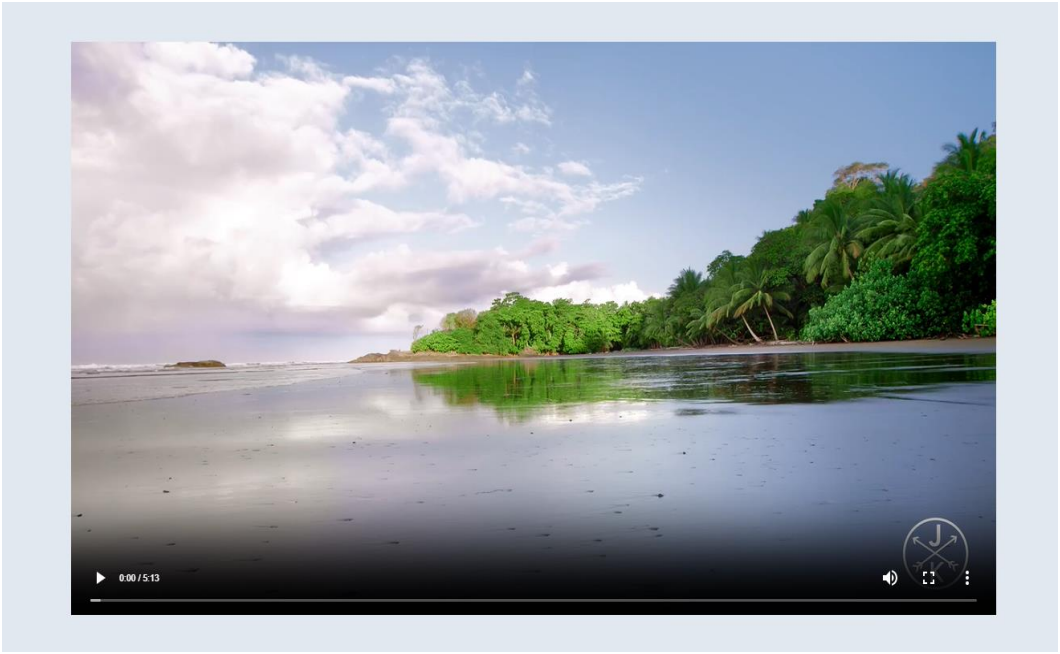


ภาพที่ 3.5 แสดงโหลดบาลานซ์ของเว็บแอปพลิเคชันบนคูเบอร์เนทิส

จากภาพที่ 3.5 จะเป็นการตั้งค่าโหลดบาลานซ์ของเว็บแอปพลิเคชันเพื่อที่จะให้เข้าถึงจากภายนอกได้

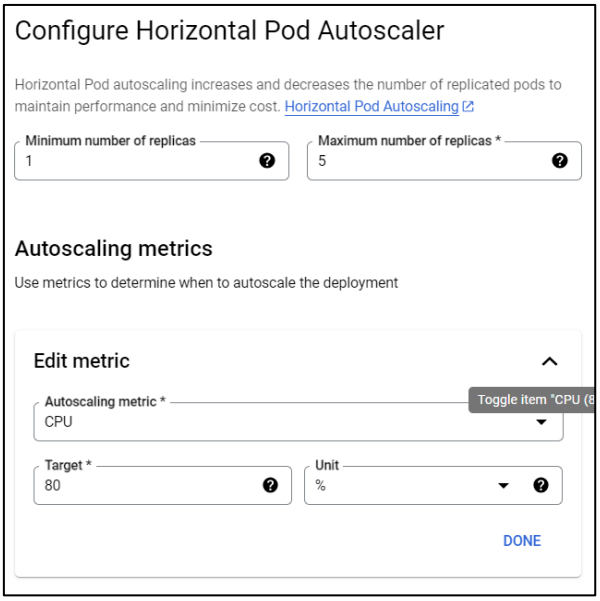


ภาพที่ 3.6 แสดงหน้าเว็บของเอนจิ้นเอ็กบนคูเบอร์เนทิส



ภาพที่ 3.7 แสดงหน้าเว็บเขียนขึ้นเพื่อทดสอบบนคูเบอร์เนทิส

จากภาพที่ 3.6 และ 3.7 เป็นหน้า Web Service ทั้ง 2 ที่อยู่บน Kubernetes เพื่อใช้ในการทดสอบการขยายตัว

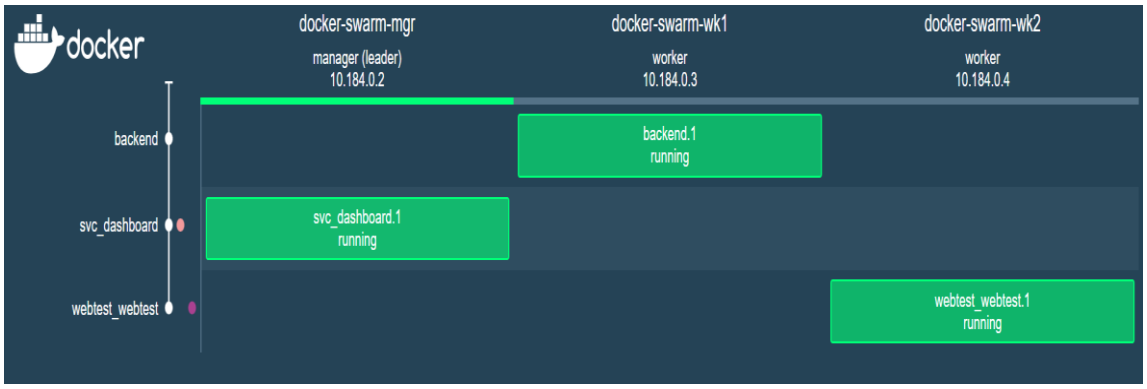


ภาพที่ 3.8 แสดงการตั้งค่าการขยายตัวของเว็บแอปพลิเคชันบนคูเบอร์เนทิส

จากภาพที่ 3.8 จะเริ่มตั้งค่าการขยายตัวของตัว Web service ด้วยการใช้ HPA (Horizontal Pod Autoscaler) สามารถใช้ค่า CPU เป็นตัวชี้วัดการเพิ่มลดจำนวน Pod ได้ตามจำนวน request ได้ให้กับ deployment โดยกำหนดเงื่อนไขในการขยายตัวของ Service โดยที่ตั้งค่าไว้ ถ้ามีการใช้งานปริมาณ CPU ถึงหรือมากกว่า 80% ให้ทำการขยายตัวเพื่อรองรับผู้ใช้งาน และขยายตัวให้มีมากถึง 5 Service และต้องใส่คำสั่งบนไฟล์ YAML ของ Service เพื่อให้ตรงกับ metrics ดังภาพที่ 3.9

```
115     spec:
116       containers:
117       - image: nginx:latest
118         imagePullPolicy: Always
119         name: nginx-1
120         resources:
121           limits:
122             cpu: 800m
123           requests:
124             cpu: 250m
125         terminationMessagePath: /dev/termination-log
126         terminationMessagePolicy: File
127       dnsPolicy: ClusterFirst
128       restartPolicy: Always
129       schedulerName: default-scheduler
130       securityContext: {}
131       terminationGracePeriodSeconds: 30
```

