# การประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพอินเกรสคอนโทรลเลอร์บนระบบคลัสเตอร์คูเบอร์เนทิส

อาธิป พวงลำใย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์และโทรคมนาคม วิทยาลัยนวัตกรรมด้านเทคโนโลยี
และวิศวกรรมศาสตร์
มหาวิทยาลัยธุรกิจบัณฑิตย์
พ.ศ.2561

# Performance Evaluation and Comparison of Ingress Controllers on Kubernetes Cluster

**Arthip Phuanglumyai**

**A Thesis Submitted in Partial Fulfillment of the Requirements**

**for the Degree of Master of Engineering**

**Department of Computer and Telecommunication**

**College of Innovative Technology And Engineering,**

**Dhurakij Pundit University**

**2018**

# ใบรับรองวิทยานิพนธ์

วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์

มหาวิทยาลัยธุรกิจบัณฑิตย์

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

หัวข้อวิทยานิพนธ์ การประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพอินเกรสคอนโทรลเลอร์บนระบบคลัสเตอร์คูเบอร์เนทิส

เสนอโดย นายอาธิป พวงลำใย

สาขาวิชา วิศวกรรมคอมพิวเตอร์และโทรคมนาคม

อาจารย์ที่ปรึกษาวิทยานิพนธ์ อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์

ได้พิจารณาเห็นชอบโดยคณะกรรมการสอบวิทยานิพนธ์แล้ว

.....ประธานกรรมการ

(อาจารย์ ดร.ประศาสน์ จันทราทิพย์)

.....กรรมการและอาจารย์ที่ปรึกษาวิทยานิพนธ์

(อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์)

..... กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณรงค์เดช กีรติพรานนท์)

..... กรรมการ

(อาจารย์ ดร.เจนจบ วีระพานิชเจริญ)

**วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์รับรองแล้ว**

..... คณบดีวิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์

(ผู้ช่วยศาสตราจารย์ ดร.ณรงค์เดช กีรติพรานนท์)

วันที่ ...19..... เดือน .....กรกฎาคม..... พ.ศ. ...2561.....

| | |
|---|---|
| หัวข้อวิทยานิพนธ์ | การประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพอินเกรสคอนโทรลเลอร์บนระบบคลัสเตอร์คูเบอร์เนทิส |
| ชื่อผู้เขียน | อาธิป พวงลำใย |
| อาจารย์ที่ปรึกษา | อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์ |
| สาขาวิชา | วิศวกรรมคอมพิวเตอร์และโทรคมนาคม |
| ปีการศึกษา | 2560 |

## บทคัดย่อ

บทความนี้ได้นำเสนอการติดตั้งและทดสอบ เลเยอร์-7 โหลดบาลานซ์ คอนโทรลเลอร์ที่ทำหน้าที่เพื่อกำหนดเส้นทางในการเข้าใช้บริการของระบบที่ให้บริการอยู่บนการประมวลผลแบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทีส จำนวน 4 ชนิด คือ 1. Nginx Ingress controller 2. Traefik Ingress controller 3. Voyager Ingress controller และ 4. GCE L7 load balancer controller (GLBC) เพื่อประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพ ทางด้านความสามารถในการให้บริการ ความเร็วในการตอบสนองและนำข้อมูลที่ได้จากการทดลองมาพิจารณา วิเคราะห์และสรุปผล

ผลการวิจัยในด้านความสามารถในการให้บริการ (Throughput) พบว่า GCE L7 load balancer controller สามารถให้บริการได้สูงกว่าอินเกรสคอนโทรลเลอร์อื่น ๆ ในขณะที่ Nginx, Traefik และ Voyager Ingress controller มีความสามารถให้บริการใกล้เคียงกัน ในด้านความเร็วในการตอบสนองเฉลี่ย (Average response time) พบว่า GCE L7 load balancer controller ใช้เวลาในการตอบสนองเฉลี่ยน้อยที่สุด น้อยกว่าอินเกรสคอนโทรลเลอร์อื่น ๆ ในขณะที่ Nginx, Traefik และ Voyager Ingress controller ใช้เวลาในการตอบสนองเฉลี่ยใกล้เคียงกันและความเร็วในการตอบสนองเฉลี่ยมีแนวโน้มสูงขึ้นเมื่อจำนวนครั้งในการร้องขอข้อมูลต่อหน่วยเวลาเพิ่มมากขึ้น ถ้าพิจารณาถึงความสะดวกต่อการใช้งานในกรณีที่เลือกใช้ Public Cloud ของ Google Cloud Platform แล้ว GCE L7 load balancer controller จะเป็นตัวเลือกที่สามารถเลือกใช้งานได้สะดวกที่สุด เนื่องจากมีการติดตั้งไว้บน Google Kubernetes Engines อยู่แล้ว โดยที่ไม่ต้องติดตั้งอินเกรสคอนโทรลเลอร์อื่นเพิ่มเติม อย่างไรก็ตามพบว่าอินเกรสคอนโทรลเลอร์อื่นมีความสามารถบางอย่างที่ไม่พบใน GCE L7 เช่นการทำโหลดบาลานซ์แบบที่มีการจัดระดับความสำคัญได้

| | |
|---|---|
| Thesis Title | Performance Evaluation and Comparison of Ingress Controllers on Kubernetes Cluster |
| Author | Arthip Phuanglumyai |
| Thesis Advisor | Chaiyaporn Khemapatapan, Ph.D |
| Department | Computer and Telecommunication Engineering |
| Academic Year | 2017 |

## ABSTRACT

This article presents an installation and test of layer-7 load balance controllers which is functioned as a load balancer for accessing a service on Kubernetes containers. There are 4 types as follows: 1. Nginx Ingress Controller 2. Traefik Ingress Controller 3. Voyager Ingress Controller and 4. GCE L7 load balancer controller (GLBC). The performance will be evaluate and compare throughput, response time and resources consumption.

The evaluation results from this study revealed that the most efficient throughput is GCE L7 load balancer controller while Nginx, Traefik and Voyager ingress controller have a similar throughput. Regarding average response time, it was found that GCE L7 load balancer controller take the least time to response meanwhile Nginx, Traefik and Voyager Ingress controller take a similar average response time which has a high tendency depending on the requests per time. Therefore, GCE L7 load balancer controller is the best option for public cloud of Google Cloud Platform in terms of its convenience since it was already installed on Google Kubernetes Engines without additional ingress controller installation. However, GCE L7 load balancer has no advanced features whereas others ingress controller have advanced features such as priority load balancing.

# กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จลุล่วงได้ด้วยความอนุเคราะห์ช่วยเหลือ ของท่านอาจารย์ที่ปรึกษา อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์ ซึ่งได้ให้คำปรึกษา เสนอแนะ แนะนำแนวทาง และความช่วยเหลือหลายต่าง ๆ สำหรับการทำวิจัย ช่วยตรวจสอบ แก้ไขข้อบกพร่อง จนทำให้วิทยานิพนธ์เล่มนี้สำเร็จได้โดยสมบูรณ์ ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงสำหรับทุกสิ่งมา ณ ที่นี้

ขอกราบขอบพระคุณท่านอาจารย์ ดร.ประศาสน์ จันทราทิพย์ ประธานกรรมการสอบวิทยานิพนธ์ ท่านอาจารย์ ผศ.ดร.ณรงค์เดช กีรติพรานนท์ และ ท่านอาจารย์ ดร.เจนจบ วีระพานิชเจริญ กรรมการสอบวิทยานิพนธ์ ที่สละเวลามาเป็นกรรมการสอบวิทยานิพนธ์ และให้คำแนะนำ ข้อเสนอแนะ แนวทางการปรับปรุงแก้ไข รวมทั้งให้ความรู้เพิ่มเติมที่เป็นประโยชน์ เพื่อให้งานวิจัยนี้มีความสมบูรณ์มากยิ่งขึ้น

ขอขอบคุณเพื่อน ๆ CT56 สำหรับความรู้ ความช่วยเหลือ และมิตรภาพที่มีให้ต่อกัน

ขอขอบคุณเจ้าหน้าที่มหาวิทยาลัยธุรกิจบัณฑิตย์ คุณกุลธรา อานนท์ คุณเสาวลักษณ์ ตู้ดำ และคุณกุลธิดา รอดบุญ ที่ช่วยดำเนินการประสานงานด้านเอกสารต่าง ๆ ช่วยแจ้งเตือนกำหนดการ และความช่วยเหลืออื่น ๆ ที่มอบให้ ขอขอบคุณจากใจจริง

ขอกราบขอบพระคุณบิดา มารดา น้องชาย และครอบครัวที่เป็นกำลังใจ แรงบันดาลใจ และคอยช่วยเหลือสนับสนุนในการทำวิทยานิพนธ์เล่มนี้ให้สำเร็จลุล่วงไปได้ด้วยดี

สุดท้ายนี้คุณค่าและประโยชน์ใด ๆ ที่อาจจะมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบให้แด่ผู้มีพระคุณทุกท่านที่ให้ความช่วยเหลือ จนการทำวิทยานิพนธ์ฉบับนี้และการศึกษาสำเร็จไปได้ด้วยดี

อาธิป พวงลำใย

# สารบัญ

# สารบัญ (ต่อ)

# สารบัญตาราง

**ตารางที่** **หน้า**

# สารบัญภาพ

**ภาพที่** **หน้า**

# สารบัญภาพ (ต่อ)

**ภาพที่** **หน้า**

# ประมวลศัพท์และคำย่อ

| | |
|---|---|
| AWS | Amazon Web Service |
| CLI | Command Line Interface |
| GCP | Google Cloud Platform |
| GKE | Google Kubernetes Engine |
| GUI | Graphic User Interface |
| k8s | Kubernetes |
| UI | User Interface |
| VM | Virtual Machine |

# บทที่ 1
# บทนำ

## 1.1 ที่มาและความสำคัญ

เทคโนโลยีสารสนเทศ (Information Technology) เป็นเครื่องมือสำคัญประการหนึ่งในการดำเนินธุรกิจของหน่วยงาน บริษัท หรือผู้ให้บริการ การประยุกต์ใช้เทคโนโลยีสารสนเทศได้อย่างเหมาะสม จะส่งผลให้การดำเนินการมีประสิทธิภาพเพิ่มมากขึ้น สามารถลดระยะเวลาในการทำงานได้อย่างมีนัยยะสำคัญ ลดการใช้ทรัพยากร ลดการใช้พลังงานไฟฟ้า แต่จะสามารถเพิ่มความง่าย ความสะดวกในการใช้งาน และเพิ่มผลผลิตได้มากยิ่งขึ้น

ผู้ให้บริการในปัจจุบันจึงได้ปรับเปลี่ยนรูปแบบในการให้บริการโดยประยุกต์ใช้เทคโนโลยีสารสนเทศ เพื่อเป็นการให้บริการผ่านอินเตอร์เน็ต (Online Services) ในรูปแบบเว็ปแอพพลิเคชั่น โมบายแอพพลิเคชั่น หรือการให้บริการในรูปแบบออนไลน์อื่น ๆ มากยิ่งขึ้น เพื่อให้เหมาะสมกับโลกยุคเทคโนโลยีในปัจจุบัน โดยนิยมที่จะแบ่งบริการออกเป็นส่วนย่อยหลาย ๆ ส่วน (Microservices) และนำมาให้บริการต่อผู้ใช้บริการตามแต่ละลักษณะหน้าที่ที่กำหนดไว้

เมื่อมีการแบ่งบริการออกเป็นส่วนย่อยหลาย ๆ ส่วนแล้ว นั้น การทำงานของระบบเบื้องหลัง จึงต้องมีการปรับปรุง โดยการประยุกต์ใช้ระบบประมวลผลแบบกลุ่มเมฆ (Cloud Computing) ร่วมกับระบบการประมวลผลแบบคอนเทนเนอร์ (Containers Technology) เพื่อความสะดวกในการดูแลและจัดการระบบการให้บริการ สามารถจัดการระบบให้มีความเหมาะสมกับภาระงานเพื่อประหยัดการใช้พลังงานในศูนย์ข้อมูล และสิ่งที่สำคัญที่สุดคือเพื่อให้บริการแก่ผู้เข้าใช้บริการด้วยการเข้าถึงที่รวดเร็วและสามารถเข้าถึงบริการได้ตลอดเวลา

ระบบบริหารจัดการการประมวลผลแบบคอนเทนเนอร์ที่เป็นที่นิยมใช้ในปัจจุบัน คือ คูเบอร์เนทิส (Kubernetes) ซึ่งถูกพัฒนาโดย Google Inc. โดยใช้ประสบการณ์ในการดูแลศูนย์ข้อมูลที่ได้ดำเนินการมาอย่างยาวนานเป็นแนวคิดและบทเรียนในการพัฒนาเพื่อให้ระบบมีประสิทธิภาพมากที่สุด การทำงานของคูเบอร์เนทิสนั้น จะทำงานในรูปแบบที่นำเครื่องคอมพิวเตอร์แม่ข่าย หรือเครื่องคอมพิวเตอร์เสมือนมาทำงานร่วมกันในลักษณะคลัสเตอร์ ที่ใช้คอมพิวเตอร์หลาย ๆ เครื่องช่วยกันประมวลผล เพื่อลดระยะเวลาในการทำงานและเพิ่มความมั่นคงต่อระบบการให้บริการในกรณีที่มีคอมพิวเตอร์เครื่องใดเครื่องหนึ่งขัดข้อง

การเข้าใช้งานระบบที่ให้บริการอยู่บนคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสที่ทำงานอยู่บนระบบประมวลผลแบบกลุ่มเมฆ สามารถที่จะเข้าถึงได้หลายวิธี เช่น ExternalIP, NodePort หรือ LoadBalancer ซึ่งแต่ละวิธีก็จะมีข้อดี-ข้อเสียที่แตกต่างกันไป แต่วิธีที่เป็นที่นิยมใช้งานเมื่อให้บริการบนอินเตอร์เน็ตนั้น คือการเข้าใช้งานผ่านอินเกรส (Ingress) ซึ่งเป็นโหลดบาลานซ์คอนโทรลเลอร์ ที่ทำงานในระดับแอปพลิเคชันเลเยอร์ (เลเยอร์-7)

งานวิจัยฉบับนี้ จึงนำเสนอการติดตั้ง ทดลองและทดสอบ เลเยอร์-7 โหลดบาลานซ์คอนโทรลเลอร์ ที่ทำหน้าที่เพื่อกำหนดเส้นทางในการเข้าใช้บริการของระบบที่ให้บริการอยู่บนการประมวลผลแบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทีส จำนวน 4 รูปแบบ คือ

1. Nginx Ingress controller[1] รุ่นที่พัฒนาโดยชุมชน Kubernetes
2. Traefik Ingress controller[2]
3. Voyager Ingress controller[3] ที่ทำงานบนพื้นฐานของ HAProxy
4. GCE L7 load balancer controller (GLBC)[4] ซึ่งเป็นระบบโหลดบาลานซ์ที่เป็นค่าปริยายบน Google Cloud Platform

โดยจะทำการทดสอบบนระบบ Public Cloud เพื่อประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพ ทางด้านความสามารถในการให้บริการ ความเร็วในการตอบสนอง การใช้ทรัพยากรของระบบ และนำข้อมูลที่ได้จากการทดลองมาพิจารณา วิเคราะห์และสรุปผล

## 1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อทดสอบและปรียบเทียบประสิทธิภาพของ เลเยอร์-7 โหลดบาลานซ์ คอนโทรลเลอร์หรืออินเกรส คอนโทรลเลอร์ (Ingress controller) บนระบบคลัสเตอร์คูเบอร์เนทิส

---

[1] Kubernetes. (2018). *NGINX Ingress Controller.* Retrieved June 15, 2018, from https://kubernetes.github.io/ingress-nginx/

[2] Traefik. (2018). *Kubernetes Ingress Controller.* Retrieved June 15, 2018, from https://docs.traefik.io/

[3] AppsCode. (2018). *Voyager Secure HAProxy Ingress Controller for Kubernetes.* Retrieved June 15, 2018, from https://appscode.com/products/voyager/

[4] Kubernetes. (2018). *Ingress controller for Google Cloud.* Retrieved June 15, 2018, from https://github.com/kubernetes/ingress-gce/

### 1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะทำการทดสอบบนระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิส ที่ทำงานบน Public Cloud (Google Cloud Platform) โดยใช้จำนวนเครื่องที่นำมาต่อเป็นคลัสเตอร์เพื่อประมวลผลจำนวน 3 เครื่อง (Worker Node) ไม่นับรวมเครื่องที่ทำหน้าที่ Control Plane (Master Node) และอุปกรณ์หลักของระบบ (เช่น CPU, Memory) มีลักษณะเหมือนกันในทุก ๆ โหนด โดยมีรายละเอียด ขอบเขตงานวิจัย ดังนี้

1. ใช้ระบบ Kubernetes รุ่นเสถียร รหัสรุ่น 1.10.2 (GKE)
2. ใช้ระบบ Container Runtime เป็น Docker
3. Public Cloud ที่ใช้ในการทดสอบ ใช้บริการจาก Google Cloud Platform (GCP) โซนของประเทศสิงคโปร์ รหัสโซน asia-southeast1-a
4. เครื่องแม่ข่ายติดตั้งระบบปฏิบัติการ Ubuntu Linux Server
5. คอมพิวเตอร์ Workstation ติดตั้งระบบปฏิบัติการ Ubuntu Linux Desktop 18.04
6. ซอฟต์แวร์ที่ใช้ในการทดสอบ ใช้โปรแกรม Apache JMeter และโปรแกรม wrk2

### 1.4 สมมติฐานงานวิจัย

สามารถทดสอบเปรียบเทียบประสิทธิภาพและคุณสมบัติของ เลเยอร์-7 โหลดบาลานซ์คอนโทรลเลอร์ (อินเกรสคอนโทรลเลอร์) บนระบบคูเบอร์เนทิส และนำผลการทดสอบที่ได้จากการทดลองมาพิจารณา วิเคราะห์และสรุปผล เพื่อเป็นข้อมูลนำไปประกอบการตัดสินใจในการเลือกใช้ให้เหมาะสมกับรูปแบบที่ให้บริการเพื่อให้บริการแก่ผู้เข้าใช้บริการด้วยการเข้าถึงที่รวดเร็วและสามารถเข้าถึงบริการได้ตลอดเวลา

### 1.5 ประโยชน์ที่คาดว่าจะได้รับ

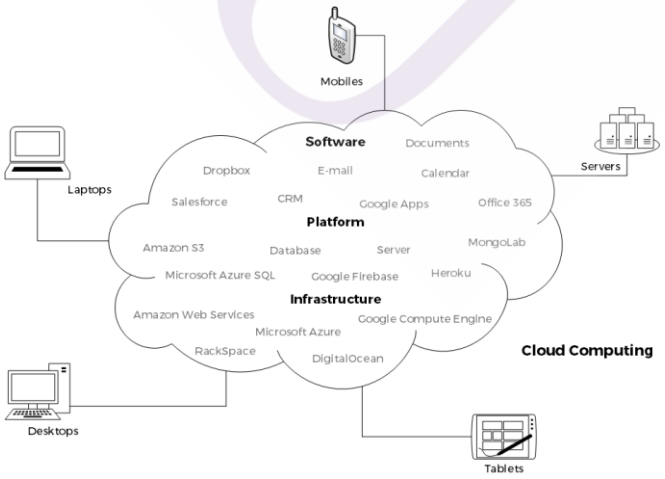
1. ได้ทราบถึงวิธีการทำงาน ประสิทธิภาพ คุณสมบัติ ข้อดี-ข้อเสีย ของอินเกรสคอนโทรลเลอร์ แต่ละระบบ
2. สามารถนำข้อมูลผลการทดสอบไปประกอบการตัดสินใจเลือกใช้อินเกรสคอนโทรลเลอร์ได้อย่างเหมาะสมกับระบบที่จะให้บริการ
3. ได้เรียนรู้การใช้งานระบบประมวลผลคอนเทนเนอร์และระบบคูเบอร์เนทิส
4. สามารถนำความรู้ทั้งหมดจากงานวิจัยชิ้นนี้มาประยุกต์ใช้งานจริง