ภาพที่ 3.9 แสดงคำสั่งการตั้งค่าการขยายตัวของเว็บแอปพลิเคชันบนคูเบอร์เนทิส



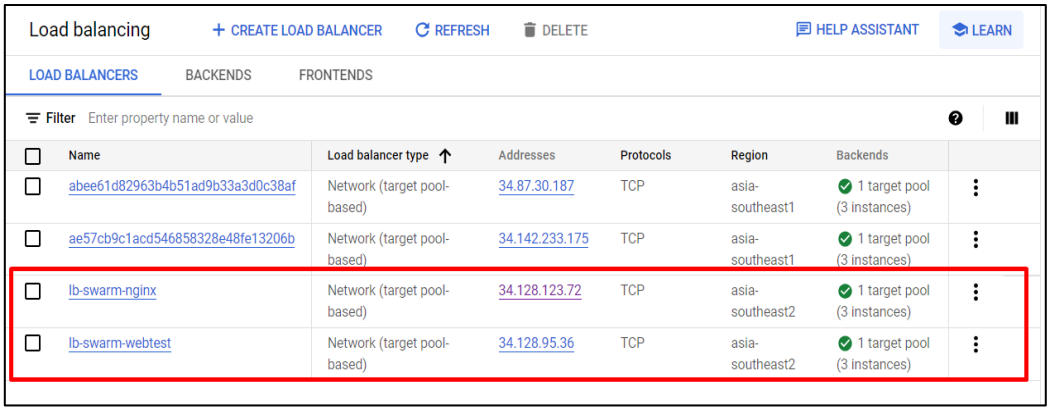
ภาพที่ 3.10 แสดงโหนดของระบบด๊อกเกอร์ สวอม

จากภาพที่ 3.10 การดำเนินการออกแบบโดยคอนเทนเนอร์ที่ 2 จะสร้างระบบคอนเทนเนอร์ Docker Swarm ขึ้นโดยใช้อยู่บน Google cloud platform ประกอบไปด้วยคลัสเตอร์ที่มี Master Node และ Worker Node

```
g625162010009@docker-swarm-mgr:~$ sudo docker service ls
ID             NAME             MODE         REPLICAS   IMAGE                              PORTS
bdsk0iilw8zm   backend          replicated   1/1        nginx:latest                       *:80->80/tcp
w9leap3ssdir   svc_dashboard    replicated   1/1        charypar/swarm-dashboard:latest    *:8081->8081/tcp
8ony8o9calne   webtest_webtest  replicated   1/1        dockerhey123/web:latest            *:8080->8080/tcp
g625162010009@docker-swarm-mgr:~$
```

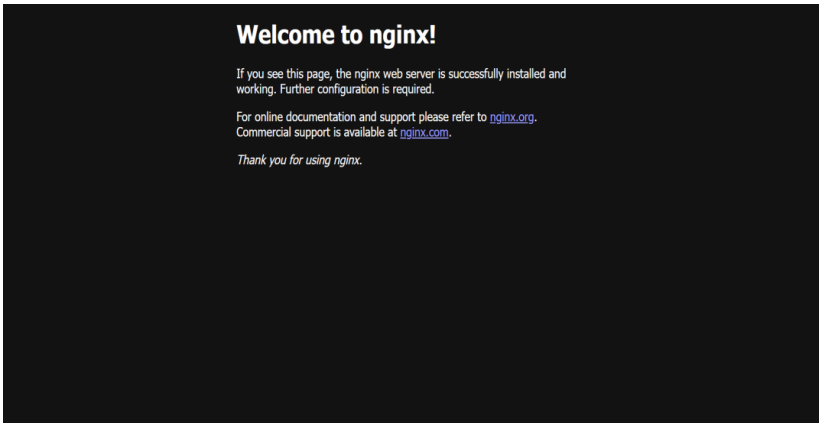
ภาพที่ 3.11 แสดงเว็บแอปพลิเคชันที่ติดตั้งบนด๊อกเกอร์ สวอม

จากภาพที่ 3.11 การติดตั้งเว็บแอปพลิเคชันเพื่อใช้ทดสอบทั้ง 2 ตัวบน Docker Swarm คือ เว็บแอปพลิเคชันโดยใช้ NGINX และเว็บแอปพลิเคชันโดยใช้ React VITE

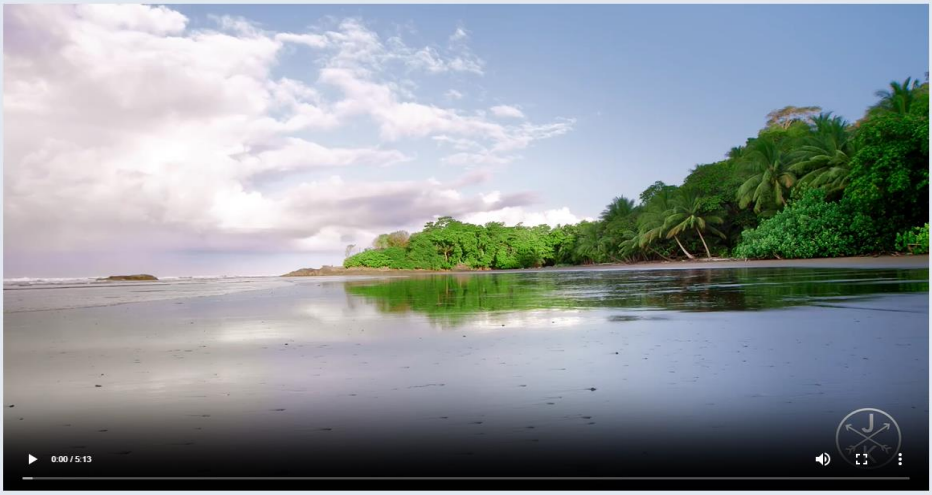


ภาพที่ 3.12 แสดงโหลดบาลานซ์ของเว็บแอปพลิเคชันบนด๊อกเกอร์ สวอม

จากภาพที่ 3.12 จะเป็นการตั้งค่าโหลดบาลานซ์บน Google Cloud เพื่อที่จะให้เข้าถึงจากภายนอกมาถึงเว็บแอปพลิเคชันได้



ภาพที่ 3.13 แสดงหน้าเว็บของเอนจิ้นเอ็กบนด๊อกเกอร์ สวอม



ภาพที่ 3.14 แสดงหน้าเว็บเขียนขึ้นเพื่อทดสอบบนด๊อกเกอร์ สวอม

จากภาพที่ 3.13 และ 3.4 เป็นหน้า Web Service ทั้ง 2 ที่อยู่บน Docker Swarm เพื่อใช้ในการทอสอบการขยายตัว

3.2.1 ขั้นตอนการทดสอบ

ในการดำเนินการทดสอบเพื่อประเมินสมรรถนะและประสิทธิภาพของ Web Service บนระบบคอนเทนเนอร์คลัสเตอร์ Kubernetes และ Docker Swarm มีขั้นตอนในการทดสอบดังนี้

(1) สร้างการทดสอบจะทำการสร้าง Generate load ด้วยโปรแกรม wrk ส่งไปยัง Web Service

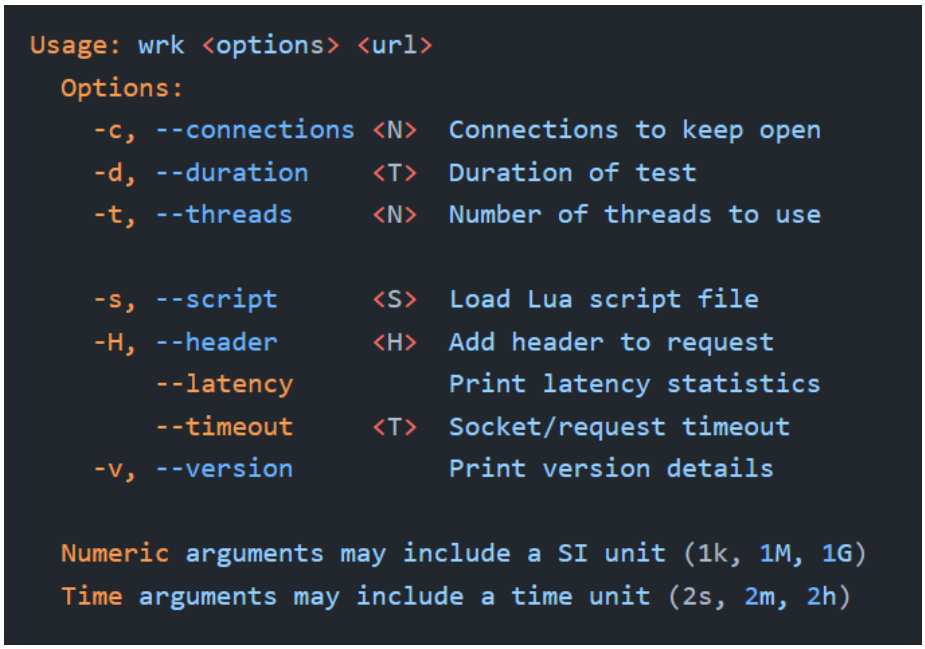
(2) เฝ้าดูพฤติกรรมของ CPU โดยที่มีการตั้งค่าของแต่ละ Service ให้มีการใช้ปกติอยู่ที่ 1 pod และขยายได้สูงสุด 5 pod มีการทำงานของ CPU เกิน 80% จะมีการขยายตัวออก เพื่อรองรับการการบริการของ Web service และถ้า CPU ต่ำกว่า 25% จะขยายตัวเข้าเพื่อลดการใช้ทรัพยากรของระบบ

(3) บันทึกผลการทดลอง เพื่อนำไปวิเคราะห์ผลการทดลองประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพ

3.3 รูปแบบและวิธีการทดสอบประสิทธิภาพ

การทดสอบจะดำเนินการทดสอบเพื่อพิจารณาประสิทธิภาพการทดสอบความสามารถในการให้บริการ (Throughput) Throughput คือความสามารถในการให้บริการของระบบต่อหน่วยเวลาในการวัดประสิทธิภาพของ Web service จะนิยมวัดเป็นจำนวนที่สามารถให้บริการได้ต่อหน่วยเวลาเป็นวินาที (request connection per second) ซึ่งการทดสอบในงานวิจัยนี้จะวัดความสามารถในการให้บริการในหน่วย requests connection per 30 second โดยทำการทดสอบการเข้าใช้บริการตั้งแต่ 100 500 1000 5000 10000 connection per 30 second ด้วยโปรแกรม wrk การทดสอบด้วยโปรแกรม wrk จะดำเนินการทดสอบบน Linux Server ที่ทำงานบน VM work Station เพื่อให้การทดสอบความใกล้เคียงกับการใช้งาน

จริงมากที่สุด การใช้งานโปรแกรม wrk จะใช้งานผ่าน Command Line Interface (CLI) โดยมีรูปแบบคำสั่งดังภาพที่ 3.15



ภาพที่ 3.15 แสดงรายละเอียดของคำสั่ง

```
$ wrk -t2 -c100 -d30s http://XXX.XXX.XXX.XXX:XX
```

โดยที่ wrk คือการเรียกใช้โปรแกรมที่ใช้ทำการทดสอบ

-t คือ จำนวน Thread ที่ใช้ในการทดสอบ

-c คือ จำนวน Connections ที่ใช้เชื่อมต่อในการทดสอบ

-d คือ ระยะเวลาที่ใช้ในการทดสอบหน่วยเป็น วินาที (s) หรือ นาที (m)