# บทที่ 2
# ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องและนำมาประยุกต์ใช้ในงานวิจัย ซึ่งประกอบด้วย การประมวลผลแบบกลุ่มเมฆ (Cloud Computing), การประมวลผลแบบคอนเทนเนอร์ (Container), ระบบจัดการคอนเทนเนอร์คูเบอร์เนทิส (Kubernetes), ระบบกระจายภาระงาน (Load Balancer) และอินเกรสคอนโทรลเลอร์ (Ingress Controller)

## 2.1 การประมวลผลแบบกลุ่มเมฆ (Cloud Computing)

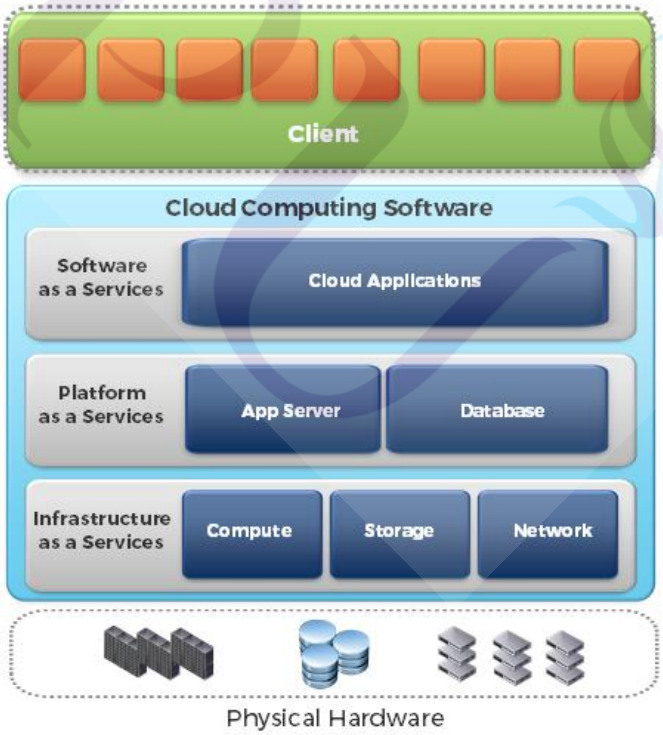
การประมวลผลหรือการให้บริการ ที่อ้างอิงตามความต้องการของผู้ใช้งาน โดยที่ผู้ใช้งานระบุความต้องการหรือเลือกรูปแบบที่เหมาะสมกับความต้องการไปยังระบบการประมวลผลแบบกลุ่มเมฆ หลังจากนั้นระบบจะจัดสรรทรัพยากรและบริการให้สอดคล้องกับความต้องการของผู้ขอใช้งาน ในขณะที่ผู้ใช้งานไม่ต้องมีความรู้ ความเชี่ยวชาญ หรือไม่จำเป็นต้องทราบถึงการทำงานเบื้องหลังของระบบว่าจะเป็นอย่างไร และในขณะที่ใช้งานสามารถปรับเปลี่ยนทรัพยากรที่ใช้งานได้อย่างสะดวกและรวดเร็ว สามารถเข้าใช้งานและเข้าถึงข้อมูลได้จากทุก ๆ ที่ทุกเวลา หรือจากทุก ๆ อุปกรณ์ ดังแสดงในภาพที่ 2.1




**ภาพที่ 2.1** การประมวลผลแบบกลุ่มเมฆ (Cloud Computing)

ในมุมมองของผู้ใช้งาน การประมวลผลแบบกลุ่มเมฆสามารถตอบสนองความต้องการได้อย่างมีประสิทธิภาพ ผู้ใช้งานสามารถเลือกใช้ทรัพยากรได้ตามความต้องการและเหมาะสมกับภาระงาน ทำให้เกิดประสิทธิภาพในการทำงาน ปัญหาในระหว่างการทำงานเนื่องจากการใช้ทรัพยากรที่ไม่เหมาะสมก็จะลดน้อยลง ผลลัพธ์ของงานก็จะเป็นไปตามเป้าหมายและผู้ใช้งานสามารถกำหนดงบประมาณค่าใช้จ่ายได้ล่วงหน้า สามารถวางแผนการใช้งานและงบประมาณได้อย่างเหมาะสม ไม่ต้องเสียค่าใช้จ่ายในการลงทุนติดตั้งระบบเอง และไม่ต้องเสียค่าใช้จ่ายที่เกินความจำเป็น

ในด้านผู้ให้บริการระบบนั้น ที่มาหรือแนวคิดของการประมวลผลแบบกลุ่มเมฆ อาจจะไม่ใช่แนวคิดใหม่ทั้งหมด เป็นการนำแนวคิดของการจำลองอุปกรณ์ (Virtualization technology) ทั้งการจำลองอุปกรณ์คอมพิวเตอร์ (Virtual machine) การจำลองระบบเครือข่าย (Network virtualization, Software-Defined Networking) แนวคิดการประมวลผลแบบกระจาย (Distributed computing) และระบบเครือข่ายและอินเตอร์เน็ต มาประยุกต์รวมเป็นระบบเดียวที่สามารถจัดการได้จากศูนย์กลาง ส่งผลให้สามารถเพิ่มหรือลดขนาดของระบบได้อย่างสะดวกง่ายดาย และรวดเร็ว ดังแสดงในภาพที่ 2.2




**ภาพที่ 2.2** เทคโนโลยีการประมวลผลแบบกลุ่มเมฆ

2.1.1 คุณลักษณะที่สำคัญของระบบประมวลผลแบบกลุ่มเมฆ[1]

2.1.1.1 On-demand self-service ผู้ใช้งานสามารถกำหนดความต้องการ และเรียกใช้งานทรัพยากรที่ต้องการ เช่น หน่วยประมวลผล หรือหน่วยจัดเก็บข้อมูล ได้เอง โดยไม่ต้องมีผู้ดูแลระบบจัดการให้

2.1.1.2 Broad network access สามารถเข้าใช้งานได้จากทุก ๆ อุปกรณ์ เช่น คอมพิวเตอร์ โทรศัพท์เคลื่อนที่ หรืออุปกรณ์พกพา ที่มีการเชื่อมต่อกับเครือข่าย

2.1.1.3 Resource pooling ทรัพยากรของระบบ เช่น หน่วยประมวลผล, หน่วยความจำ, หน่วยจัดเก็บข้อมูล และเครือข่าย จะถูกนำมารวมกันเป็นระบบที่ศูนย์กลาง เพื่อให้บริการกับผู้ใช้งานหลาย ๆ รายที่ขอใช้บริการและมีความต้องการทรัพยากรแตกต่างกัน โดยสามารถใช้งานได้อย่างอิสระต่อกัน

2.1.1.4 Rapid elasticity มีความสามารถที่จะปรับเปลี่ยน เพิ่ม-ลด ทรัพยากรได้อย่างสะดวกรวดเร็ว สามารถกำหนดให้มีการทำงานโดยอัตโนมัติได้ และปรับเปลี่ยนได้โดยไม่จำกัดจำนวนครั้งและระยะเวลา

2.1.1.5 Measured service มีการตรวจวัดปริมาณการใช้งานของทรัพยากรในระบบ ทำให้สามารถตรวจสอบ ควบคุมและจัดทำรายงานการใช้งานทรัพยากรของระบบได้อย่างสะดวกรวดเร็ว ผู้ใช้บริการทราบถึงความต้องการใช้ทรัพยากรของตนเอง และสามารถเลือกเสียค่าบริการได้เหมาะสมกับความต้องการ

2.1.2 รูปแบบการให้บริการระบบประมวลผลแบบกลุ่มเมฆ[2]

- Software as a Service (SaaS) เป็นการให้บริการ Application หรือ Software ที่ทำงานอยู่บนโครงสร้างพื้นฐานของระบบประมวลผลแบบกลุ่มเมฆ ผู้ใช้งานสามารถเข้าใช้งานได้จากหลากหลายอุปกรณ์ เช่น Web Browser หรือช่องทางที่กำหนดไว้เฉพาะของการให้บริการนั้น ๆ ผู้ใช้งานสามารถทำได้เพียงใช้งานตามที่กำหนดไว้เท่านั้น ไม่สามารถจัดการหรือปรับเปลี่ยนทรัพยากรที่ใช้งานได้ ตัวอย่างบริการ SaaS เช่น Microsoft Office 365, Google Apps, Dropbox, Salesforce เป็นต้น

---

[1] National Institute of Standards and Technology. (2011). *The NIST Definition of Cloud Computing* (pp. 6). Gaithersburg, MD: National Institute of Standards and Technology.

[2] แหล่งเดิม

- Platform as a Service (PaaS) เป็นการให้บริการระบบหรือสภาพแวดล้อม เพื่อใช้พัฒนา Software เช่นให้บริการระบบฐานข้อมูล ระบบจัดเก็บข้อมูล เป็นต้น ทำงานอยู่บนโครงสร้างพื้นฐานของระบบประมวลผลแบบกลุ่มเมฆ ผู้ใช้งานไม่สามารถจัดการหรือปรับเปลี่ยนทรัพยากรที่ใช้งานได้ สามารถใช้งานได้ตามที่ผู้ดูแลระบบจัดสรรให้เท่านั้น ตัวอย่างบริการ PaaS เช่น Microsoft Azure SQL Database, Amazon EC2, Amazon S3, MongoDB, Heroku, Google Firebase เป็นต้น

- Infrastructure as a Service (IaaS) เป็นการให้บริการระบบโครงสร้างพื้นฐานของระบบประมวลผลแบบกลุ่มเมฆ ผู้ใช้งานสามารถจัดการหรือปรับเปลี่ยนทรัพยากร เช่น หน่วยประมวลผล, หน่วยความจำ, หน่วยจัดเก็บข้อมูล, ระบบเครือข่าย และระบบอื่น ๆ ที่เกี่ยวข้องที่ใช้งานได้เอง หรือสร้างระบบงานได้ตามต้องการ แต่ผู้ใช้งานจะถูกจำกัดไม่ได้สิทธิ์เข้าจัดการถึงระดับ Hardware จริง ๆ การจัดการจะทำผ่านระบบเสมือนที่มีระบบจัดการช่วยอำนวยความสะดวกให้กับผู้ใช้งาน ตัวอย่างผู้ให้บริการ IaaS เช่น Microsoft Azure, Amazon Web Services (AWS), Google Cloud Platform, Rackspace เป็นต้น

2.1.3 รูปแบบการใช้งานระบบระบบประมวลผลแบบกลุ่มเมฆ[3]

- Private cloud เป็นระบบประมวลผลแบบกลุ่มเมฆ ที่มีการใช้งานเฉพาะภายในองค์กรเท่านั้น องค์กรเป็นเจ้าของและดูแลจัดการเอง อาจจะดำเนินการติดตั้งเองหรือใช้บริการจากผู้ให้บริการ และอาจจะติดตั้งไว้ภายในองค์กรหรือภายนอกองค์กรในสถานที่ของผู้ให้บริการ

- Public cloud เป็นระบบประมวลผลแบบกลุ่มเมฆ ที่เปิดให้ใช้งานทั่วไปต่อสาธารณะ อาจจะมีเจ้าของและดูแลโดยหน่วยงานธุรกิจเอกชน หน่วยงานราชการ สถาบันการศึกษา หรือเป็นเจ้าของร่วมกัน มีการติดตั้งภายนอกหน่วยงานในสถานที่ของผู้ให้บริการ

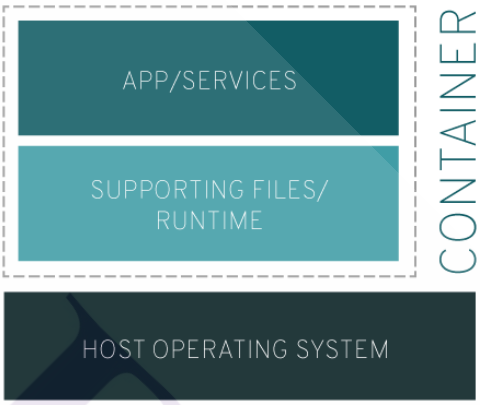
- Hybrid cloud เป็นระบบประมวลผลแบบกลุ่มเมฆที่ผสมผสานหรือรวมกันของ 2 ระบบหรือมากกว่า ระบบประมวลผลแบบกลุ่มเมฆที่แตกต่างกัน (Public, Private, Community) และสามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ

---

[3] National Institute of Standards and Technology. (2011). *The NIST Definition of Cloud Computing* (pp. 7). Gaithersburg, MD: National Institute of Standards and Technology.

## 2.2 การประมวลผลแบบคอนเทนเนอร์ (Container)

Container หรือ Software Container คือ การจำลองสภาพแวดล้อมที่เฉพาะให้กับซอฟต์แวร์เพื่อทำงานอย่างใดอย่างหนึ่งโดยที่ไม่รบกวนการทำงานของซอฟต์แวร์ตัวอื่น ๆ ที่ทำงานอยู่บนระบบปฏิบัติการเดียวกัน และสามารถที่จะนำเอา Software Container ไปทำงานบนเครื่องคอมพิวเตอร์หรือเครื่องแม่ข่ายเครื่องอื่น ๆ ได้โดยที่ระบบงานหรือซอฟต์แวร์ใน Container ยังสามารถทำงานได้เหมือนเดิม

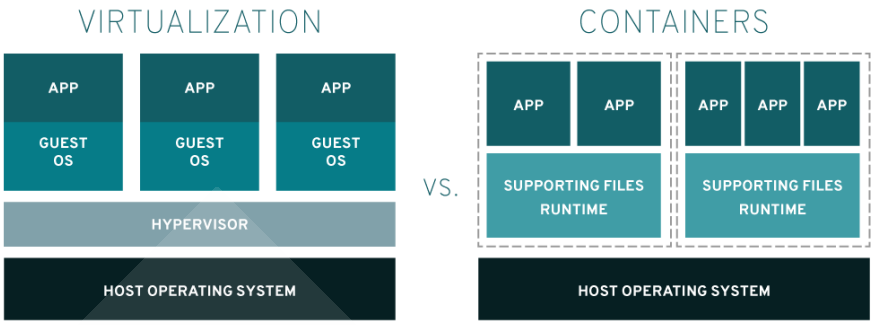



**ภาพที่ 2.3** แนวคิดของเทคโนโลยี Software Container[4]

แนวคิดการสร้างสภาพแวดล้อมจำลองให้กับซอฟต์แวร์หรือ Software Container นั้นจะแตกต่างจากแนวคิดของคอมพิวเตอร์เสมือน (Virtualization Technology) ที่เป็นเทคโนโลยีพื้นฐานของระบบประมวลผลแบบกลุ่มเมฆหรือ Cloud Computing เมื่อ Container สามารถทำงานได้หลาย ๆ Container บนระบบปฏิบัติการเดียวกัน แต่แนวคิดของคอมพิวเตอร์เสมือนคือการแบ่งเครื่องคอมพิวเตอร์จริง ๆ ออกเป็นเครื่องคอมพิวเตอร์เสมือนหลาย ๆ เครื่อง (Virtual Machines) ซึ่งแต่ละเครื่องเสมือนจำเป็นที่จะต้องติดตั้งระบบปฏิบัติการในทุก ๆ เครื่อง ซึ่งอาจจะทำให้สิ้นเปลืองเวลาในการติดตั้งระบบปฏิบัติการและเสียทรัพยากรให้กับระบบปฏิบัติการที่ต้องติดตั้งในทุก ๆ เครื่องคอมพิวเตอร์เสมือน

---

[4] Red Hat, Inc. *What is a Linux container?* Retrieved June 15, 2018, from https://www.redhat.com/en/topics/containers/whats-a-linux-container/




**ภาพที่ 2.4** เปรียบเทียบการทำงานของ Virtualization กับ Containers[5]

2.2.1 จุดเด่นของ Container เมื่อเปรียบเทียบกับ Virtual Machine[6]

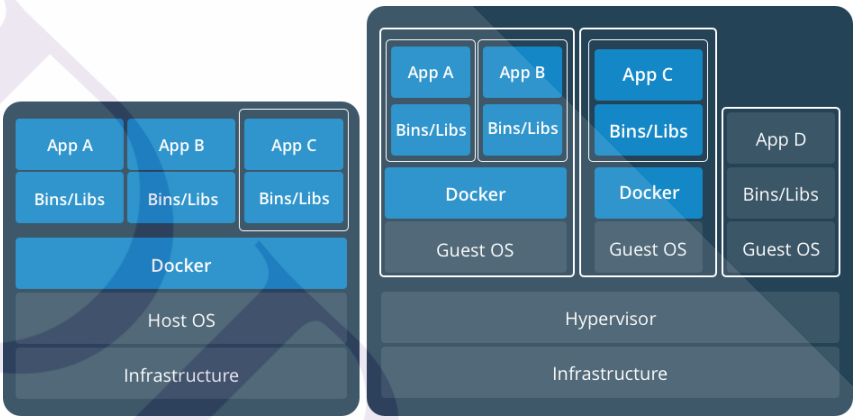
- สามารถรันโปรแกรมที่อยู่ใน Container ได้อย่างรวดเร็ว (เป็นหลักวินาที) เมื่อเปรียบเทียบกับการ Boot VM ที่ใช้เวลาหลายวินาทีจนถึงหลายนาที
- เสีย Overhead ของเครื่องเมื่อต้องติดตั้งระบบปฏิบัติการบน VM 5%-10% เมื่อระบบ Container มี Overhead ที่น้อยกว่า 1%
- ขนาด Image Size ของ VM จะใหญ่เป็นระดับ GB แต่ Image ของระบบ Container โดยทั่ว ๆ ไป จะมีขนาดเพียง 10 MB ถึง 500 MB เพียงเท่านั้น

---

[5] Red Hat, Inc. *What is a Linux container?* Retrieved June 15, 2018, from https://www.redhat.com/en/topics/containers/whats-a-linux-container/

[6] Chanwit Kaewkasi. (2559). *คอนเทนเนอร์ คืออะไร*. สืบค้น 15 มิถุนายน 2561, จาก https://sites.google.com/site/chanwit/blogs/what-is-container/

ถึงแม้ว่าเทคโนโลยี Virtualization, Cloud Computing และ Software Container จะมีแนวความคิดที่แตกต่างกัน แต่เมื่อนำเทคโนโลยีมาประยุกต์ใช้งานร่วมกันจะทำให้เกิดความสะดวกและรวดเร็วในการบริหารจัดการ โดยใช้ระบบคอมพิวเตอร์เสมือนที่สามารถจัดการได้อย่างรวดเร็วผ่านระบบ Cloud Computing เพื่อใช้ติดตั้งระบบประมวลผลแบบคอนเทนเนอร์ กล่าวอย่างง่าย คือ การสร้างระบบประมวลผลแบบคอนเทนเนอร์ เป็น Platform-as-a-Service ที่ทำงานอยู่บนระบบ Cloud Computing ซึ่งปัจจุบันมีให้บริการบน Public Cloud Provider โดยทั่วไป