ในงานวิจัยนี้ กำหนดค่าพารามิเตอร์ที่ใช้ในการทดสอบของโปรแกรม wrk ดังนี้

-t จำนวน Thread = 2 threads

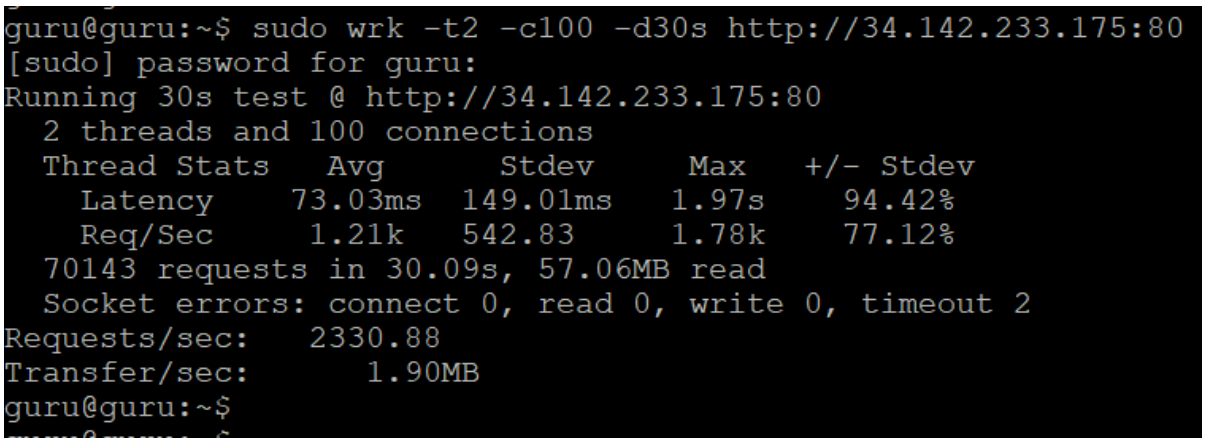
-c จำนวน Connections = 100 500 1000 5000 10000 connections

-d ระยะเวลาที่ใช้ในการทดสอบ = 30 วินาที (s)

โดยมีโฮสต์ที่ต้องการทดสอบจำนวน 4 โฮสต์ตามจำนวนของ Web Service

1. http://34.87.30.187:8080 (React VITE web service on Kubernetes)
2. http://34.142.233.175:80 (Nginx web service on Kubernetes)
3. http://34.128.95.36:8080 (React VITE web service on Docker Swarm)
4. http://34.128.123.72:80 (Nginx web service on Docker Swarm)

เมื่อทำการทดสอบด้วยโปรแกรม wrk เสร็จเรียบร้อยแล้วจะได้ผลลัพธ์ ดังแสดงตามตัวอย่าง



ภาพที่ 3.16 แสดงรายละเอียดผลลัพธ์ของการใช้โปรแกรม wrk

จากภาพที่ 3.16 จากรูปหมายความว่าใน 30s เรายิงทดสอบไปทั้งหมด 70,143 request มี timeout 2 request

ตารางที่ 3.1 ตัวอย่างตารางบันทึกผลการทดสอบ CPU Throughput

Throughput (Connection / CPU %)						
Containers / Services	**100**	**500**	**1000**	**5000**	**10000**	**CPU %**
Docker Swarm-NGINX						**%**
Docker Swarm-WEB-LOADTEST						**%**
Kubernetes-NGINX						**%**
Kubernetes-WEB-LOADTEST						**%**

ตารางที่ 3.2 ตัวอย่างตารางบันทึกผลการทดสอบการขยายตัวของ 2 คอนเทนเนอร์

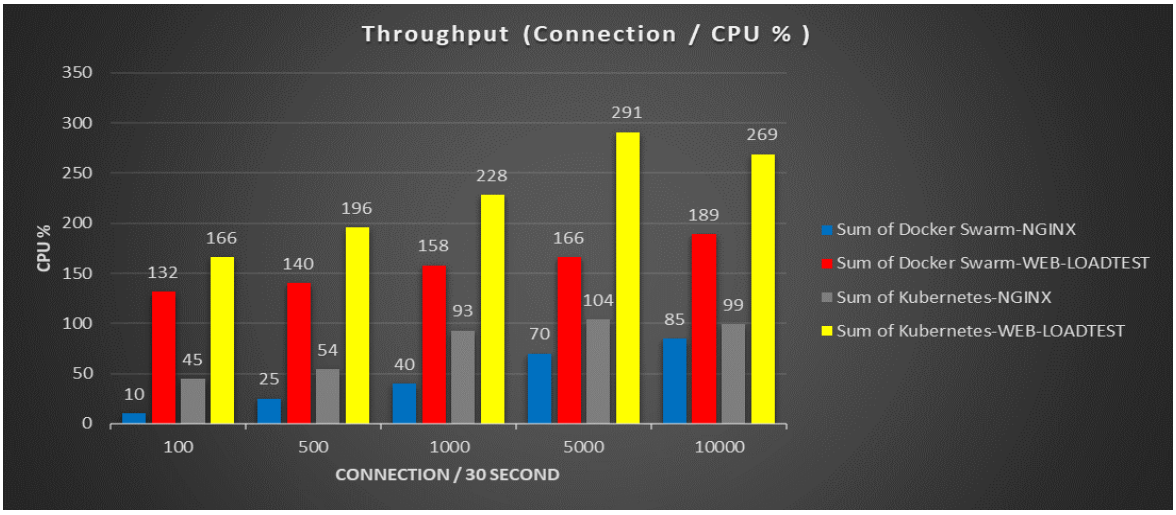
Kube vs Docker Swarm Scale up/second					
Containers	**Services**	**Pod 2**	**Pod 3**	**Pod 4**	**Pod 5**
Docker Swarm	NGINX				
	WEB-LOADTEST				
Kubernetes	NGINX				
	WEB-LOADTEST				

บทที่ 4
ผลการวิจัย

เพื่อให้ผลการทดลองบรรลุวัตถุประสงค์ของการทำวิจัยจึงได้ทดสอบ CPU โดยแบ่งเป็น HTTP requests 100 500 1000 5000 10000 ต่อ 30 วินาที

ตารางที่ 4.1 แสดงผลการทดสอบการทำงานของ CPU

Throughput (Connection / CPU %)						
Containers / Services	**100**	**500**	**1000**	**5000**	**10000**	**CPU %**
Docker Swarm-NGINX	10	25	40	70	85	**%**
Docker Swarm-WEB-LOADTEST	132	140	158	166	189	**%**
Kubernetes-NGINX	45	54	93	104	99	**%**
Kubernetes-WEB-LOADTEST	166	196	228	291	269	**%**



ภาพที่ 4.1 แสดงผลการทดสอบการทำใช้งานของซีพียู

จากภาพที่ 4.1 เป็นการเปรียบเทียบทำงานของ CPU เมื่อมีการเรียกใช้งาน Web Service จะเห็นได้ว่า CPU ของแต่ละคอนเทนเนอร์และของแต่ Web Service ที่มีการเรียกใช้ Connection ที่มีปริมาณงานที่เท่ากัน แต่การใช้งานงานปริมาณ CPU แตกต่างกัน ผลที่ออกมาจะแสดงให้เห็นว่า Service ของ NGINX ที่ไม่มีเนื้อหา จึงมีการใช้งานของ CPU ที่น้อย เมื่อเทียบกับ Web test เพราะ Web test มีวีดีโอ เป็น Full HD:

ความละเอียดของภาพ 1080p จึงทำให้ CPU มีการทำงานมากขึ้นอย่างเห็นได้ชัดทั้งที่อยู่ในคอนเทนเนอร์เดียวกัน

4.1 การทดสอบการทำงานเบื้องต้น

4.1.1 การทดสอบการขยายตัวของคอนเทนเนอร์ Kubernetes

ตารางที่ 4.2 แสดงผลการทดสอบการขยายตัวบน Kubernetes

Kubernetes Auto-Scale up/second					
Containers	Services	Pod 2	Pod 3	Pod 4	Pod 5
Kubernetes	NGINX	1	1	12	12
	WEB-LOADTEST	0.1	0.1	0.1	15

Nginx:
**Start load test : 20 June 23 / 11:42:40 PM
- Cpu resource utilization (percentage of request) above target : 11:43:18 PM
- Sclaed up Pod 2 : 11:43:19 PM
- Sclaed up Pod 3 : 11:43:19 PM
- Sclaed up Pod 4 : 11:43:30 PM
- Sclaed up Pod 5 : 11:43:30 PM

#sudo wrk -t2 -c10000 -d2m http://34.142.233.175:80

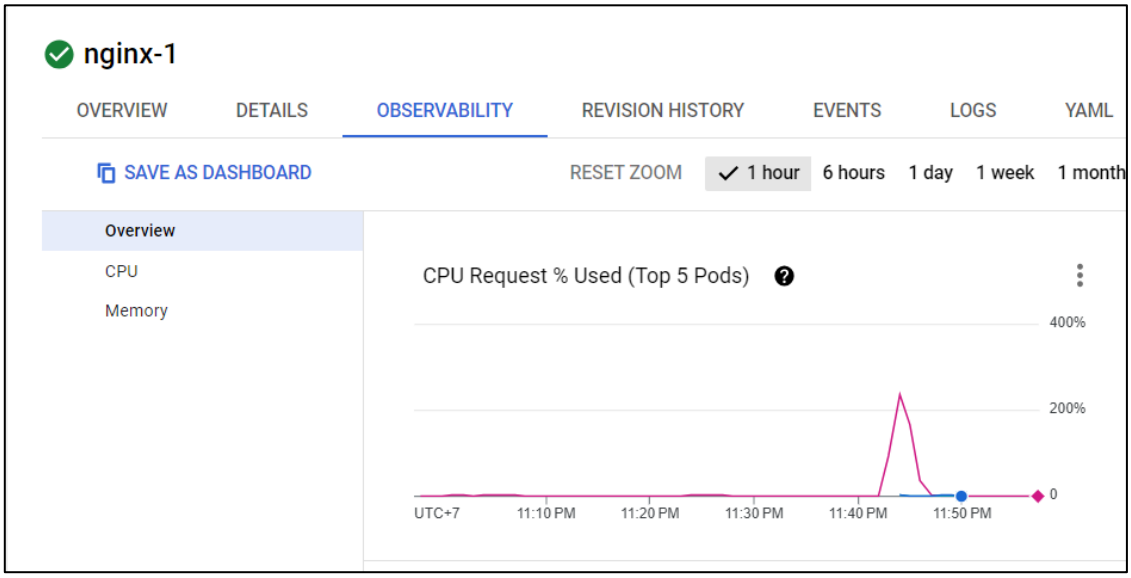
ภาพที่ 4.2 แสดงผลช่วงเวลาในการทดของเอนจิ้นเอ็กบนคูเบอร์เนทิส

Web-loadtest:
**Start load test : 20 June 23 / 11:12:40 PM
- Cpu resource utilization (percentage of request) above target : 11:12:42 PM
- Sclaed up Pod 2 : 11:12:42 PM
- Sclaed up Pod 3 : 11:12:42 PM
- Sclaed up Pod 4 : 11:12:42 PM
- Sclaed up Pod 5 : 11:12:57 PM

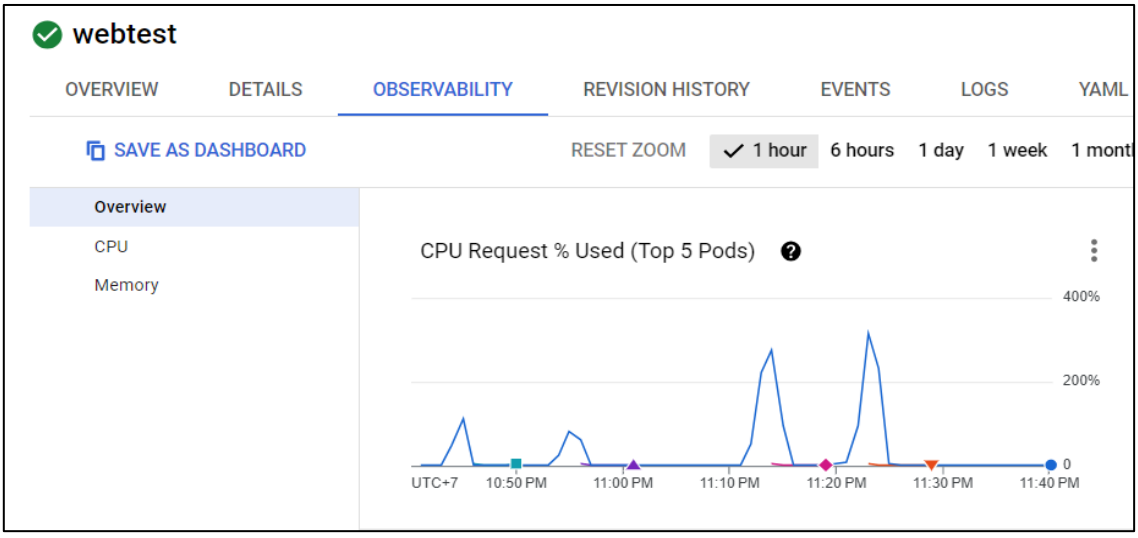
#sudo wrk -t2 -c10000 -d2m http://34.87.30.187:8080