**ภาพที่ 2.5** การประยุกต์ใช้งาน Containers ร่วมกับ Virtualization[7]

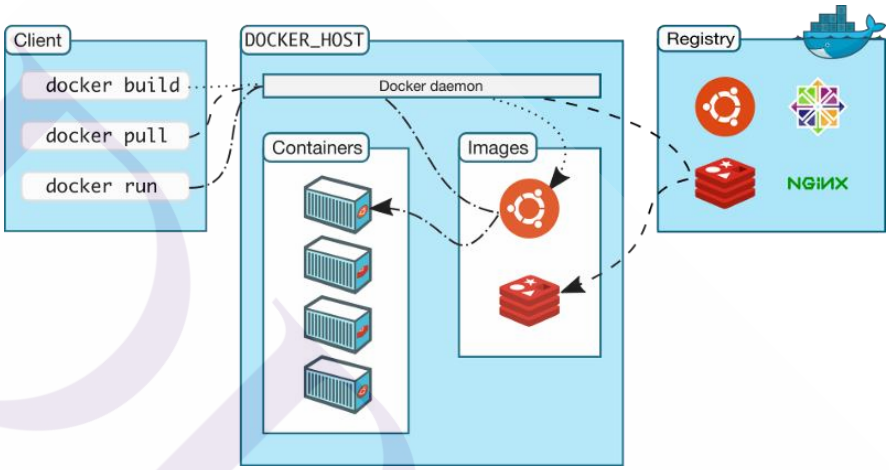
อันที่จริงแล้วแนวคิดการสร้างสภาพแวดล้อมจำลองหรือ Software Container มีการพัฒนาและใช้งานกันมานานแล้ว เช่น ทางฝั่ง Linux คือ Linux Container (LXC) หรือบนระบบปฏิบัติการ Solaris ที่เรียกว่า Solaris Containers แต่ไม่เป็นที่นิยมมากนัก เนื่องจากความยากในการใช้งาน แต่ในปัจจุบันมีชุมชนที่พัฒนาให้ระบบ Container สามารถใช้งานได้อย่างสะดวกและใช้งานง่าย เรียกว่า Docker Engine จึงทำให้ได้รับความนิยมในปัจจุบันเป็นอย่างมาก และถูกนำไปประยุกต์ใช้ในศูนย์ข้อมูลเป็นจำนวนมาก ซึ่ง Docker Engine นั้น ปัจจุบันมีทั้งเวอร์ชั่นที่แจกจ่ายให้ใช้งานฟรีโดยจำกัดคุณสมบัติบางประการ เรียกว่า Docker Community Edition (Docker CE) และเวอร์ชั่นที่ใช้งานระดับ Production ในองค์กรซึ่งมีค่าใช้จ่ายในการใช้งาน เรียกว่า Docker Enterprise Edition (Docker EE)

---

[7] Docker Inc. *What is a Container? A standardized unit of softwar*e. Retrieved June 15, 2018, from https://www.docker.com/what-container/

2.2.2 Docker Engine

เปิดตัวเริ่มใช้งานในปี ค.ศ.2013 ซึ่งหลังจากได้ทำการเปิดตัวแล้วจนถึงปัจจุบัน พบว่ามีการเติบโตอย่างรวดเร็ว ในปี ค.ศ.2017 พบว่า มีการ Pulls Image ใช้งานเป็นจำนวนถึง 12 พันล้านครั้ง (12,000,000,000) ซึ่งหลักการทำงานของ Docker Engine ที่ทำให้เป็นที่นิยมอย่างสูง คือ Build, Ship, Run Any App, Anywhere ที่เพิ่มความสะดวกสะบายให้กับนักพัฒนาและผู้ดูแลระบบเป็นอย่างมาก

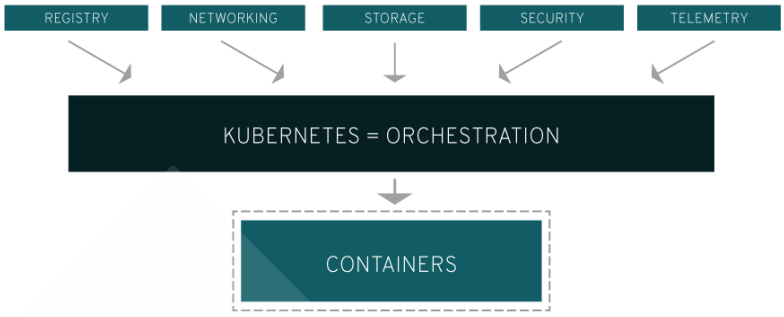



**ภาพที่ 2.6** สถาปัตยกรรมและการทำงานของ Docker Engine[8]

## 2.3 ระบบจัดการคอนเทนเนอร์คูเบอร์เนทิส (Kubernetes)

Kubernetes (k8s) คือ ระบบจัดการคอนเทนเนอร์ (Container Orchestration) ที่เปิดให้ใช้งานได้อย่างเสรี ไม่มีค่าใช้จ่าย ที่เริ่มพัฒนาโดย Google ใช้เพื่อจัดการการสร้างระบบที่ให้บริการแบบอัตโนมัติ (Automating deployment) เพิ่มหรือลดจำนวนคอมพิวเตอร์ที่ใช้เพื่อให้บริการได้อย่างรวดเร็ว (Scaling) จัดการการเรียกใช้ Software Containers รวมถึงการจัดการทรัพยากรต่าง ๆ เพื่อใช้งานร่วมกับระบบ Container เช่น Networking, Storage และ Security ภาพรวมการทำงานแสดงดังภาพที่ 2.7

[8] Docker Inc. *Docker overview*. Retrieved June 15, 2018, from https://www.docker.com/what-container/




**ภาพที่ 2.7** ภาพรวมการทำงานของ Kubernetes[9]

2.3.1 คุณลักษณะเด่นของระบบ Kubernetes[10]

- Automatic binpacking การเลือกใช้งาน Containers บนเครื่องที่เหมาะสม โดยพิจารณาจากข้อกำหนดและความต้องการ เพื่อการทำงานที่มีประสิทธิภาพสูงที่สุด
- Self-healing เรียกใช้งาน สร้างใหม่ หรือย้าย Containers ให้ครบตามจำนวนที่กำหนดไว้โดยอัตโนมัติ เมื่อมีเครื่องในระบบขัดข้อง
- Horizontal scaling เพิ่มหรือลดจำนวนเครื่องในการทำงานอย่างอัตโนมัติโดยพิจารณาจากทรัพยากรที่กำหนด เช่น CPU หรือสั่งงานเองด้วยคำสั่งอย่างง่าย
- Automated rollouts and rollbacks สามารถย้อนกลับไปใช้ซอฟต์แวร์รุ่นเก่าได้เมื่อเกิดการผิดพลาดจากการอัพเกรดซอฟต์แวร์ที่ให้บริการ
- Secret and configuration management มีระบบจัดการตั้งค่าและจัดเก็บข้อมูลลับเพื่อความปลอดภัยของระบบ

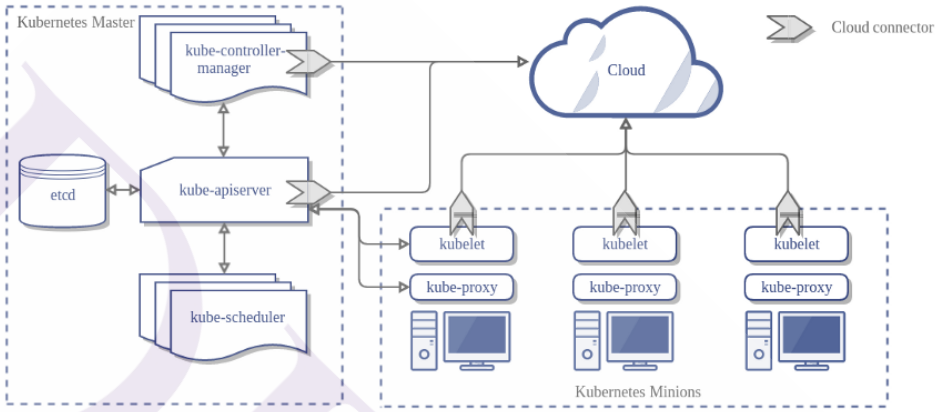
---

[9] Red Hat, Inc. *What is Kubernetes?* Retrieved June 15, 2018, from https://www.redhat.com/en/topics/containers/what-is-kubernetes/

[10] The Linux Foundation. *Kubernetes: Production-Grade Container Orchestration.* Retrieved June 15, 2018, from https://kubernetes.io/

2.3.2 สถาปัตยกรรมของ Kubernetes

Kubernetes เป็นระบบจัดการ Container ที่ทำงานในลักษณะของคลัสเตอร์ ที่ใช้เครื่องคอมพิวเตอร์หลาย ๆ เครื่องมาเชื่อต่อกันเพื่อทำงานร่วมกัน กระจายภาระงานเพื่อเสถียรภาพของระบบ สามารถให้บริการได้ตลอดเวลาแม้มีเครื่องใดในระบบขัดข้อง และสามารถขยายระบบได้อย่างสะดวกรวดเร็ว




**ภาพที่ 2.8** สถาปัตยกรรมของระบบคลัสเตอร์ Kubernetes[11]

Kubernetes Cluster จะประกอบไปด้วย Master Node ที่ทำหน้าที่เป็น Control Plane และ Worker Node (หรือ Minions Node) ที่ทำหน้าที่ในการรัน Containerized แอปพลิเคชัน คู่มืออย่างเป็นทางการของ Kubernetes แนะนำให้มี Master Node อย่างน้อย 1 Node และ Worker Node จำนวน 3 Nodes หรือมากกว่า ต่อ หนึ่งคลัสเตอร์

[11] The Linux Foundation. *Concepts Underlying the Cloud Controller Manager.* Retrieved June 15, 2018, from https://kubernetes.io/docs/concepts/architecture/cloud-controller/

2.3.3 ส่วนประกอบ บน Master Node

2.3.3.1 etcd เป็นฐานข้อมูลในลักษณะ key-value ใช้สำหรับจัดเก็บการตั้งค่าต่าง ๆ ของคลัสเตอร์ สามารถเข้าถึงข้อมูลที่จัดเก็บไว้ได้จากทุกเครื่องในคลัสเตอร์ โดยเข้าถึงผ่านช่องทาง HTTP/JSON API เพื่อขอข้อมูลที่ต้องการ

2.3.3.2 kube-apiserver เป็นส่วนประกอบที่สำคัญที่ทำหน้าที่เป็น API Server แบบ RESTful เพื่ออ่านค่าการตั้งค่าต่าง ๆ ของคลัสเตอร์จาก etcd และทำหน้าที่เป็นส่วนติดต่อผู้ใช้งาน ที่ใช้งานผ่านโปรแกรม kubectl

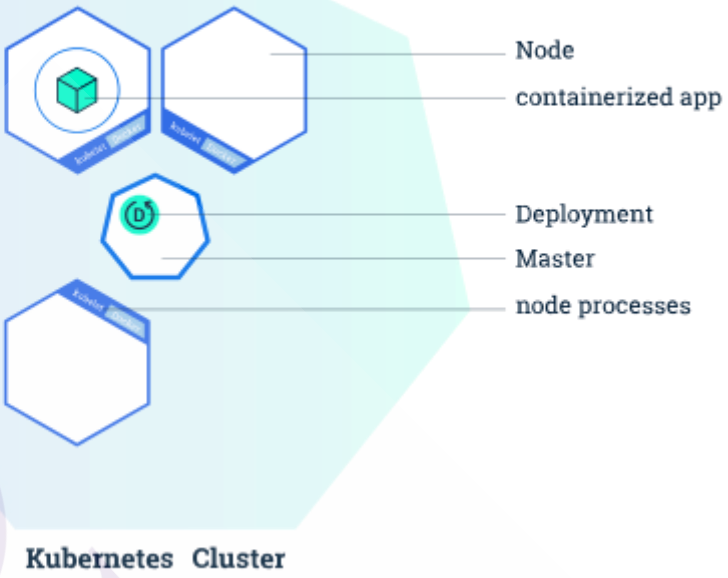
2.3.3.3 kube-scheduler เป็นส่วนประกอบที่ทำหน้าที่จัดการ Workloads ของระบบ กำหนดการทำงานของ Containerized แอปพลิเคชัน ให้กับเครื่องในคลัสเตอร์ที่เหมาะสมโดยวิเคราะห์และพิจารณาจากทรัพยากรและข้อกำหนดต่าง ๆ คอยติดตามการเปลี่ยนแปลงของระบบเพื่อจัดการหการทำงานของระบบให้มีประสิทธิภาพมากที่สุด

2.3.4 ส่วนประกอบ บน Worker Nodes

2.3.4.1 Container Runtime เป็นส่วนประกอบที่จำเป็นต้องติดตั้งไว้ในทุก ๆ เครื่องที่เป็น Worker Node เพื่อทำหน้าที่ในการทำงานและจัดการ Containers โดยทั่วไปจะติดตั้ง Docker Engine เป็น Runtime ของระบบ แต่สามารถเลือกใช้งาน Container Runtime อื่น ๆ ได้ เช่น rkt หรือ runc

2.3.4.2 kubelet ทำหน้าที่เป็นส่วนติดต่อกับ Control Plane เพื่อติดต่อสื่อสารข้อมูลจากฐานข้อมูล etcd ของเครื่อง Master

2.3.4.3 kube-proxy เป็นส่วนประกอบที่ทำหน้าที่จัดการระบบสื่อสาร (networking) จัดการ subnet ส่งต่อข้อมูลให้กับเครื่องที่เหมาะสม และทำหน้าที่เป็น ตัวกระจายโหลดอย่างง่ายให้กับ Services ต่าง ๆ ที่ทำงานอยู่บนคลัสเตอร์




**ภาพที่ 2.9** ส่วนประกอบของ Kubernetes Cluster[12]

2.3.5 หน่วยการทำงานของ Kubernetes

2.3.5.1 Pods เป็นหน่วยการทำงานที่เล็กที่สุดของระบบคลัสเตอร์ Kubernetes มีหน้าที่ในการเริ่มการทำงาน และจัดการการทำงาน ของ Container Image หรือ Containerized application โดยที่ใน 1 pod อาจจะมีหลาย ๆ container ทำงานอยู่ภายในและแต่ละ pod จะมี IP Address เป็นของตนเอง

2.3.5.2 Replication Controller และ Replication Set เป็นส่วนที่จัดการจำนวน, การทำสำเนา pod, การเพิ่มหรือลด pods (Horizontal Scaling) โดยที่ Replication Controller เป็นหน่วยการทำงานแบบเก่า และ Replication Set เป็นส่วนการทำงานที่ปรับปรุงขึ้นมาจาก Replication Controller ให้สามารถปรับค่าได้ยืดหยุ่นมากขึ้น โดยทั่วไป Replication Set จะถูกสร้างขึ้นพร้อมกับและเป็นส่วนหนึ่งของ Deployment

---

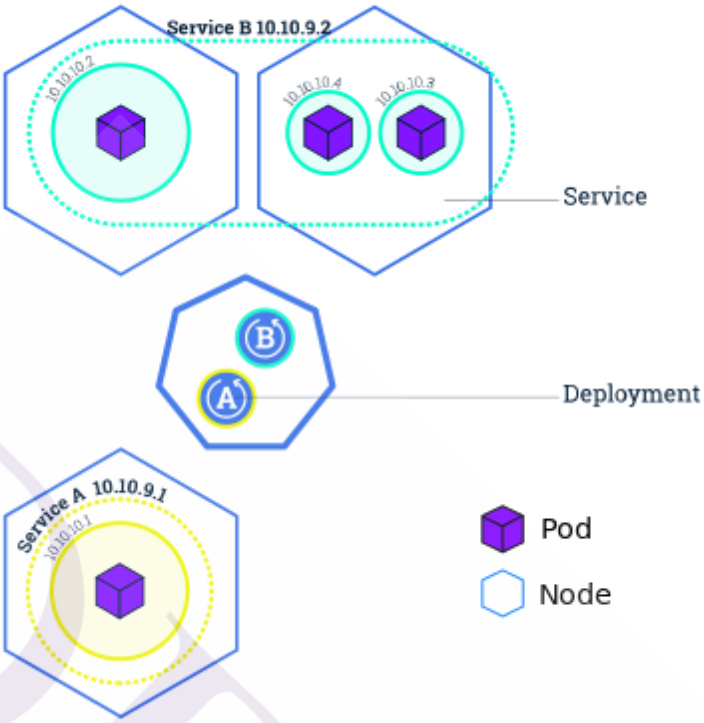
[12] The Linux Foundation. *Using kubectl to Create a Deployment.* Retrieved June 15, 2018, from https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

2.3.5.3 Deployment เป็นส่วนที่จัดการและควบคุมการอัพเดทหรือถอยกลับเวอร์ชั่นเมื่อเกิดปัญหาของแอปพลิเคชัน โดยปกติเมื่อทำการสร้าง Deployment จะสร้าง Replication Sets และ pods ขึ้นทำงานพร้อมกัน

2.3.5.4 Services เป็นส่วนที่ใช้ติดต่อระหว่างผู้ใช้บริการหรือการเรียกใช้บริการกับคอนเทนเนอร์แอปพลิเคชันที่ทำงานอยู่บนคลัสเตอร์ที่ปกติแล้วจะสามารถเรียกใช้ได้เฉพาะใน pod เดียวกันเท่านั้น

2.3.5.5 Volumes และ Persistent Volumes เป็นส่วนจัดเก็บข้อมูลของคลัสเตอร์เพื่อไม่ให้ข้อมูลสูญหาย เพราะโดยปกติแล้วถ้าไม่มีการใช้ Volumes เพื่อจัดเก็บข้อมูล เมื่อทำการลบ pod ออกจากระบบจะทำให้ข้อมูลถูกลบไปพร้อมกับ pod

2.3.5.6 Labels และ Annotations เป็นส่วนการจัดการป้ายชื่อหรือชื่อเรียกของหน่วยการทำงานเพื่อความสะดวกในการจัดกลุ่มและเรียกใช้ และ Annotations เป็นการกำหนดการตั้งค่าพิเศษของหน่วยการทำงานเมื่อจำเป็นต้องกำหนดค่าเพิ่มเติม

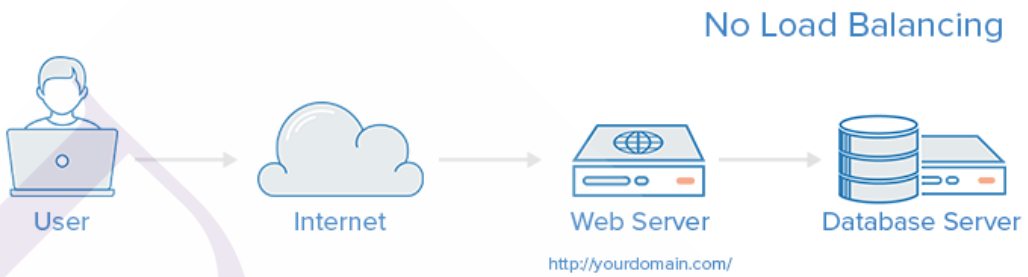



**ภาพที่ 2.10** หน่วยการทำงานของ Kubernetes[13]

[13] The Linux Foundation. *Using a Service to Expose Your App.* Retrieved June 15, 2018, from https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/

## 2.4 ระบบกระจายภาระงาน (Load Balancer)

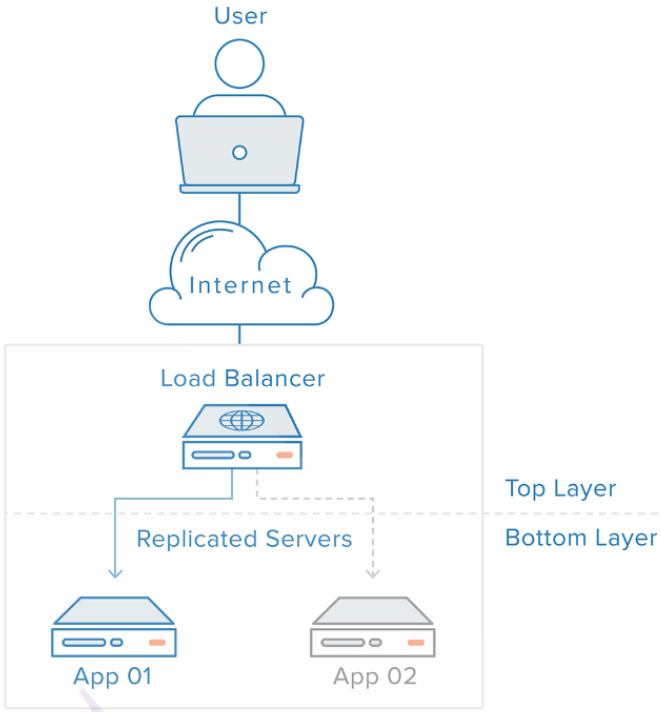
Load Balancer คือระบบที่ทำหน้าที่ในการกระจายภาระงาน (load, request) ไปยังเครื่องหรือระบบที่ให้บริการจำนวนหลาย ๆ เครื่องที่เชื่อมต่อกันเพื่อทำหน้าที่ที่เหมือนกัน เพื่อแบ่งเบาภาระการทำงานทำให้สามารถทำงานได้อย่างมีประสิทธิภาพและสามารถรองรับการทำงานได้เป็นจำนวนมาก




**ภาพที่ 2.11** ตัวอย่างระบบที่ให้บริการโดยไม่มี Load Balancer[14]

โดยทั่วไประบบ Load Balance จะนิยมนำมาใช้กับการให้บริการเว็บไซต์ หรือเครื่องแม่ข่ายที่ให้บริการต่าง ๆ ที่ให้บริการผ่านอินเตอร์เน็ต เช่น DNS Server, NTP Server หรือ API Server เพื่อให้สามารถรองรับผู้ใช้งานได้เป็นจำนวนมาก

[14] DigitalOcean Inc. (2017). *What is Load Balancing?* Retrieved June 15, 2018, from https://www.digitalocean.com/community/tutorials/what-is-load-balancing/




**ภาพที่ 2.12** ตัวอย่างระบบที่ให้บริการโดยมีระบบ Load Balancer[15]

2.4.1 Load Balancing Algorithms

2.4.1.1 Round Robin เป็นการกระจายโหลดในลักษณะวนภายในกลุ่ม เช่น เมื่อมีจำนวนเครื่องในกลุ่มที่ให้บริการในระบบ Load Balance จำนวน 3 เครื่อง เมื่อเรียกใช้ครั้งที่ 1 ก็จะส่งไปให้เครื่องที่ 1 ครั้งที่ 2 ก็จะส่งไปให้เครื่องที่ 2 ครั้งที่ 3 ก็จะส่งไปให้เครื่องที่ 3 เมื่อครบตามจำนวนเครื่องทั้งหมดแล้วก็จะเริ่มที่เครื่องที่ 1 ใหม่อีกครั้ง

2.4.1.2 Least Connections เป็นการกระจายโหลดโดยเลือกจากเครื่องที่ภาระงานน้อยที่สุดเมื่อภาระงานน้อยก็จะส่งผลให้มีทรัพยากรเหลือที่จะให้บริการเยอะกว่าเครื่องอื่น ๆ

2.4.1.3 Source IP หรือ Hash IP เป็นการกระจายโหลดโดยเลือกจากความสัมพันธ์ระหว่าง IP Address ของผู้ใช้บริการเพื่อให้มั่นใจได้ว่าผู้ใช้บริการได้ติดต่อกับเครื่องแม่ข่ายที่ต้องการจริง ๆ

---

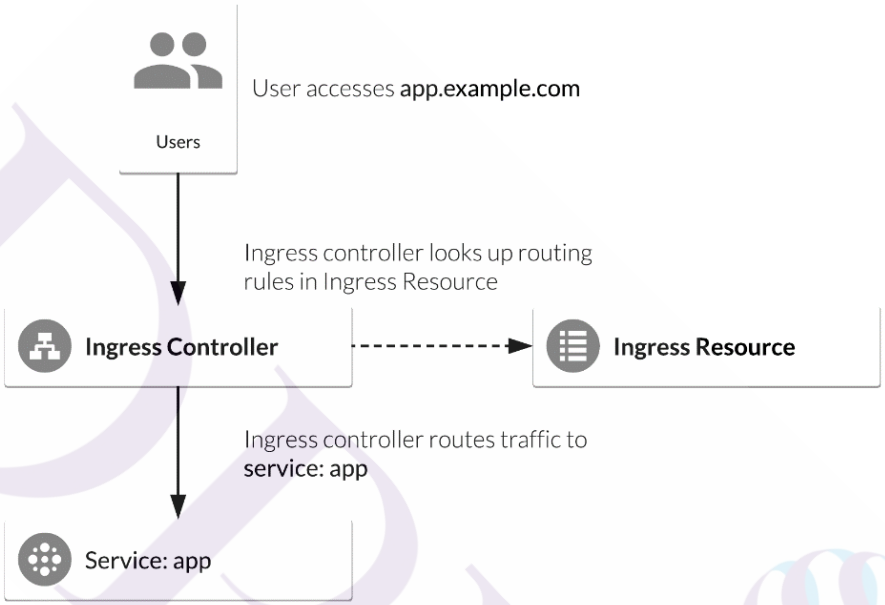
[15] DigitalOcean Inc. (2017). *What is Load Balancing?* Retrieved June 15, 2018, from https://www.digitalocean.com/community/tutorials/what-is-load-balancing/

2.4.2 ประโยชน์ของการทำ Load Balance

- สามารถรองรับการเข้าใช้งานของระบบได้เป็นจำนวนมาก
- ผู้เข้าใช้บริการจะไม่สามารถทราบจำนวนเครื่องที่ให้บริการได้ เป็นการเพิ่มความปลอดภัยให้กับระบบ
- สามารถให้บริการได้ตลอดเวลาถึงแม้ว่าจะมีเครื่องในระบบเกิดความขัดข้องไม่สามารถทำงานได้ก็ไม่ส่งผลกระทบให้ระบบล่ม
- ง่ายต่อการจัดการ ในกรณีที่เกิดเครื่องในระบบเกิดความขัดข้องก็สามารถถอดออกจากระบบเพื่อมาแก้ไขปัญหาได้โดยที่ไม่ส่งผลกระทบต่อระบบโดยรวม
- สามารถ เพิ่ม/ลด ขนาดของระบบที่ให้บริการได้อย่างง่าย (Scaling) หรือสามารถเพิ่ม/ลด ขนาดของระบบได้โดยอัตโนมัติตามภาระงาน

## 2.5 อินเกรสคอนโทรลเลอร์ (Ingress Controller)

Ingress Controller คือ Layer-7 Load Balance Controller ที่ทำหน้าที่กระจายภาระงานและกำหนดเส้นทางในการเข้าใช้งาน Service ที่ให้บริการบนระบบคลัสเตอร์คูเบอร์เนทิสในระดับแอปพลิเคชัน ทำให้รองรับการทำงานบนโปรโตคอล HTTP, HTTPS, TCP และ UDP




**ภาพที่ 2.13** การทำงานของ Ingress Controller[16]

[16] Ameer Abbas. (2018). *Ingress with NGINX controller on Google Kubernetes Engine*. Retrieved June 15, 2018, from https://cloud.google.com/community/tutorials/nginx-ingress-gke/

การทำงานของ Service บนระบบคลัสเตอร์คูเบอร์เนทิส สามารถเลือกกำหนดการเข้าถึงบริการได้หลายวิธีโดยแต่ละวิธีก็มีข้อดี-ข้อเสีย แตกต่างกันไป สามารถสรุปได้ดัง ตารางที่ 2.1

**ตารางที่ 2.1** ผลการเปรียบเทียบ ข้อดี-ข้อเสีย ของรูปแบบการเข้าใช้งาน Service บน Kubernetes

| รูปแบบ | ข้อดี | ข้อเสีย |
|---|---|---|
| kube-proxy | สะดวกต่อการเรียกใช้งาน (built-in) | การใช้งานยุ่งยาก |
| ClusterIP | เหมาะกับระบบที่ใช้ภายในคลัสเตอร์ | ไม่สามารถเข้าถึงจากภายนอกได้ |
| NodePort | สะดวก, เข้าใช้งานง่าย | ความปลอดภัยต่ำ (เปิดพอร์ต) |
| LoadBalancer | สะดวก, เสถียร | ต้องใช้บริการบนระบบที่มีบริการ |
| Ingress | เร็ว, คุณสมบัติเยอะ, ปรับแต่งได้มาก | ต้องติดตั้งระบบเพิ่มเติม |

จากตารางที่ 2.1 ข้างต้น จะพบว่า Ingress Controller มีคุณสมบัติหลากหลายที่สามารถปรับแต่งให้เหมาะสมกับระบบให้บริการที่ต้องการได้ ซึ่งคุณสมบัติสำคัญที่ช่วยเพิ่มประสิทธิภาพของการให้บริการ มีดังนี้

- Path-based / Name-based routing สามารถกำหนดเส้นทางการเข้าใช้งานได้อย่างอิสระและหลากหลาย ตามชื่อหรือพาธที่กำหนด
- Content-based routing สามารถกำหนดการกระจายโหลดได้จากรูปแบบของข้อมูลที่ให้บริการได้
- SSL termination เพิ่มความปลอดภัยให้กับระบบที่ให้บริการด้วยการใช้งานร่วมกับ HTTPS ได้อย่างง่ายดาย
- Active health checks มีระบบตรวจสอบความพร้อมใช้งานของระบบเบื้องหลังที่ให้บริการ
- Security สามารถปรับแต่งเพื่อเพิ่มความปลอดภัยให้กับระบบ
- Logging บันทึกเพื่อตรวจสอบการทำงานของระบบ
- Real-time statistics แสดงสถิติการใช้งาน

เมื่อ Ingress Controller ที่เลือกใช้งานมีประสิทธิภาพที่ดี ก็จะส่งผลถึงประสิทธิภาพในการให้บริการที่ดีตามไปด้วย ทำให้สามารถให้บริการได้อย่างรวดเร็วและมีการกระจายภาระงานอย่างเหมาะสมเพื่อประสิทธิภาพโดยรวมของระบบที่สูงที่สุด

## 2.6 งานวิจัยที่เกี่ยวข้อง

2.6.1 งานวิจัยเรื่อง Performance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat[17]

งานวิจัยนี้นำเสนอการสร้างระบบให้บริการ Web Server ที่ติดตั้งระบบจัดการกระจายภาระงานด้วยโปรแกรม Heartbeat และ HAProxy บนเครื่อง Bare-metal เพื่อทดสอบและเปรียบเทียบประสิทธิภาพของอัลกอริทึมในการกระจายภาระงาน จำนวน 3 รูปแบบ คือ

1.) Round Robin

2.) Least Connection

3.) Source IP

ดำเนินการทดสอบดัชนีประสิทธิภาพในด้านต่าง ๆ ดังนี้

1.) Connection rate

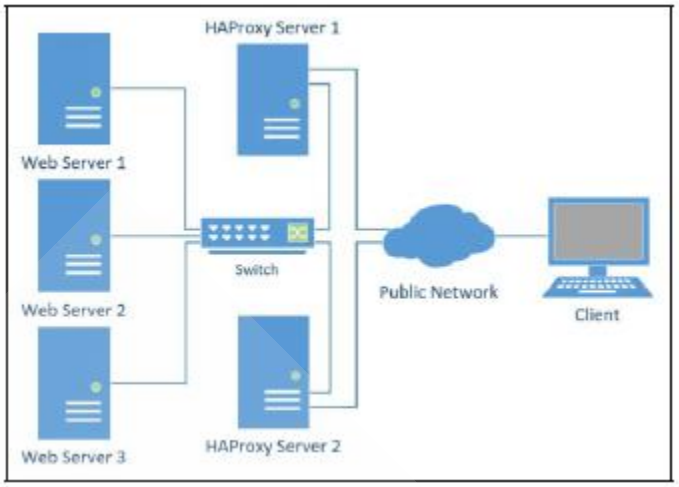
2.) Response time

3.) Throughput (Data rate)

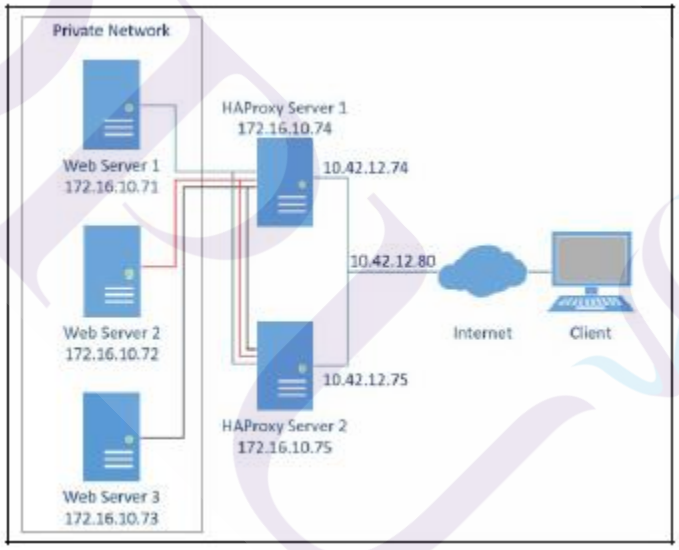
4.) Failed connections

โดยมีรูปแบบในการเชื่อมต่อของระบบที่ใช้ทดสอบในงานวิจัย แสดงดังภาพที่ 2.14 และภาพที่ 2.15

[17] Agung B. Prasetijo, Eko D. Widianto & Ersya T. Hidayatullah. (2016). *Performance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat.*




**ภาพที่ 2.14** รูปแบบการเชื่อมต่อทางกายภาพของระบบที่ใช้ทดสอบ

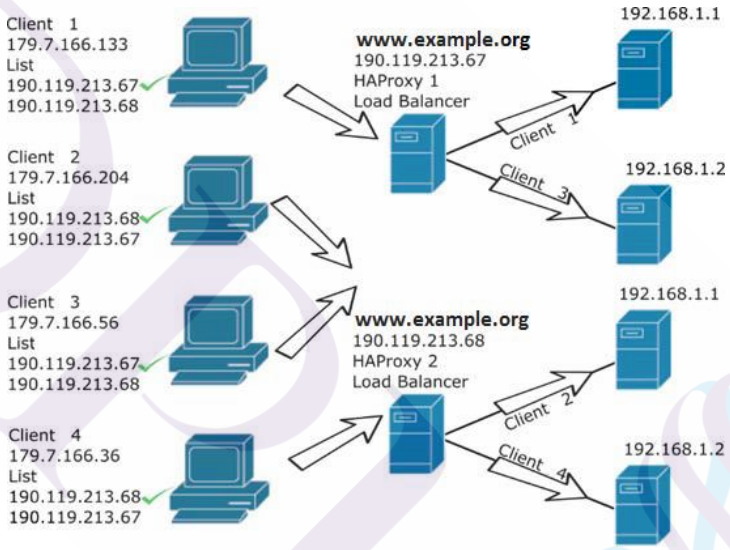



**ภาพที่ 2.15** รูปแบบการเชื่อมต่อเครือข่ายของระบบที่ใช้ทดสอบ

ผลการทดสอบพบว่าอัลกอริทึมแบบ Least Connection มีประสิทธิภาพที่ดีกว่าอีก 2 อัลกอริทึม (Round Robin และ Source IP) ในทุก ๆ การทดสอบ ที่ดำเนินการทดสอบทั้ง 4 ดัชนีประสิทธิภาพ

2.6.2 งานวิจัยเรื่อง Design of a High Availability System with HAProxy and Domain Name Service for Web Services[18]

งานวิจัยนี้นำเสนอการออกแบบระบบ High Availability โดยใช้โปรแกรม HAProxy ที่เลือกใช้อัลกอริทึม Round Robin ในการกระจายภาระงาน เพื่อปรับปรุงความสามารถในการให้บริการและนำไปใช้งานกับระบบให้บริการ Domain Name Service (DNS) ที่ทำหน้าที่เพื่อให้บริการร่วมกับระบบ Web Services และทำการทดสอบเพื่อตรวจสอบการกระจายภาระงานและความสามารถในการให้บริการของระบบ โดยมีรูปแบบในการเชื่อมต่อของระบบที่ใช้ทดสอบในงานวิจัย แสดงดังภาพที่ 2.16




**ภาพที่ 2.16** ระบบที่ออกแบบโดยใช้ HAProxy (ใช้อัลกอริทึม Round Robin) ร่วมกับ DNS

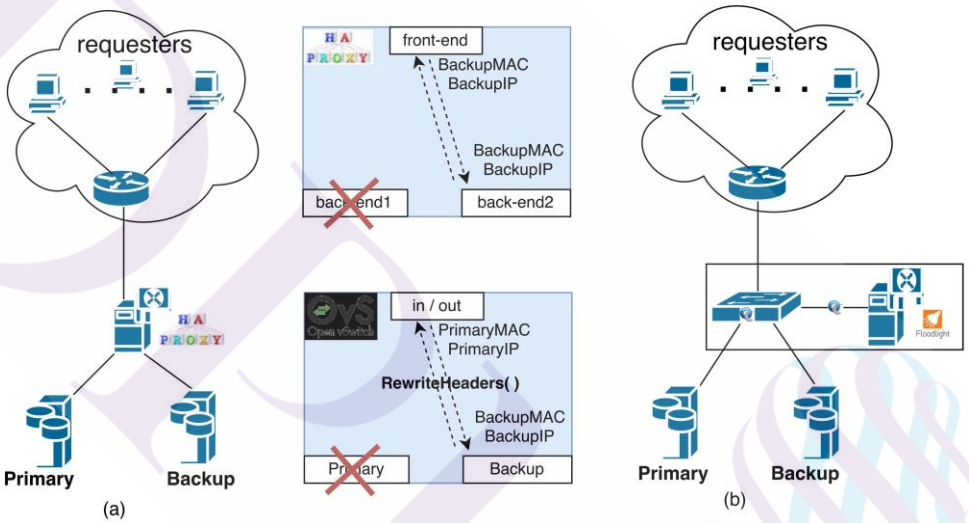
ผลการทดสอบพบว่าระบบ High Availability ที่ออกแบบ สามารถกระจายภาระงานไปสู่เครื่องแม่ข่ายได้อย่างเท่า ๆ กัน โดยที่จำนวนภาระงานในแต่ละเครื่องแม่ข่ายเท่ากับจำนวนการร้องขอข้อมูลทั้งหมดหารด้วยจำนวนเครื่องแม่ข่าย และระบบมีความสามารถในการให้บริการเท่ากับ ร้อยละ 99.905