ภาพที่ 4.3 แสดงผลช่วงเวลาในการทดของเว็บเทสบนคูเบอร์เนทิส

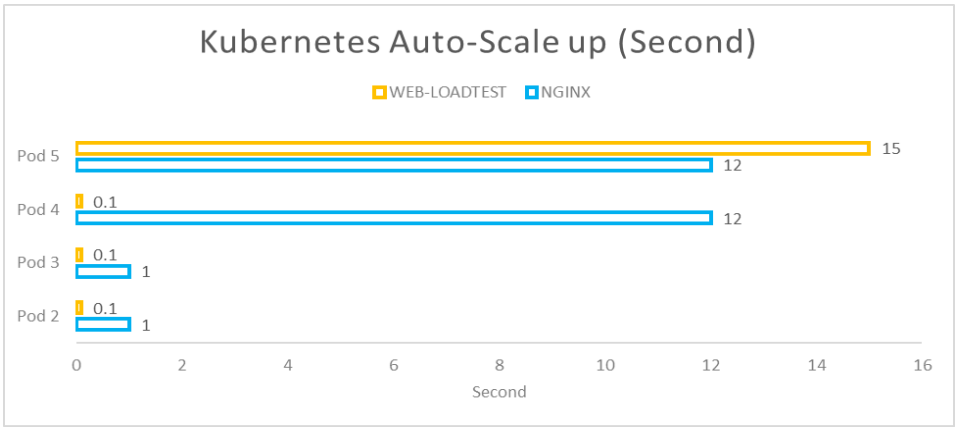
จากภาพที่ 4.2 และ 4.3 จะบอกช่วงเวลาตั้งแต่เริ่มทำการสร้าง Generate load จนเริ่มการทำงานของ CPU เมื่อมีการใช้ CPU ถึงเกณฑ์ที่ตั้งไว้ การขยายตัวของ Survive จะเริ่มทำงานจะดูได้จากกราฟของ CPU ดังภาพของ 4.4 และ 4.5



ภาพที่ 4.4 แสดงผลการทำงานของ ซีพียูเอนจิ้นเอ็กบนคูเบอร์เนเตส



ภาพที่ 4.5 แสดงผลการทำงานของซีพียูเว็บเทสบนคูเบอร์เนเตส



ภาพที่ 4.6 กราฟแสดงผลช่วงเวลาการขยายตัวในการทดสอบ

จากภาพที่ 4.6 จะแสดงการเปรียบเทียบการขยายตังของ Kubernetes ทั้ง 2 service จะเห็นได้ว่า service web test จะใช้เวลาในการขยายตัวเพื่อรองรับได้เร็วกว่า แต่จะเห็นว่า Pod ที่ 5 ช้ากว่าเนื่องจาก pod 1-4 ของ Web-load test มีเพียงพอต่อการใช้งานเบื้องต้น เลยทำให้ CPU ของ Pod ที่ 5 เริ่มทำงานช้ากว่าของ NGINX

4.1.2 การทดสอบการขยายตัวของคอนเทนเนอร์ของ Docker Swarm

ตารางที่ 4.3 แสดงผลการทดสอบการทำใช้งานของ CPU บน Docker Swarm

Docker Swarm Scale up/second					
Containers	**Services**	**Pod 2**	**Pod 3**	**Pod 4**	**Pod 5**
Docker Swarm	NGINX	2	3	5	6
	WEB-LOADTEST	3	69	72	74

Nginx:
**Start load test : 21 June 23 / 12:49:00 PM
- Cpu resource utilization (percentage of request) above target : 12:49:40 PM
- Sclaed up Pod 2 : 12:49:42 PM
- Sclaed up Pod 3 : 12:49:43 PM
- Sclaed up Pod 4 : 12:49:45 PM
- Sclaed up Pod 5 : 12:49:46 PM

#sudo wrk -t2 -c10000 -d2m http://34.128.123.72:80

ภาพที่ 4.7 แสดงผลช่วงเวลาในการทดสอบของเอนจิ้นเอ็กบนด๊อกเกอร์ สวอม

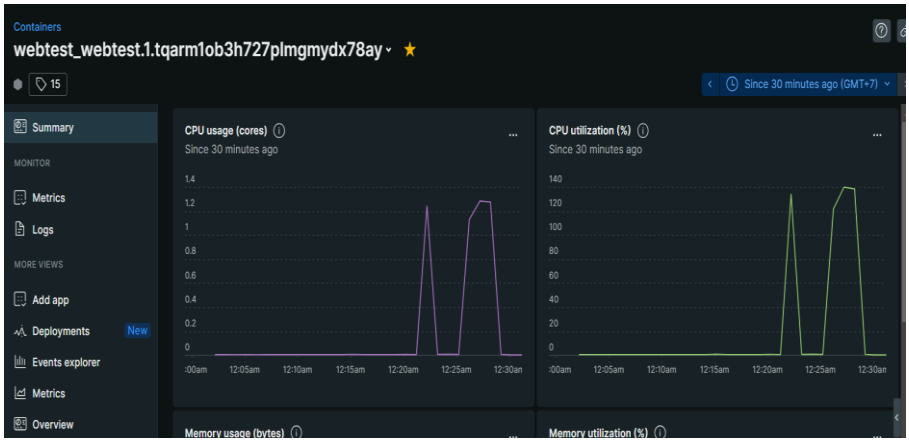
```
Web-loadtest:
**Start load test : 21 June 23 / 12:27:00 AM
- Cpu resource utilization (percentage of request) above target : 12:27:27 PM
- Sclaed up Pod 2 : 12:27:30 PM
- Sclaed up Pod 3 : 12:28:36 PM
- Sclaed up Pod 4 : 12:28:39 PM
- Sclaed up Pod 5 : 12:28:41 PM

#sudo wrk -t2 -c10000 -d2m http://34.128.95.36:8080
```