---

[18] José Emmanuel Cruz de la Cruz & Christian Augusto Romero Goyzueta. (2017). *Design of a High Availability System with HAProxy and Domain Name Service for Web Services.*

2.6.3 งานวิจัยเรื่อง Evaluating Traffic Redirection Mechanisms for High Availability Servers[19]

งานวิจัยนี้นำเสนอการสร้างระบบ High Availability เพื่อประเมินสมรรถนะของระบบ Reverse proxy ที่ทำหน้าที่เปลี่ยนเส้นทางของข้อมูลเมื่อเครื่องแม่ข่ายหลักไม่สามารถให้บริการได้ จำนวน 2 ระบบ คือ ระบบที่ทำงานด้วย HAProxy-based และระบบที่ทำงานด้วย SDN-based โดยใช้ OpenFlow เพื่อทดสอบเปรียบเทียบประสิทธิภาพและความสามารถในการให้บริการในกรณีที่เครื่องแม่ข่ายหลักไม่สามารถให้บริการได้ โดยมีรูปแบบในการเชื่อมต่อของระบบที่นำเสนอในงานวิจัย แสดงดังภาพที่ 2.17




**ภาพที่ 2.17** การทำงานของ (a) HAProxy-based redirection (b) SDN-based redirection

ผลการทดสอบพบว่า เครื่องแม่ข่ายสำรอง ที่ใช้ในระบบ SDN-based สามารถตอบสนองเมื่อเครื่องแม่ข่ายหลักไม่สามารถให้บริการได้ รวดเร็วกว่าในระบบที่ใช้ HAProxy และอัตราการเชื่อมต่อล้มเหลวบนระบบ HAProxy สูงกว่าระบบ SDN-based เป็นผลให้ความสามารถในการให้บริการเฉลี่ย (Average throughput) ของระบบ SDN-based สูงกว่า ระบบที่ใช้ HAProxy

[19] Elias Konidis, Panagiotis Kokkinos & Emmanouel Varvarigos. (2016). *Evaluating Traffic Redirection Mechanisms for High Availability Servers.*

# บทที่ 3
# ระเบียบวิธีวิจัย

ในบทนี้จะกล่าวถึงแนวทางการดำเนินการวิจัย เครื่องมือที่นำมาใช้ แผนการดำเนินงาน ขั้นตอนและวิธีการดำเนินงาน และการออกแบบและติดตั้งระบบเพื่อประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพของอินเกรสคอนโทรลเลอร์

## 3.1 แนวทางการดำเนินการวิจัย

การดำเนินการวิจัย จะสร้างระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสขึ้นมาบน Public Cloud (Google Cloud Platform) โดยใช้ Google Kubernetes Engine จำนวน 1 คลัสเตอร์ โดยเป็นคลัสเตอร์ที่มี Master Node จำนวน 1 Node และ Worker Node จำนวน 3 Node และติดตั้ง Ingress Controller ทั้ง 4 รูปแบบ คือ

1. Nginx Ingress Controller
2. Traefik Ingress Controller
3. Voyager Ingress Controller
4. GCE L7 load balancer controller (GLBC)

ไว้บนคลัสเตอร์ และทำการทดสอบโดยใช้โปรแกรม Apache JMeter[1] และ wrk2[2] และบันทึกผลการทดลอง เพื่อนำไปวิเคราะห์ผลการทดลอง ประเมินสมรรถนะและเปรียบเทียบประสิทธิภาพ

---

[1] Apache Software Foundation. *Apache JMeter™.* Retrieved June 15, 2018, from https://jmeter.apache.org/

[2] Gil Tene (giltene). *A constant throughput, correct latency recording variant of wrk.* Retrieved June 15, 2018, from https://github.com/giltene/wrk2/

**3.2 แผนการดำเนินการวิจัยและพัฒนาระบบ**

3.2.1 ศึกษา ค้นคว้าและรวบรวมทฤษฎีที่เกี่ยวข้อง การประยุกต์ใช้งานของระบบประมวลผลแบบกลุ่มเมฆ (Cloud Computing)

- ภาพรวมของระบบ
- การใช้งานและการประยุกต์
- ประโยชน์ของการใช้งานระบบประมวลผลแบบกลุ่มเมฆ
- การเข้าใช้งานและความปลอดภัย
- ซอฟต์แวร์ที่นิยมนำมาใช้งาน

3.2.2 ศึกษา ค้นคว้าและรวบรวมทฤษฎีที่เกี่ยวข้อง การประยุกต์ใช้งานของระบบประมวลผลแบบคอนเทนเนอร์ (Software Container)

- ทฤษฎีและการนำมาประยุกต์ใช้
- ประโยชน์ของการใช้งานระบบประมวลผลแบบคอนเทนเนอร์
- การใช้งาน Container ร่วมกับ Cloud Computing
- การติดตั้งและการใช้งาน

3.2.3 ศึกษาและค้นคว้าทฤษฎีที่เกี่ยวข้อง

- ระบบกระจายภาระงาน (Load Balancer)
- เลเยอร์-7 โหลดบาลานซ์ คอนโทรลเลอร์ หรือ อินเกรสคอนโทรลเลอร์ (Ingress Controller)
- ระบบจัดการคอนเทนเนอร์คูเบอร์เนทิส (Kubernetes)

3.2.4 ค้นคว้าระเบียบวิธีวิจัย และงานวิจัยที่เกี่ยวข้อง

3.2.5 ออกแบบ พัฒนาและติดตั้งระบบเพื่อทำการทดสอบ

3.2.6 ทำการทดสอบและเก็บผลการทดลองตามแผนงานที่ได้วางแผนไว้

3.2.7 วิเคราะห์ เปรียบเทียบและสรุปผลการทดลอง

3.2.8 จัดทำเอกสารการวิจัย (วิทยานิพนธ์)

โดยมีสรุปแผนการดำเนินการวิจัยและพัฒนาระบบ ดังแสดงในตารางที่ 3.1

**ตารางที่ 3.1** แผนการดำเนินงานวิจัย

| ระยะเวลา / การดำเนินงาน | ม.ค. – ก.พ. 2561 | ก.พ. – มี.ค. 2561 | มี.ค. – เม.ย. 2561 | เม.ย. – พ.ค. 2561 | พ.ค. – มิ.ย. 2561 | มิ.ย. – ก.ค. 2561 | ก.ค. – ส.ค. 2561 | ส.ค. – ก.ย. 2561 |
|---|---|---|---|---|---|---|---|---|
| ศึกษา ค้นคว้าข้อมูล และการประยุกต์ใช้งานของระบบประมวลผลแบบกลุ่มเมฆ | | | | | | | | |
| ศึกษา ค้นคว้าข้อมูล และการประยุกต์ใช้งานของระบบประมวลผลแบบคอนเทนเนอร์ | | | | | | | | |
| ศึกษาและค้นคว้าทฤษฎี ระเบียบวิธีวิจัย งานวิจัยที่เกี่ยวข้อง | | | | | | | | |
| ออกแบบ พัฒนาและติดตั้งระบบเพื่อทำการทดสอบ | | | | | | | | |
| ทำการทดสอบและเก็บผลการทดลองตามแผนงานที่ได้วางแผนไว้ | | | | | | | | |
| วิเคราะห์ เปรียบเทียบและสรุปผลการทดลอง | | | | | | | | |
| รวบรวมข้อมูลทั้งหมดเพื่อจัดทำเอกสารการวิจัย (วิทยานิพนธ์) | | | | | | | | |

## 3.3 เครื่องมือและอุปกรณ์ที่ใช้ในการวิจัย

3.3.1 ฮาร์ดแวร์

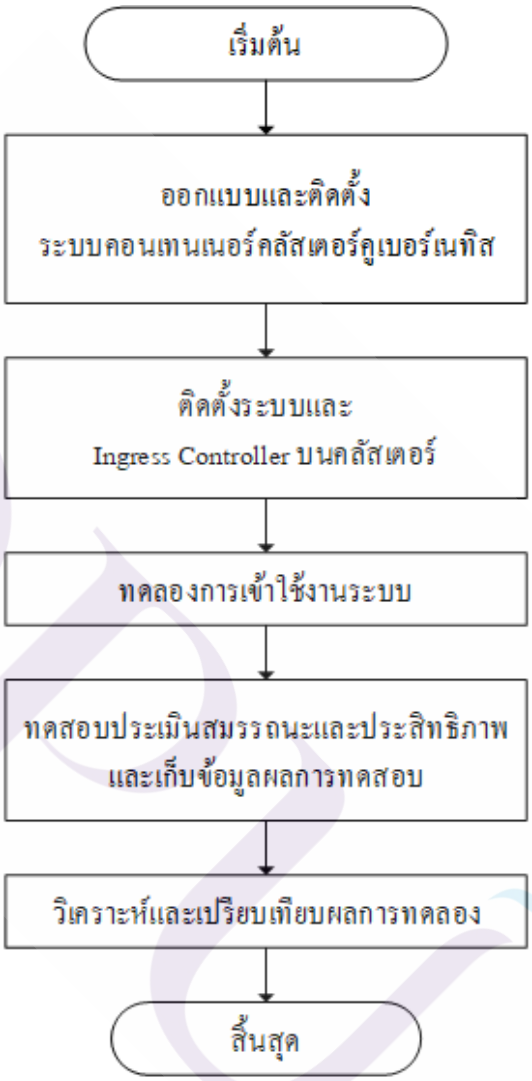
- คอมพิวเตอร์เสมือนและคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทีสบน Public Cloud Provider (Google Cloud Platform)
- คอมพิวเตอร์เวิร์คสเตชั่น (Workstation) จำนวน 1 เครื่อง เพื่อใช้ทดสอบระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทีส ที่มีส่วนประกอบดังนี้
  - CPU: 1 Socket, 4 Cores per Socket
  - Main Memory: 16 GB DDR3
  - Storage: 256 GB SSD
  - Network Adapter: 1 Gigabit Port

3.3.2 ซอฟต์แวร์

- Server Operating System: Ubuntu Linux Server
- Workstation Operating System: Ubuntu Linux Desktop รุ่น 18.04
- Container Orchestrator: Kubernetes รุ่นเสถียร รุ่น 1.10.2 (GKE)
- Container Runtime: Docker CE
- Ingress Controller:
  - Nginx Ingress Controller รุ่น 0.15.0
  - Traefik Ingress Controller รุ่น 1.6
  - Voyager Ingress Controller รุ่น 7.4.0
  - GCE L7 load balancer controller (GLBC)
- Benchmark Software: Apache JMeter และ wrk2
- Editor: Visual Studio Code รุ่น 1.25.0

## 3.4 ขั้นตอนและวิธีการดำเนินงาน

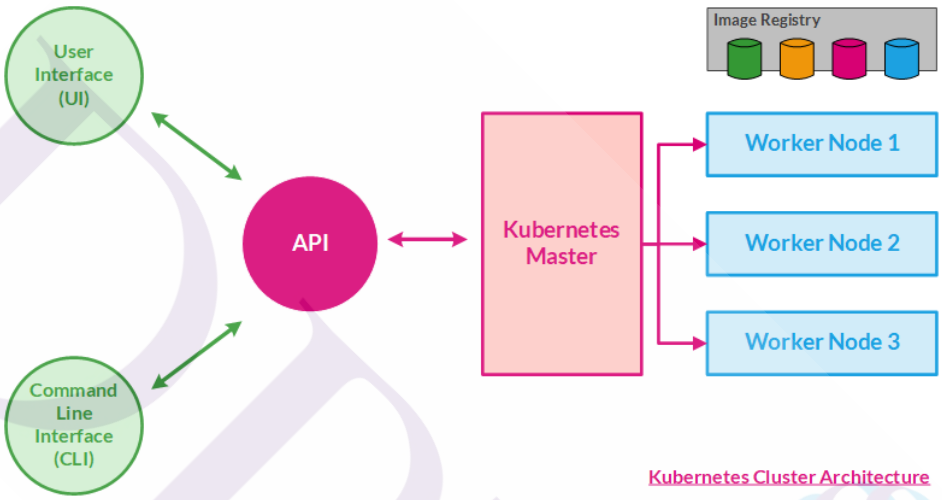
3.4.1 แผนผังขั้นตอนการดำเนินงาน




**ภาพที่ 3.1** แผนผังขั้นตอนการดำเนินงานทดสอบระบบ

3.4.2 สถาปัตยกรรมคลัสเตอร์คูเบอร์เนทิสที่ใช้ในงานวิจัย

ระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสที่นำมาใช้ในงานวิจัยนี้จะใช้รูปแบบที่อ้างอิงมาจากข้อแนะนำของ Google ที่แนะนำให้มี Master Node จำนวน 1 Node และ Worker Node จำนวน 3 Nodes มีส่วนติดต่อกับระบบคูเบอร์เนทิสผ่าน API Server ที่สามารถเข้าถึงได้จาก User Interface (UI) และ Command Line Interface (CLI) และใช้ Image Registry จาก Docker Hub และ Google Container Registry สถาปัตยกรรมที่ใช้ในงานวิจัยแสดงดังภาพที่ 3.2




**ภาพที่ 3.2** สถาปัตยกรรมคลัสเตอร์คูเบอร์เนทิสที่ใช้ในงานวิจัย

3.4.3 การออกแบบและพัฒนาระบบ

ในการทดสอบนี้จะดำเนินการสร้างระบบประมวลผลคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสบน Public Cloud ของ Google Cloud Platform (Google Kubernetes Engine) ใช้โซน asia-southeast1-a (Singapore) โดยมี Worker Node จำนวน 3 เครื่อง ที่มีทรัพยากรที่เท่า ๆ กันทุกเครื่อง โดยมีรายละเอียด แสดงดังตารางที่ 3.2

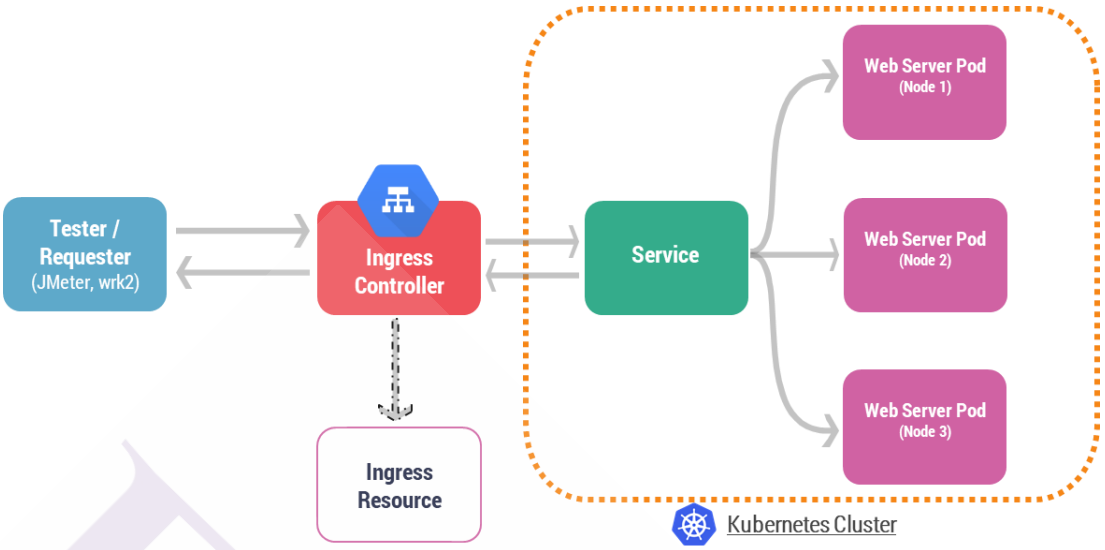
**ตารางที่ 3.2** การกำหนดทรัพยากรเครื่อง Worker Node

| **Components** | **Server** |
|---|---|
| | **Worker Node** |
| Processor | 2 vCPUs |
| Processor Platform | Intel Broadwell |
| Memory | 2.5 GB |
| Disk | 100 GB, Standard persistent disk |
| OS | Ubuntu Linux |

เมื่อทำการกำหนดทรัพยากรของเครื่องในระบบแล้ว ก็ทำการออกแบบระบบที่จะใช้ในการทดสอบ ที่จะประกอบด้วยส่วนประกอบที่สำคัญ 3 ส่วนคือ

1. คอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสและ Service (Web server) ที่ให้บริการ
2. อินเกรสคอนโทรลเลอร์ ที่ทำหน้าที่จัดการการเข้าใช้งาน Service และกระจายภาระงานไปสู่ระบบที่ทำงานอยู่เบื้องหลัง ซึ่งเป็นส่วนที่จะดำเนินการทดสอบเพื่อประเมินสมรรถนะและประสิทธิภาพในงานวิจัยนี้
3. โปรแกรมที่ใช้ในการทดสอบ (Tester/Requester) เพื่อใช้ทดสอบประสิทธิภาพของระบบ ทดสอบโดยโปรแกรม Apache JMeter และ wrk2

โดยมีรายละเอียดแสดงระบบที่ใช้ในการทดสอบ ดังภาพที่ 3.3




**ภาพที่ 3.3** ระบบที่ใช้ในการทดสอบในงานวิจัย

3.4.4 ขั้นตอนการทดสอบ

ในการดำเนินการทดสอบเพื่อประเมินสมรรถนะและประสิทธิภาพของอินเกรสคอนโทรลเลอร์บนระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิส มีขั้นตอนในการทดสอบ ดังนี้

1. สร้างระบบประมวลผลคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิส บน Google Kubernetes Engine โดยกำหนดทรัพยากรของเครื่อง Worker Node ตามรายละเอียดในหัวข้อสถาปัตยกรรมคลัสเตอร์คูเบอร์เนทิสที่ใช้ในงานวิจัยและตารางที่ 3.2

2. เมื่อทำการสร้างระบบประมวลผลคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิสพร้อมใช้งานแล้ว ขั้นตอนต่อไปก็เป็นการสร้าง pod เพื่อใช้ทำงานเป็น Webserver เพื่อตอบกลับ request ด้วยชื่อของเครื่องที่ให้บริการ (Hostname) หรือชื่อของ pod จำนวนเท่ากับจำนวน Worker Node ในการทดลองนี้ คือ 3 pods

3. สร้าง Service ในระบบคูเบอร์เนทิส เพื่อเป็นส่วนติดต่อระหว่าง External network กับ Cluster network (Internal network)

4. ติดตั้งอินเกรสคอนโทรลเลอร์ (Ingress controller) ที่จะทำการทดสอบบนคลัสเตอร์คูเบอร์เนทิส เพื่อทำหน้าที่จัดการในการเข้าใช้งาน Service บนคลัสเตอร์

5. ดำเนินการทดสอบประสิทธิภาพของอินเกรสคอนโทรลเลอร์ (Ingress controller) และบันทึกผลการทดสอบ

## 3.5 รูปแบบและวิธีการทดสอบประสิทธิภาพ

การทดสอบจะดำเนินการทดสอบเพื่อพิจารณาประสิทธิภาพของอินเกรสคอนโทรลเลอร์ในหัวข้อ ดังต่อไปนี้

### 3.5.1 การทดสอบความสามารถในการให้บริการ (Throughput)

Throughput คือความสามารถในการให้บริการของระบบต่อหน่วยเวลา ในการวัดประสิทธิภาพของ Web service จะนิยมวัดเป็น จำนวนที่สามารถให้บริการได้ต่อหน่วยเวลาเป็นวินาที (requests per second) ซึ่งการทดสอบในงานวิจัยนี้ จะวัดความสามารถในการให้บริการ (Throughput) ในหน่วย requests per second โดยทำการทดสอบการเข้าใช้บริการตั้งแต่ 100 – 5,000 requests per second ด้วยโปรแกรม wrk2 และบันทึกผลที่ระบบสามารถให้บริการได้จริง

การทดสอบด้วยโปรแกรม wrk2 จะดำเนินการทดสอบบน Linux Workstation ที่ทำงานบน VM ที่สร้างบน Google Cloud Platform ในโซน asia-southeast1-a (Singapore) โซนเดียวกับ Kubernetes Cluster ที่สร้างบน Google Kubernetes Engine (GKE) เพื่อลดระยะเวลาในการเชื่อมต่อกับ service (Latency time) เพื่อให้การทดสอบถูกรบกวนน้อยที่สุดและผลการทดสอบมีความเที่ยงตรงมากที่สุด

การใช้งานโปรแกรม wrk2 จะใช้งานผ่าน Command Line Interface (CLI) โดยมีรูปแบบคำสั่ง ดังนี้

```
$ wrk -t2 -c10 -d30s -R100 http://nginx.gotdns.ch/
```

โดยที่

wrk คือการเรียกใช้โปรแกรมที่ใช้ทำการทดสอบ

-t คือ จำนวน Thread ที่ใช้ในการทดสอบ

-c คือ จำนวน Connections ที่ใช้เชื่อมต่อในการทดสอบ

-d คือ ระยะเวลาที่ใช้ในการทดสอบ หน่วยเป็น วินาที (s) หรือ นาที (m)

-R คือ จำนวน Requests ต่อ วินาที (Constant throughput) ที่ต้องการทดสอบ

http://... คือ โฮสต์ที่ต้องการทดสอบ

ในงานวิจัยนี้ กำหนดค่าพารามิเตอร์ที่ใช้ในการทดสอบ ของโปรแกรม wrk2 ดังนี้

-t จำนวน Thread = 2 threads

-c จำนวน Connections = 10 connections

-d ระยะเวลาที่ใช้ในการทดสอบ = 30 วินาที (s)

-R จำนวน Requests ต่อ วินาที = 100, 200, 500, 1000, 1500, 2000, 2500, 3000, 4000 และ 5000 requests/second

- โฮสต์ที่ต้องการทดสอบ จำนวน 4 โฮสต์ตามจำนวนของอินเกรสคอนโทรลเลอร์

1. http://nginx.gotdns.ch/ (Nginx Ingress Controller)
2. http://traefik.gotdns.ch/ (Traefik Ingress Controller)
3. http://voyager.gotdns.ch/ (Voyager Ingress Controller)
4. http://gce.gotdns.ch/ (GLBC Ingress Controller)

เมื่อทำการทดสอบด้วยโปรแกรม wrk2 เสร็จเรียบร้อยแล้วจะได้ผลลัพธ์ ดังแสดงตามตัวอย่าง

```
wrk -t2 -c10 -d30s -R100 http://traefik.gotdns.ch/
Running 30s test @ http://traefik.gotdns.ch/
  2 threads and 10 connections
  Thread calibration: mean lat.: 2.328ms, rate sampling interval: 10ms
  Thread calibration: mean lat.: 2.344ms, rate sampling interval: 10ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.31ms    1.18ms  46.88ms   96.78%
    Req/Sec    52.57     89.34   333.00     81.03%
  3002 requests in 30.00s, 724.12KB read
Requests/sec:    100.06
Transfer/sec:     24.13KB
```

ซึ่งจะแสดงผลลัพธ์ที่ได้จากการทดสอบ ในด้านต่าง ๆ เช่น จำนวน Requests, เวลาที่ใช้ในการทดสอบ, จำนวนข้อมูลที่ส่งผ่าน, Requests/sec และ Transfer/sec และนำข้อมูลจากการทดสอบในแต่ละกรณีมาบันทึกลงในตาราง โดยมีตัวอย่างตารางบันทึกผล แสดงดังตารางที่ 3.3

**ตารางที่ 3.3** ตัวอย่างตารางบันทึกผลการทดสอบ Throughput

| **Ingress** | **Throughput (requests/second)** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **100** | **200** | **500** | **1000** | **1500** | **2000** | **2500** | **3000** | **4000** | **5000** |
| GLBC | | | | | | | | | | |
| Nginx | | | | | | | | | | |
| Traefik | | | | | | | | | | |
| Voyager | | | | | | | | | | |

3.5.2 การทดสอบความเร็วในการตอบสนอง (Response time)

Response time คือ เวลาที่ใช้ในการติดต่อระหว่างเครื่องผู้ใช้งาน (client) กับระบบที่ให้บริการ (server) จนถึงเวลาที่ได้รับข้อมูลตอบกลับ

โดยที่

**Response time = 2 * Transit Time + Processing time (at server)** **(3-1)**

และ **Transit Time** คือ เวลาที่ใช้ในการส่งข้อมูลถึงจุดหมายผ่านทางเครือข่าย

โดยทั่วไป Response time จะมีหน่วยเป็น มิลลิวินาที (ms) ในการทดสอบนี้จะทำการทดสอบการเข้าใช้บริการระบบ โดยใช้โปรแกรม Apache JMeter ทดสอบและบันทึกผลเวลาที่ระบบใช้ในการตอบสนอง ของอินเกรสคอนโทรเลอร์ 4 ตัว ในกรณีต่าง ๆ ดังนี้

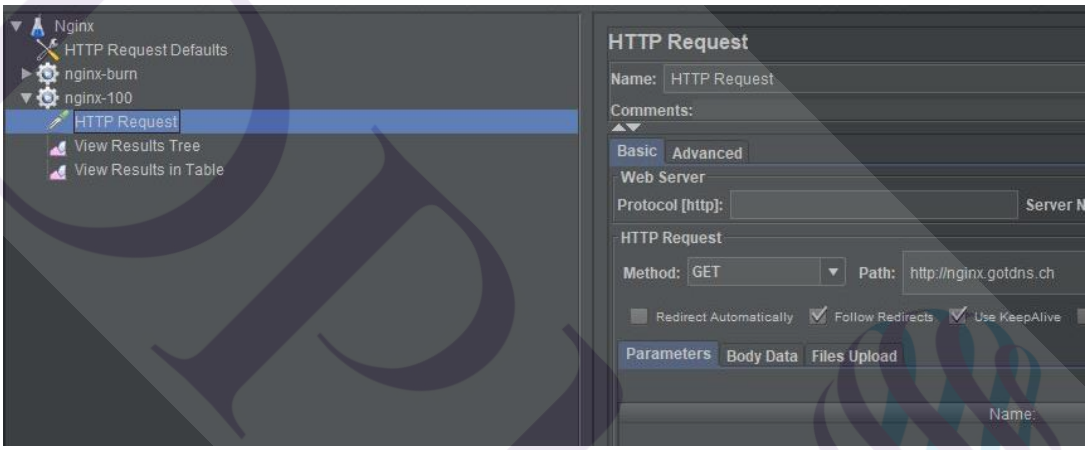
- จำนวน 100 requests ในเวลา 1 วินาที
- จำนวน 200 requests ในเวลา 1 วินาที
- จำนวน 500 requests ในเวลา 1 วินาที
- จำนวน 1,000 requests ในเวลา 1 วินาที
- จำนวน 1,500 requests ในเวลา 1 วินาที

จะได้จำนวณกรณีที่ทำการทดสอบ เท่ากับ 4 x 5 = 20 กรณี

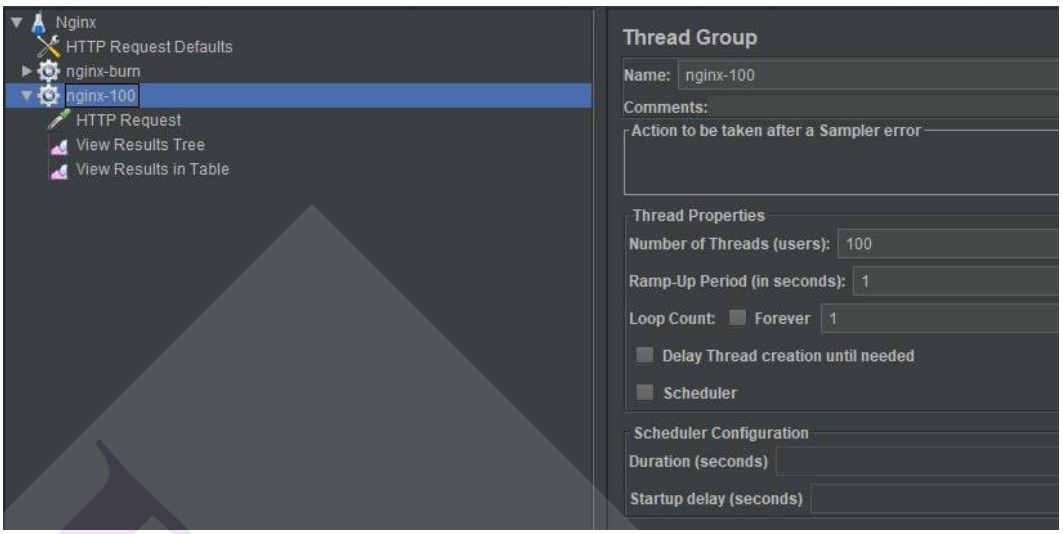
ซึ่งเหตุผลที่ใช้เวลาทดสอบ เท่ากับ 1 วินาที เพื่อให้สอดคล้องกับการทดสอบ Throughput ที่มีหน่วยเป็น requests ต่อ วินาที และเวลาที่มีจำนวนน้อย ๆ (เช่น 1 วินาที) จะสามารถวัดความสามารถในการให้บริการในกรณีที่มีการเรียกใช้บริการสูงได้ดีกว่าการใช้เวลาที่เป็นจำนวนมากขึ้น (เช่น 5 หรือ 10 วินาที)

การทดสอบด้วยโปรแกรม Apache JMeter จะดำเนินการทดสอบบน Desktop Workstation ที่เชื่อมต่อผ่านเครือข่ายจากประเทศไทย เพื่อทดสอบโดยใช้ระยะเวลาเชื่อมต่อให้เหมือนกับการเรียกใช้งานจากประเทศไทย ทำให้การทดสอบเสมือนการเรียกใช้งานเว็ปไซต์จากประเทศไทย ซึ่งใกล้เคียงกับพฤติกรรมการใช้งานมากที่สุด

การใช้งานโปรแกรม Apache JMeter สามารถเรียกใช้งานได้จาก Graphic User Interface (GUI) เพื่อกำหนดการทดสอบค่าในการทดสอบแต่ละกรณีได้อย่างสะดวก เช่น การกำหนดโฮสต์ที่ต้องการทดสอบ ดังภาพที่ 3.4 หรือ การกำหนดจำนวนและเวลาที่ต้องการทดสอบ ดังภาพที่ 3.5




**ภาพที่ 3.4** การกำหนดโฮสต์ที่ต้องการทดสอบบนโปรแกรม Apache JMeter




**ภาพที่ 3.5** การกำหนดจำนวนและเวลาที่ต้องการทดสอบบนโปรแกรม Apache JMeter

การทดสอบโดยโปรแกรม Apache JMeter จะใช้งานผ่าน Command Line Interface (CLI) โดยมีรูปแบบคำสั่ง ดังนี้

```
$ ./jmeter.sh –n –t [test-file-config.jmx]
```

โดยที่

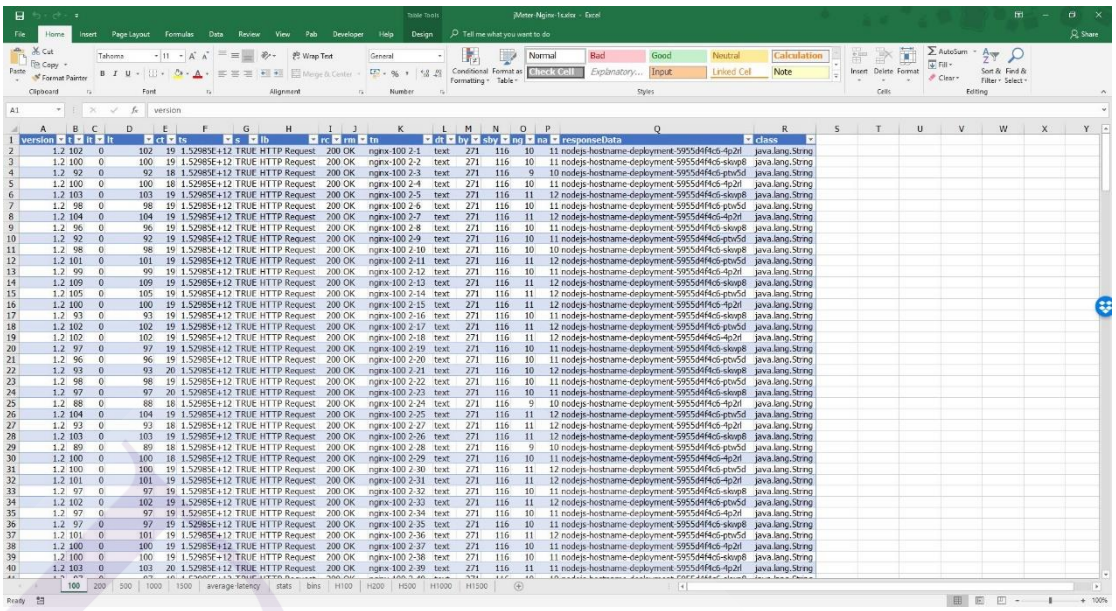
./jmeter.sh คือการเรียกใช้โปรแกรม Apache JMeter ที่ใช้ทำการทดสอบ (Linux)

-n คือ การทดสอบโดยไม่มี GUI (non-GUI mode)

-t คือ การระบุไฟล์ที่กำหนดค่าการทดสอบ (.jmx)

ตั้งค่า ผลลัพธ์ที่ได้ เป็นไฟล์ .XML (Extensible Markup Language)

และโปรแกรม Apache JMeter สามารถบันทึกและส่งออกผลการทดลองในรูปแบบ XML (Extensible Markup Language) ที่สามารถนำไปวิเคราะห์ผลการทดลองผ่านโปรแกรมภายนอกเช่น MATLAB หรือ Microsoft Excel ได้ ตัวอย่างไฟล์ XML ที่บันทึกผลการทดลอง แสดงดังภาพที่ 3.6



**ภาพที่ 3.6** ตัวอย่างผลการทดลองที่ได้จากโปรแกรม Apache JMeter

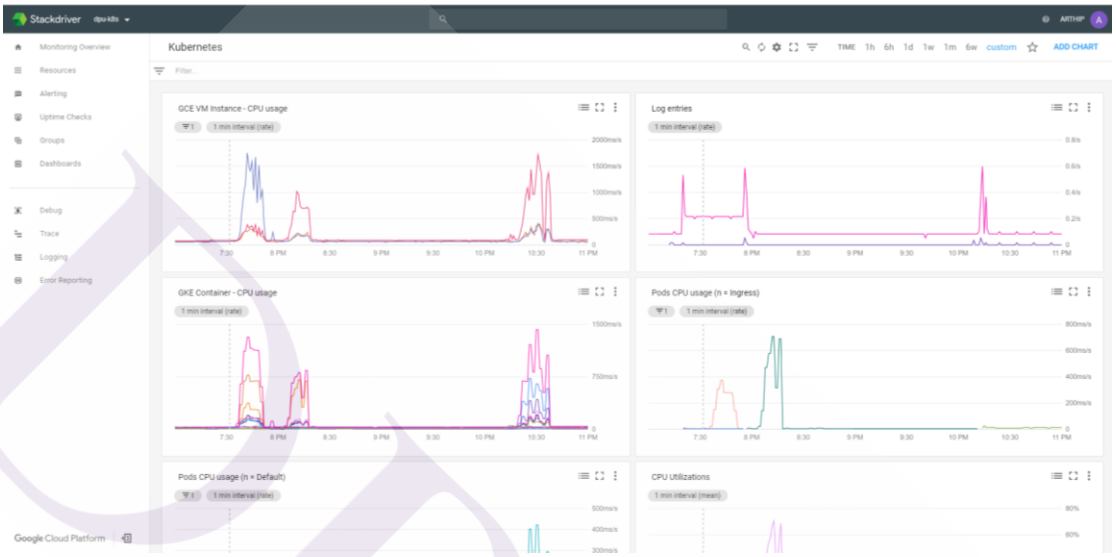
### 3.5.3 การทดสอบการใช้งานทรัพยากรของระบบ (Resources consumption)

Resources Monitoring เป็นการทดสอบและตรวจสอบเพื่อวิเคราะห์การใช้งานทรัพยากรของระบบเช่น การใช้งาน CPU, Memory หรือ Network เป็นต้น ในการทำงานของแต่ละอินเกรสคอนโทรลเลอร์จะใช้จำนวนของทรัพยากรที่ไม่เท่ากัน การใช้ทรัพยากรที่น้อยกว่าจะช่วยให้สามารถลดค่าใช้จ่ายในการลงทุนและดูแลระบบ (เช่น การลดค่าใช่จ่ายในการเช่าระบบ Public Cloud ที่มีขนาดเล็กกว่าและราคาถูกกว่า) และสามารถลดค่าใช้จ่ายในด้านพลังงานลงได้ รวมทั้งส่งผลให้ระบบมีประสิทธิภาพที่ดีพร้อมทำงานตลอดเวลา ทำให้การใช้งานทรัพยากรของระบบเป็นปัจจัยหลักที่ผู้ดูแลระบบต้องพิจารณาตรวจสอบ

ในการทดสอบนี้จะทดสอบการใช้ทรัพยากรใน 2 ด้าน เท่านั้น คือ

- การใช้งานหน่วยประมวลผลกลาง (CPU usage)
- การใช้งานหน่วยความจำ (Memory usage)

วิธีการตรวจสอบการใช้ทรัพยากรของระบบจะตรวจสอบจาก ระบบ Stackdriver[3] เป็นระบบที่จะทำการบันทึกข้อมูลการใช้ทรัพยากรจากระบบ Google Cloud Platform และสามารถนำมาเลือกแสดงผลเป็นแผนภูมิเพื่อวิเคราะห์ข้อมูลได้อย่างสะดวก ตัวอย่างแผนภูมิจากระบบ Stackdriver แสดงตามภาพที่ 3.7




**ภาพที่ 3.7** ตัวอย่างแผนภูมิแสดงข้อมูลจากระบบ Stackdriver

[3] Google Inc. *Stackdriver Monitoring Console.* Retrieved July 18, 2018, from https://app.google.stackdriver.com/

# บทที่ 4
# ผลการศึกษา

ในบทนี้จะกล่าวถึงผลจากการทดลองและการอภิปรายผลการทดลอง ในการทดสอบอินเกรสคอนโทรลเลอร์ จำนวน 4 ชนิด บนระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิส ที่มีจำนวน Worker Node เท่ากับ 3 Nodes บนระบบ Public Cloud (Google Cloud Platform) ที่ทำการทดลองในด้านต่าง ๆ คือ การทดสอบความสามารถในการให้บริการ (Throughput), การทดสอบความเร็วในการตอบสนอง (Response time) และการทดสอบการใช้งานทรัพยากรของระบบ (Resources consumption) โดยที่มีตาราง รูปภาพแสดงผลการทดลองและแผนภูมิเพื่อใช้ในการวิเคราะห์ผลการทดลองในส่วนถัดไป