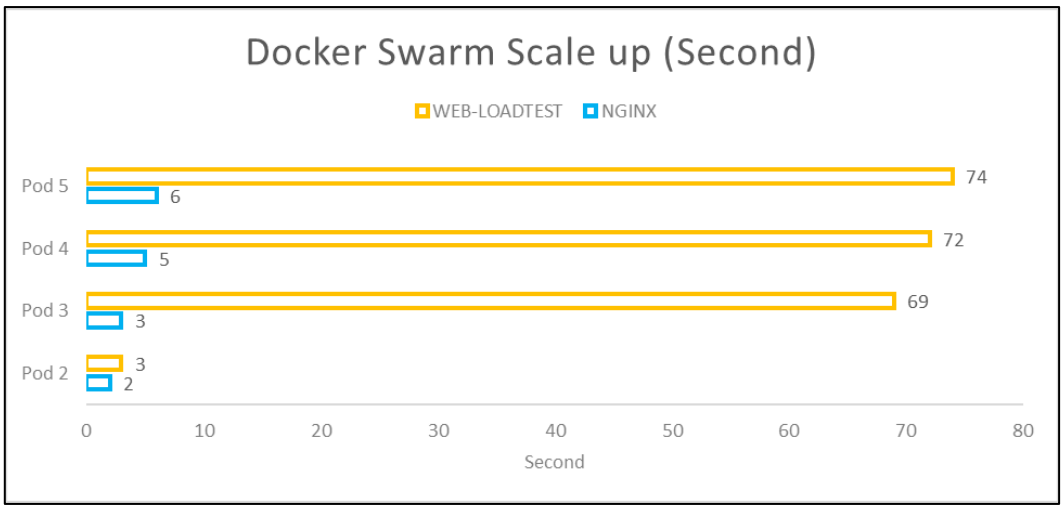
ภาพที่ 4.8 แสดงผลช่วงเวลาในการทดสอบของเว็บเทสบนด๊อกเกอร์ สวอม

จากภาพที่ 4.7 และ 4.8 จะบอกช่วงเวลาตั้งแต่เริ่มทำการสร้าง Generate load จนเริ่มการทำงานของ CPU เมื่อมีการใช้ CPU ถึงเกณฑ์ที่ตั้งไว้ การขยายตัวของ Service จะเริ่มทำงานจะดูได้จากกราฟของ CPU ดังภาพของ 4.9 และ 4.10

ภาพที่ 4.9 แสดงผลการทำงานของซีพียูเอนจิ้นเอ็กบนด๊อกเกอร์ สวอม



ภาพที่ 4.10 แสดงผลการทำงานของซีพียูเว็บเทสบนด๊อกเกอร์ สวอม



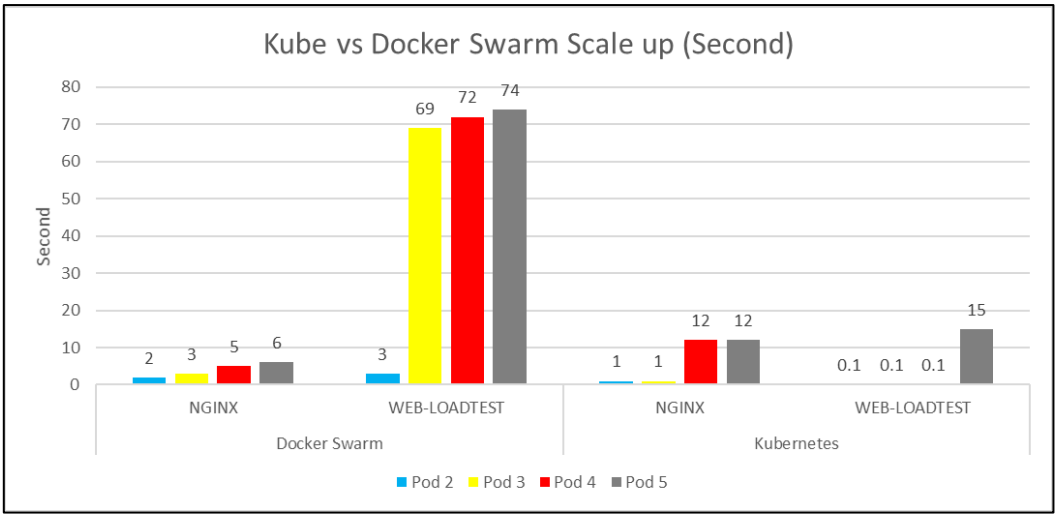
ภาพที่ 4.11 กราฟแสดงผลช่วงเวลาการขยายตัวในการทดสอบ

จากภาพที่ 4.11 จะแสดงการเปรียบเทียบการขยายตังของ Docker Swarm ทั้ง 2 service จะเห็นได้ว่า service web test จะใช้เวลาในการขยายตัวเพื่อรองรับได้ช้ากว่า เนื่องจากขนาดของตัวเว็บที่ไฟล์ขนาดใหญ่กว่าของ NGINX จึงทำให้เวลาในการสร้างแต่ละ Pod มีเวลานาน

4.1.3 การเปรียบเทียบการขยายตัวของคอนเทนเนอร์ของ Kubernetes และ Docker Swarm

ตารางที่ 4.4 แสดงผลการทดสอบเปรียบเทียบของด็อกเกอร์ สวอม และคูเบอร์เนทิส

Kube vs Docker Swarm Scale up/second					
Containers	**Services**	**Pod 2**	**Pod 3**	**Pod 4**	**Pod 5**
Docker Swarm	NGINX	2	3	5	6
	WEB-LOADTEST	3	69	72	74
Kubernetes	NGINX	1	1	12	12
	WEB-LOADTEST	0.1	0.1	0.1	15