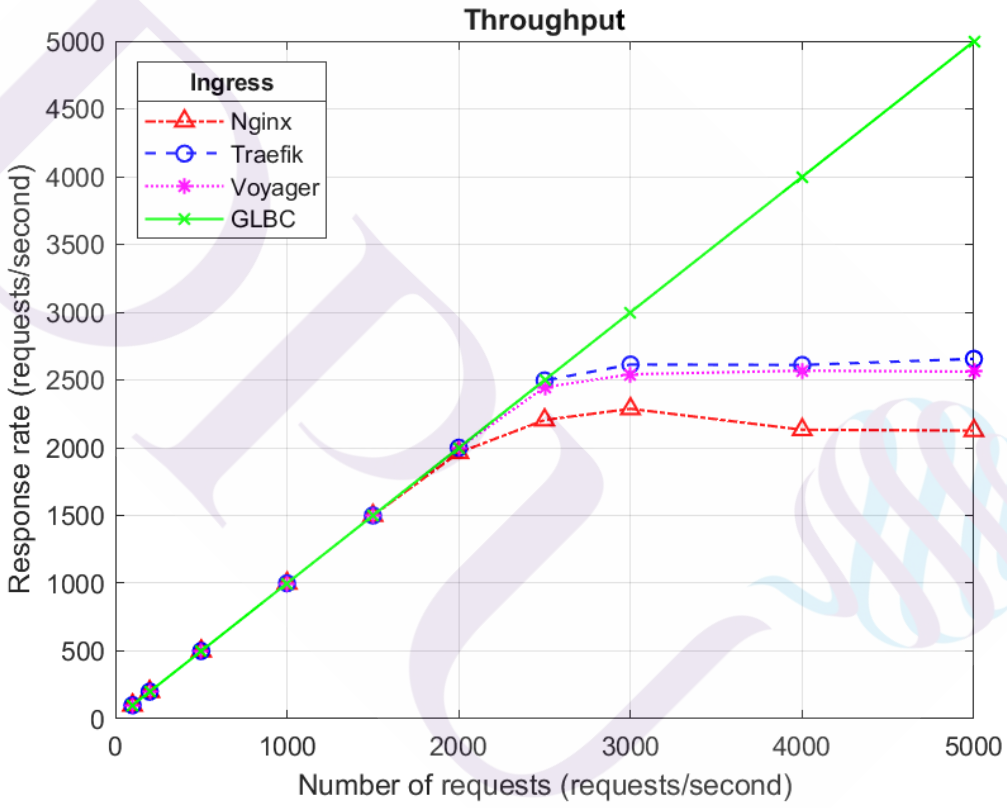
## 4.1 ความสามารถในการให้บริการ (Throughput)

ผลการทดสอบความสามารถในการให้บริการของระบบด้วยโปรแกรม wrk2 ในกรณีต่าง ๆ แสดงผลการทดสอบในตารางที่ 4.1

**ตารางที่ 4.1** ผลการทดสอบความสามารถในการให้บริการ (Throughput)

| **Ingress** | **Throughput (requests/second)** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **100** | **200** | **500** | **1000** | **1500** | **2000** | **2500** | **3000** | **4000** | **5000** |
| GLBC | 100.06 | 200.05 | 499.96 | 999.76 | 1499.54 | 1999.23 | 2499.01 | 2998.99 | 3998.42 | 4998.11 |
| Nginx | 100.06 | 200.05 | 499.95 | 999.72 | 1499.48 | 1963.55 | 2204.42 | 2286.91 | 2132.42 | 2125.80 |
| Traefik | 100.06 | 200.05 | 499.96 | 999.72 | 1499.59 | 1999.20 | 2496.41 | 2613.51 | 2610.26 | 2655.62 |
| Voyager | 100.06 | 200.05 | 499.66 | 999.69 | 1499.58 | 1997.55 | 2445.88 | 2541.61 | 2568.42 | 2560.96 |

ตารางที่ 4.1 แสดงผลการเปรียบเทียบความสามารถในการให้บริการของระบบ หรือ Throughput ผลการทดสอบพบว่า Nginx Ingress controller สามารถให้บริการได้น้อยที่สุด ที่ค่าเฉลี่ยประมาณ 2,200 requests/second ลำดับที่มีประสิทธิภาพดีขึ้นมา คือ Voyager Ingress controller ที่สามารถให้บริการได้ที่ค่าเฉลี่ย 2,560 requests/second และ Traefik Ingress controller ที่สามารถให้บริการได้ที่ค่าเฉลี่ย 2,600 requests/second ตามลำดับ โดยที่ GCE L7 load balancer controller (GLBC) สามารถให้บริการได้มากกว่า 5,000 requests/second ซึ่งสูงกว่าจำนวนที่สูงที่สุดที่ใช้ในการทดสอบ แสดงผลการเปรียบเทียบเป็นแผนภูมิดังแสดงในภาพที่ 4.1




**ภาพที่ 4.1** ผลการทดสอบความสามารถในการให้บริการ (Throughput)

## 4.2 ความเร็วในการตอบสนอง (Response time)

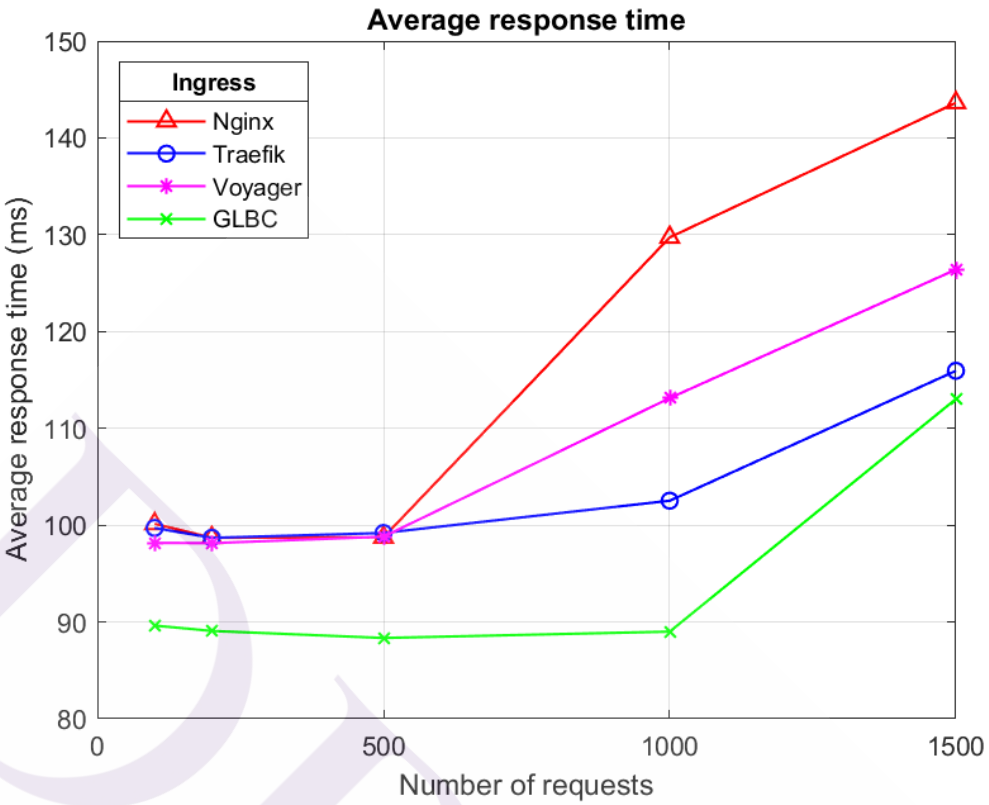
4.2.1 ความเร็วในการตอบสนองเฉลี่ย (Average response time)

ตารางที่ 4.2 แสดงค่าของความเร็วในการตอบสนองเฉลี่ย หรือ Average response time ที่ทำการทดสอบเป็นจำนวน 100, 200, 500, 1000 และ 1500 ครั้ง ผลการทดสอบพบว่า GCE L7 load balancer controller (GLBC) มีค่าความเร็วในการตอบสนองเฉลี่ยน้อยที่สุด ซึ่งค่าน้อยหมายความว่ามีประสิทธิภาพที่ดี ลำดับที่มีประสิทธิภาพที่ดีถัดมา คือ Traefik Ingress controller และ Voyager Ingress controller ตามลำดับ และ Nginx Ingress controller มีค่าความเร็วในการตอบสนองเฉลี่ยสูงที่สุด แสดงผลการเปรียบเทียบเป็นแผนภูมิดังแสดงในภาพที่ 4.2

**ตารางที่ 4.2** ผลการทดสอบความเร็วในการตอบสนองเฉลี่ย (Average response time)

| Ingress | Average response time (ms) | | | | |
|---|---|---|---|---|---|
| | **100 RPS** | **200 RPS** | **500 RPS** | **1000 RPS** | **1500 RPS** |
| GLBC | 89.63 | 89.075 | 88.346 | 89.003 | 113.03 |
| Nginx | 100.14 | 98.70 | 98.744 | 129.723 | 143.624 |
| Traefik | 99.71 | 98.69 | 99.192 | 102.523 | 115.944 |
| Voyager | 98.17 | 98.14 | 98.808 | 113.125 | 126.40 |

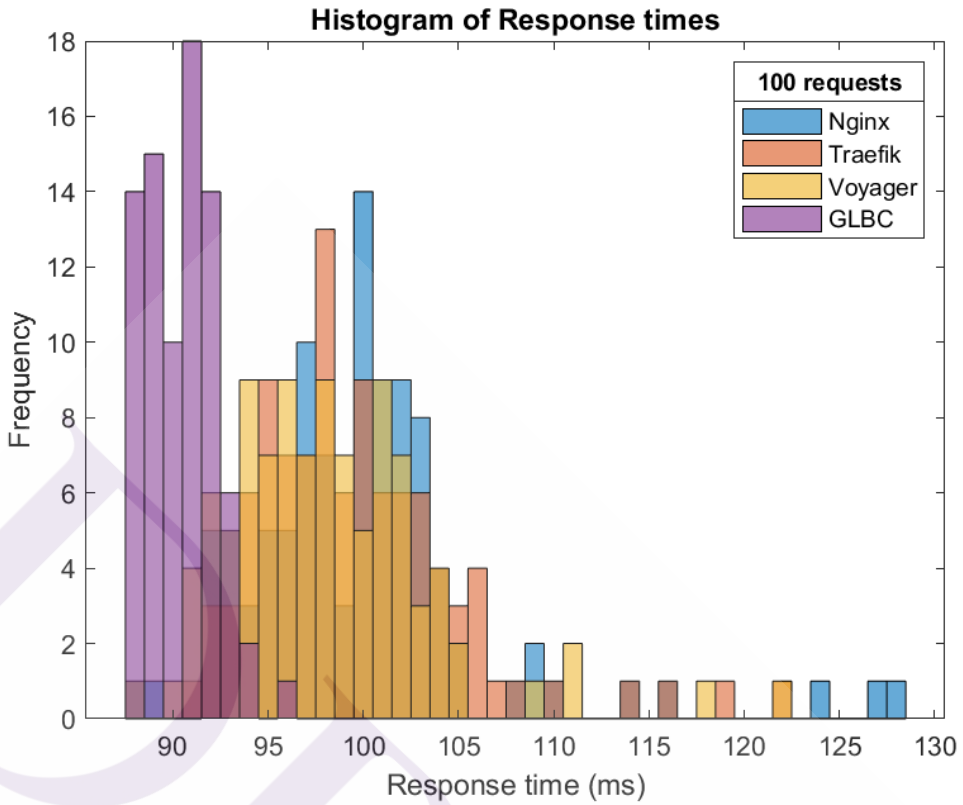
แนวโน้มจากการทดสอบพบว่าเมื่อจำนวนครั้งในการร้องขอข้อมูลเพิ่มมากขึ้นจะทำให้ความเร็วในการตอบสนองเฉลี่ยสูงขึ้นในทุก ๆ อินเกรสคอนโทรลเลอร์




**ภาพที่ 4.2** ผลการทดสอบความเร็วในการตอบสนองเฉลี่ย (Average response time)

4.2.2 การแจกแจงความถี่ของผลการทดสอบความเร็วในการตอบสนอง ที่ 100 requests/second

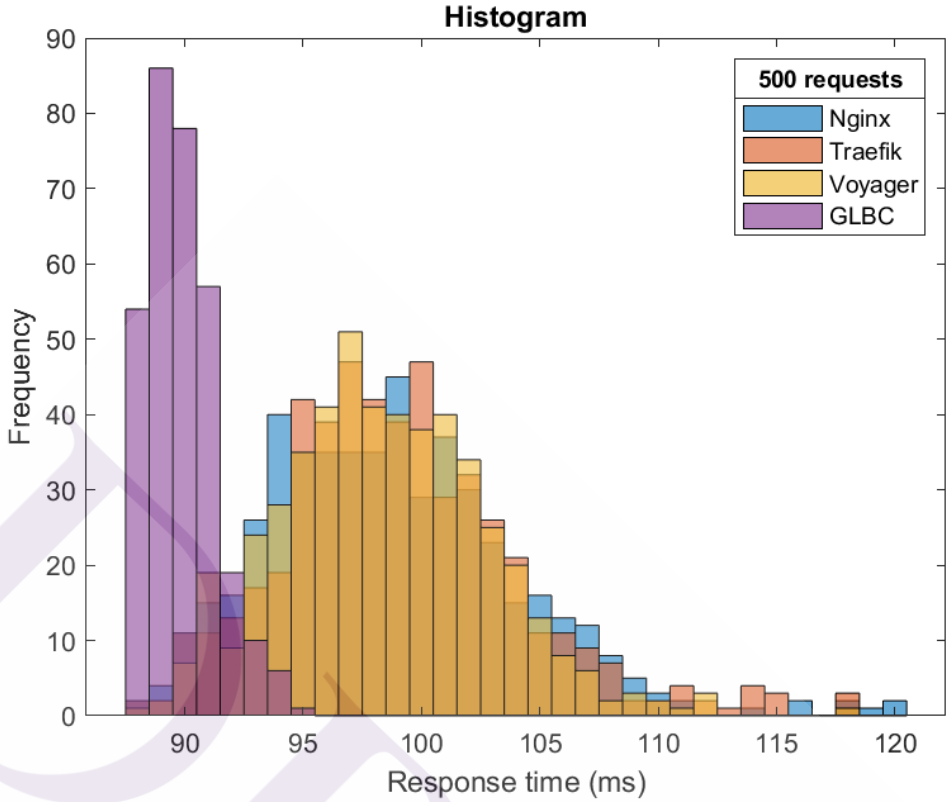
การแจกแจงความถี่ ของผลการทดสอบความเร็วในการตอบสนองที่จำนวน 000 requests/second ในทุก ๆ อินเกรสคอนโทรลเลอร์ จากรูปจะพบว่า GCE L7 load balancer controller (GLBC) มีจำนวนความถี่ที่ 10 – 18 ครั้ง อยู่ที่ค่าระหว่าง 88 – 92 มิลลิวินาที ซึ่งน้อยกว่าของอินเกรสคอนโทรลเลอร์อื่น ๆ โดยที่ Traefik Ingress controller, Voyager Ingress controller และ Nginx Ingress controller มีจำนวนความถี่ ประมาณ 9 ครั้ง อยู่ที่ค่าประมาณ 95 - 105 มิลลิวินาที จึงแสดงว่ามีค่าความเร็วในการตอบสนองที่ช้ากว่า GCE L7 load balancer controller (GLBC) (ค่า Response time น้อย แสดงถึงการมีประสิทธิภาพที่ดี) ดังแสดงในภาพที่ 4.3




**ภาพที่ 4.3** การแจกแจงความถี่ของผลการทดสอบความเร็วในการตอบสนอง ที่ 100 requests/second

4.2.3 การแจกแจงความถี่ของผลการทดสอบความเร็วในการตอบสนอง ที่ 500 requests/second

การแจกแจงความถี่ ของผลการทดสอบความเร็วในการตอบสนองที่จำนวน 500 requests/second ในทุก ๆ อินเกรสคอนโทรลเลอร์ จากรูปจะพบว่า GCE L7 load balancer controller (GLBC) มีจำนวนความถี่ที่ 70 – 90 ครั้ง อยู่ที่ค่าระหว่าง 89 – 90 มิลลิวินาที ซึ่งน้อยกว่าของอินเกรสคอนโทรลเลอร์อื่น ๆ โดยที่ Traefik Ingress controller, Voyager Ingress controller และ Nginx Ingress controller มีจำนวนความถี่ ประมาณ 40 ครั้ง อยู่ที่ค่าประมาณ 96 - 102 มิลลิวินาที จึงแสดงว่ามีค่าความเร็วในการตอบสนองที่ช้ากว้า GCE L7 load balancer controller (GLBC) ดังแสดงในภาพที่ 4.4



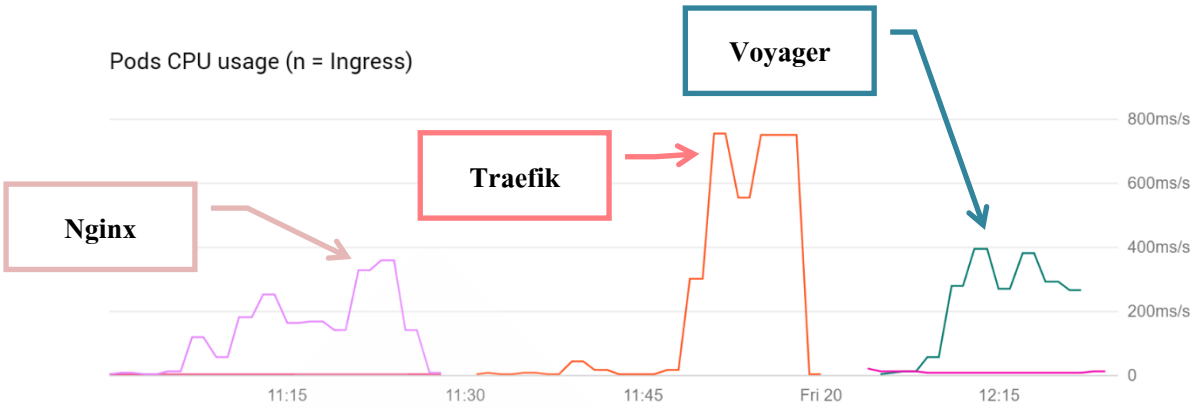

**ภาพที่ 4.4** การแจกแจงความถี่ของผลการทดสอบความเร็วในการตอบสนอง ที่ 500 requests/second

### 4.3 การใช้งานทรัพยากรของระบบ (Resources consumption)

ตรวจสอบการใช้งานทรัพยากรของระบบ จาก Log ของระบบ Google Cloud Platform ผ่านแผนภูมิของระบบ Stackdriver