ภาพที่ 4.12 กราฟแสดงผลการทดสอบเปรียบเทียบของด็อกเกอร์ สวอม และคูเบอร์เนทิส

จากภาพที่ 4.12 จะได้แสดงให้เห็นอย่างชัดเจนว่าคอนเทนเนอร์ของ Kubernetes การขยายตัวที่เร็วกว่ามากถึง 93% ถ้าดูจากกราฟแล้ว Docker swarm จะใช้เวลากว่ามาก เนื่องจาก Docker Swarm ไม่มี Auto scaling จำเป็นต้องให้ผู้ดูแลระบบทำ manual scaling จึงทำให้เกิดความล่าช้าและมีการขยายตัวของ Service ที่มี content ที่มีเนื้อหาเยอะได้ช้า จึงสรุปได้ว่าการที่จะเลือกใช้คอนเทนเนอร์เพื่อมาใช้ในการให้บริการทั้งภายในหรือบริการให้ลูกค้าควรเลือกคอนเทนเนอร์ที่มีประสิทธิภาพการรองรับการให้บริการที่ดีกว่าอย่าง Kubernetes

บทที่ 5
สรุปผลการวิจัย อภิปรายผล และข้อเสนอแนะ

ในบทนี้จะเป็นการอภิปรายเพื่อสรุปผลที่ได้จากการทดสอบงานวิจัย และข้อเสนอแนะสำหรับแนวทางในการพัฒนางานวิจัยนี้ต่อไปเพื่อแก้ข้อบกพร่องของงานวิจัยให้มีประสิทธิภาพ

5.1 สรุปผลตามวัตถุประสงค์ของงานวิจัย

5.1.1 สามารถวิเคราะห์และออกแบบคอนเทนเนอร์ทั้ง 2 แพลตฟอร์ม โดยมี 1 Kubernetes 2 Docker Swarm ได้

5.1.2 สามารถจัดการโครงสร้างของ Kubernetes และ Docker Swarm ด้วย Google Cloud Platform ได้

5.1.3 สามารถอำนวยความสะดวกให้กับผู้ดูแลระบบภายในองค์กรในการเลือกใช้คอนเทนเนอร์

5.1.4 สามารถลดความเสี่ยงในการให้บริการด้าน Web Service

5.2 ปัญหาอุปสรรคและข้อจำกัดของงานวิจัย

5.2.1 ปัญหาของ Docker Swarm ยังไม่สามารถขยายตัวอัตโนมัติได้

5.2.2 มีข้อจำกัดในเรื่องของการ Monitor CPU และการ Deploy ที่มีความยุ่งยากบน Docker Swarm

5.2.3 ระบบ Google Kubernetes Engine สามารถกำหนดทรัพยากรของเครื่องได้เฉพาะ Worker Node เท่านั้น เครื่อง Master Node ระบบจะจัดเตรียมไว้ให้พร้อมกับ Cluster ที่สามารถเข้าใช้งานได้เพียงอย่างเดียวเท่านั้น

5.2.4 wrk ไม่สามารถนำมาวิเคราะห์เปรียบเทียบได้ในทันที ต้องนำเข้าผลการทดลองเพื่อคำนวณและวิเคราะห์ผลและนำมาจัดทำแผนภูมิเปรียบเทียบด้วยโปรแกรม Microsoft Excel

5.3 ข้อเสนอแนะสำหรับงานวิจัยในอนาคต

5.3.1 ต้องศึกษาในการขยายตัวอัตโนมัติของ Docker Swarm

5.3.2 ศึกษาในการ Monitor CPU และการ Deploy เพื่อลดความยุ่งยากบน Docker Swarm

5.3.3 การทดสอบบน Google Cloud Platform ซึ่งอาจจะมีผลการทดสอบที่คลาดเคลื่อน เมื่อนำไปทดสอบหรือใช้งานบนระบบ Public Cloud อื่น ๆ เนื่องจากประสิทธิภาพของระบบ Cloud load balancer และประสิทธิภาพของระบบเครือข่ายบนระบบ Public Cloud นั้นๆ มีความแต่งต่างกัน โดยจะต้องดำเนินงานวิจัยต่อไปจะดำเนินการทดลองบน Public Cloud ระบบอื่น ๆ เช่น Amazon web services, Microsoft Azure เป็นต้น

บรรณานุกรม

บรรณานุกรม

[1] [Online]. https://www.aosoft.co.th/article/342/Cloud-computing.html

[2] [Online]. https://www.mindphp.com/%E0%B8%84%E0%B8%B9%E0%B9%88%E0%B8%A1-%E0%B8%B7%E0%B8%AD/73%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3/9187-containers.html

[3] [Online]. https://thiti.dev/blog/6884/

[4] [Online]. https://blog.cloudnatician.com/%E0%B8%AD%E0%B8%98%E0%B8%B4%E0%B8-%9A%E0%B8%B2%E0%B8%A2kubernetes%E0%B8%9E%E0%B8%B7%E0%B9%89%E0%B8%99%E0%B8%90%E0%B8%B2%E0%B8%99%E0%B9%83%E0%B8%995%E0%B8%99%E0%B8%B2%E0%B8%97%E0%B8%B5-823cb6190c65

[5] [Online]. https://www.bestinternet.co.th/single_blog.php?id=107&nginx%20(NGINX)%20%-E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%3

[6] [Online]. https://mosprogramer.medium.com/%E0%B8%A1%E0%B8%B2%E0%B8%-A3%E0%B8%B9%E0%B9%89%E0%B8%88%E0%B8%B1%E0%B8%81react%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B9%80%E0%B8%96%E0%B8%AD%E0%B8%B0-eb04ab8b8ec1

[7] สิทธินนท์ศรีสวัสดิ์และทรงฤทธิ์กิติศรีวรพันธุ์ The HorizontalPodAutoscalerwithan Online Calendar for Kubernetes 2021 [Online]. https://ph02.tcithaijo.org/index.php/ectiard/-article/view/243951/165521

[8] Teerapat Khunpech An Auto-scaling Mechanism in Docker System [Online]. https://nccit.net/wp-content/uploads/2016/05/Example_Paper.pdf

[9] M. Song, C. Zhang and E. Haihong, “An autoscaling system for api gateway based on Kubernetes,” In 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), pp. 109-112, Nov 2018.

ประวัติผู้เขียน

ชื่อ – นามสกุล วรเทพ อหันทริก

ประวัติการศึกษา

พ.ศ. 2558 - ระดับปริญญาตรี วิทยาศาสตรบัณฑิต สาขาเทคโนโลยีสารสนเทศ (ซอฟต์แวร์) มหาวิทยาลัยราชภัฏบ้านสมเด็จเจ้าพระยา