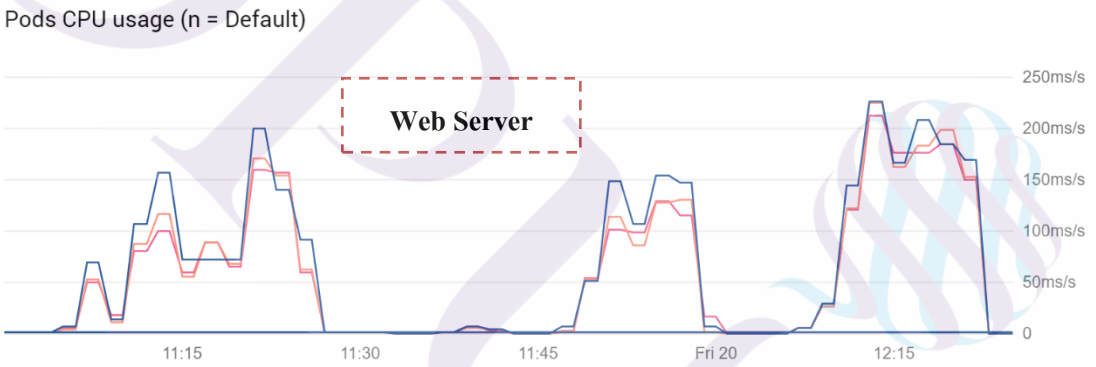
#### 4.3.1 การใช้งานหน่วยประมวลผลกลาง (CPU usage)

จากภาพที่ 4.5 แสดงการใช้ CPU ในขณะทำการทดสอบ จากรูปพบว่า Traefik Ingress Controller ใช้ CPU ในขณะทำการทดสอบมากที่สุดที่ ค่าประมาณ 750 ms/s อินเกรสคอนโทรลเลอร์ที่ใช้ CPU น้อย ลำดับถัดมา คือ Voyager Ingress Controller ที่ใช้ CPU ในขณะทำการทดสอบที่ค่าประมาณ 400 ms/s และ Nginx Ingress Controller ใช้ CPU ขณะทดสอบน้อยที่สุดที่ค่าประมาณ 320 ms/s โดยที่ GCE L7 load balancer controller (GLBC) ไม่สามารถตรวจวัดได้ เนื่องจากไม่มีการติดตั้งเพิ่มเติมบนระบบ




**ภาพที่ 4.5** การใช้ CPU ของ Namespace = Ingress

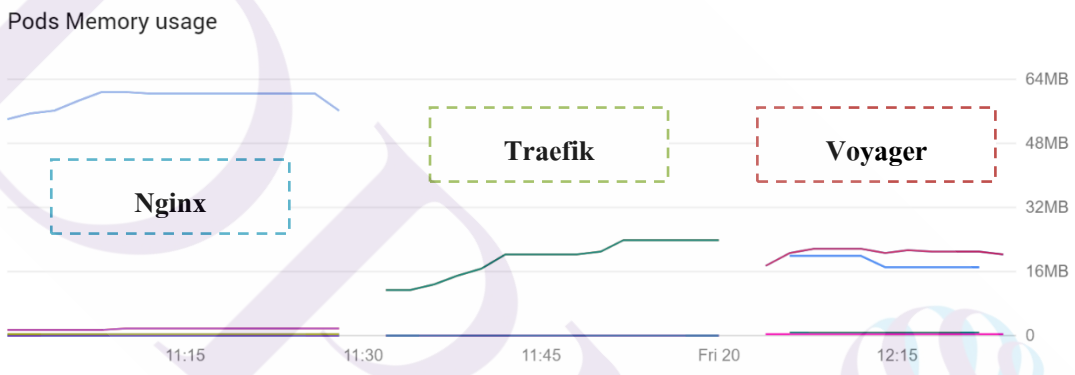
ภาพที่ 4.6 แสดงการใช้ CPU ของ Backend Pods (3 pods) ที่ทำหน้าที่ให้บริการเว็บเซอร์วิส จากรูปพบว่ามีการใช้งาน CPU ในขณะที่ทำการทดสอบใกล้เคียงกันทุก ๆ อินเกรสคอนโทรลเลอร์ ที่ค่าประมาณ 150 ms/s




**ภาพที่ 4.6** การใช้ CPU ของ Namespace = Default

4.3.2 การใช้งานหน่วยความจำ (Memory usage)

จากภาพที่ 4.7 แสดงการใช้ Memory ในขณะทำการทดสอบ พบว่า Voyager Ingress Operator ใช้ Memory ขณะทำการทดสอบที่ 16MB แต่ในการทำงานของ Voyager จะมี 2 pods ที่ทำงานพร้อม ๆ กัน คือ Voyager Ingress Operator ที่ทำหน้าที่จัดการ Ingress resources และ HAProxy ที่ทำหน้าที่กระจายภาระงาน จากรูปจะเห็นได้ว่า การใช้งาน Memory รวม ในการทำงานของ Voyager Ingress controller จะเท่ากับ 16MB บวกกับ 20MB รวมเท่ากับ 36MB ในขณะที่ Traefik Ingress Controller ใช้ Memory ขณะทำการทดสอบที่ 25MB และ Nginx Ingress Controller ใช้ Memory ขณะทำการทดสอบมากที่สุดที่ 60MB โดยที่ GCE L7 load balancer controller (GLBC) ไม่สามารถตรวจวัดได้




**ภาพที่ 4.7** การใช้ Memory ขณะทำการทดสอบ

จากข้อมูลการใช้ทรัพยากรข้างต้น สามารถนำมาสรุปเปรียบเทียบเป็นตารางได้ดังตารางที่ 4.3

**ตารางที่ 4.3** ผลการเปรียบเทียบการใช้ทรัพยากรของระบบ

| Resources / Ingress | CPU | Memory |
|---|---|---|
| Nginx | น้อย | มาก |
| Traefik | มาก | น้อย |
| Voyager | ปานกลาง | ปานกลาง |
| GLBC | ไม่สามารถตรวจวัดได้ | |

## 4.4 การทำงานของอินเกรสคอนโทรลเลอร์

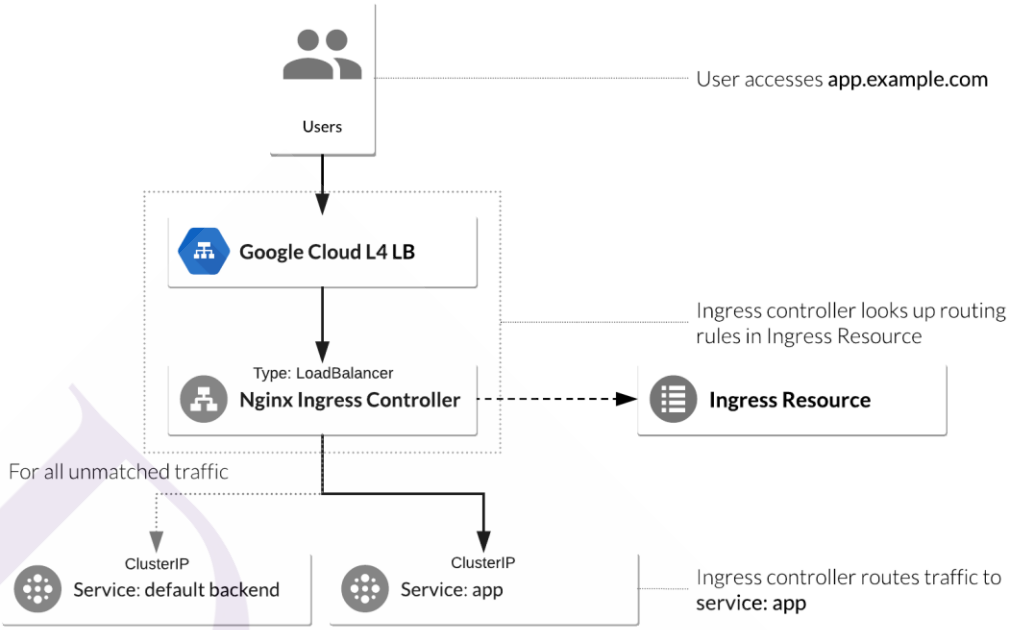
จากการติดตั้งและทดสอบการทำงานของอินเกรสคอนโทรลเลอร์ ทั้ง 4 ชนิดแล้วพบว่าอินเกรสคอนโทรลเลอร์แต่ละชนิดมีวิธีการทำงานที่แตกต่างกัน มีรายละเอียด ดังนี้

### 4.4.1 GCE L7 load balancer controller (GLBC)

ใช้คุณสมบัติของ Google Cloud HTTP(S) Load Balancing โดยตรง ซึ่งระบบ Load Balancing ของ Google Cloud สามารถทำงานได้ทั้งในระดับ Layer-4 (TCP) และ Layer-7 (HTTP(S)) ซึ่งเมื่อมีการกำหนด Ingress resource บนคลัสเตอร์คูเบอร์เนทิสแล้ว คอนโทรลเลอร์ จะสร้าง Load Balancer บนระบบ Google Cloud ที่ทำงานในระดับ Layer-7 และเชื่อมต่อกับ Backend service ทันที ข้อดีคือประสิทธิภาพของ Load Balancer สูงมากสามารถรองรับภาระงานได้เป็นจำนวนมากและตอบกลับข้อมูลได้อย่างรวดเร็ว แต่ข้อเสียคือรองรับแค่คุณสมบัติพื้นฐานในการทำงานเท่านั้น ไม่รองรับคุณสมบัติขั้นสูงบางอย่างที่ช่วยอำนวยความสะดวกในการทำงาน

### 4.4.2 Nginx Ingress Controller

การทำงานจะทำหน้าที่เป็น Reverse proxy ระดับ Layer-7 ที่กำหนดค่าอัตโนมัติผ่าน Ingress Resource ของระบบคูเบอร์เนทิส ข้อมูลจะผ่านจาก Layer-4 Load Balancer ผ่านเข้าสู่ Nginx load balancer และผ่านไปสู่ Backend service หลักการทำงานของ Nginx Ingress Controller แสดงดังภาพที่ 4.8



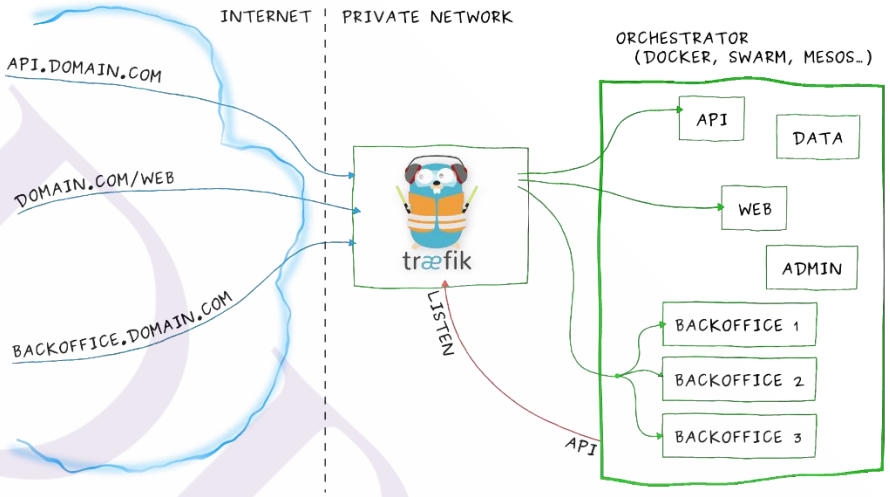

**ภาพที่ 4.8** การทำงานของ Nginx Ingress controller[1]

ข้อดีคือสามารถใช้ L4 Load Balancer เพียงตัวเดียวกับหลาย ๆ โดเมนได้ทำให้ประหยัดค่าใช้จ่ายและตัว Nginx Ingress controller ก็มีคุณสมบัติขั้นสูงที่อำนวยความสะดวกหลาย ๆ อย่าง แต่ข้อเสียก็คือ ตัวคอนโทรลเลอร์จะทำหน้าที่หนักมากซึ่งจะส่งผลให้ประสิทธิภาพไม่สูง ไม่สามารถรองรับภาระงานได้เป็นจำนวนมาก ๆ และตอบกลับข้อมูลช้าหรือไม่สามารถตอบกลับได้ในกรณีที่มีการร้องขอข้อมูลเข้ามาพร้อมกันเป็นจำนวนมาก ๆ

[1] Ameer Abbas. (2018). *Ingress with NGINX controller on Google Kubernetes Engine.* Retrieved June 15, 2018, from https://cloud.google.com/community/tutorials/nginx-ingress-gke/

4.4.3 Traefik Ingress Controller

ใช้หลักการทำงานเดียวกันกับ Nginx Ingress controller แตกต่างตรงที่ มี default-backend ในตัวเอง ทำให้ประสิทธิภาพที่ได้จะใกล้เคียงกันกับ Nginx Ingress controller บนระบบที่มีทรัพยากรเท่า ๆ กัน และรองรับคุณสมบัติขั้นสูงเพื่ออำนวยความสะดวกในการทำงานเป็นจำนวนมาก การทำงานของ Traefik Ingress controller แสดงดังภาพที่ 4.9



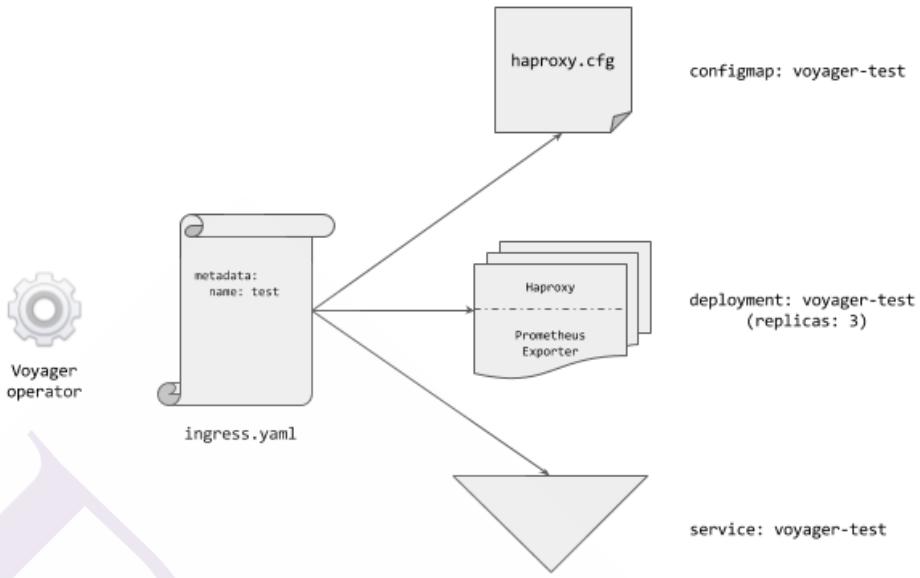

**ภาพที่ 4.9** การทำงานของ Traefik Ingress controller[2]

4.4.4 Voyager Ingress Controller

ใช้พื้นฐานการทำงานจาก HAProxy ตัวคอนโทรลเลอร์จะทำหน้าที่ตรวจดู Ingress Resource บนระบบคูเบอร์เนทิส และสั่งการในการสร้าง Reverse proxy ที่ใช้ HAProxy โดยอัตโนมัติ ทำให้สามารถลดภาระการทำงานของตัวคอนโทรลเลอร์ลงได้ ประสิทธิภาพโดยรวมสูงขึ้น และมีคุณสมบัติขั้นสูงในการอำนวยความสะดวกในการทำงานเป็นจำนวนมาก ข้อเสียคือจะมีการสร้าง Pod และ Service เพิ่มขึ้นบนระบบคูเบอร์เนทิส ซึ่งจะทำให้เสียทรัพยากรของระบบมากขึ้นหรือมีค่าใช้จ่ายในการทำงานเพิ่มมากขึ้น การทำงานของ Voyager Ingress controller แสดงดังภาพที่ 4.10

---

[2] Traefik. (2018). *Kubernetes Ingress Controller*. Retrieved June 15, 2018, from https://docs.traefik.io/




**ภาพที่ 4.10** การทำงานของ Voyager Ingress controller[3]

## 4.5 คุณสมบัติอื่น ๆ ของอินเกรสคอนโทรลเลอร์

นอกเหนือจากคุณสมบัติหลัก ๆ ของอินเกรสคอนโทรลเลอร์คือความเร็วและความมีเสถียรภาพในการทำงานแล้ว ควรพิจารณาคุณสมบัติอื่น ๆ ที่ช่วยอำนวยความสะดวกในการทำงานประกอบการเลือกใช้ ในหัวข้อนี้จะทำการเปรียบเทียบคุณสมบัติของอินเกรสคอนโทรลเลอร์ทั้ง 4 ชนิด และนำมาเปรียบเทียบเป็นตาราง แสดงดังตารางที่ 4.4

[3] AppsCode. (2018). *Voyager Secure HAProxy Ingress Controller for Kubernetes.* Retrieved June 15, 2018, from https://appscode.com/products/voyager/

**ตารางที่ 4.4** เปรียบเทียบคุณสมบัติของอินเกรสคอนโทรลเลอร์

| Features | GLBC | Nginx | Traefik | Voyager |
|---|---|---|---|---|
| Web Socket | × | √ | √ | √ |
| HTTP/2 | √ | √ | √ | √ |
| Advanced load balancing algorithms | × | √ | √ | √ |
| Web UI | × | √ | √ | √ |
| Basic Authentication | × | √ | √ | √ |
| Basic HTTPS | √ | √ | √ | √ |
| SSL Services | × | √ | √ | √ |
| Pre-install | √ | × | × | × |

# บทที่ 5
# สรุปผลและข้อเสนอแนะ

ในบทนี้จะกล่าวถึงสรุปผลที่ได้จากการทดลองของงานวิจัย อภิปรายข้อจำกัดของระบบและอุปสรรคที่พบจากการทดลอง และข้อเสนอแนะสำหรับแนวทางในการพัฒนางานวิจัย เพื่อปรับปรุงแก้ไขของงานวิจัยให้มีประสิทธิภาพและสมบูรณ์มากยิ่งขึ้น

## 5.1 สรุปผลการวิจัย

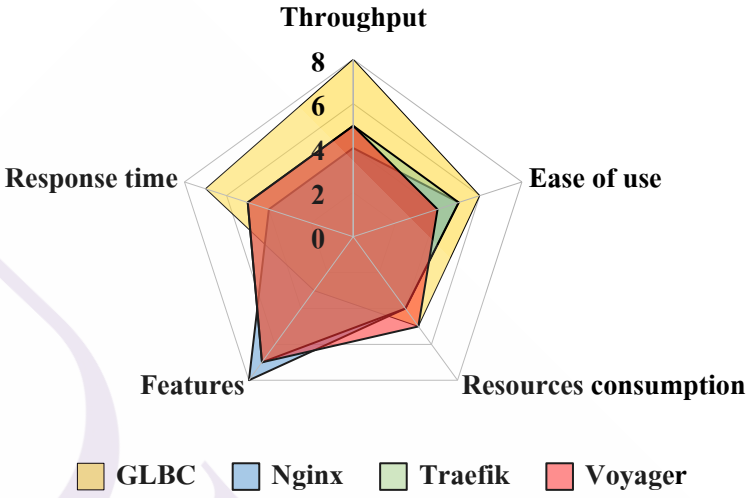
งานวิจัยนี้ทำการทดสอบประสิทธิภาพของอินเกรสคอนโทรลเลอร์บนระบบคอนเทนเนอร์คลัสเตอร์คูเบอร์เนทิส ผลการทดลองพบว่า

ในด้านความสามารถในการให้บริการ (throughput) พบว่า GCE L7 load balancer controller สามารถให้บริการได้มากกว่า 5,000 requests/second ซึ่งสูงกว่าจำนวนที่สูงที่สุดที่ใช้ในการทดสอบในงานวิจัยนี้ (ที่กำหนดไว้ที่ 5,000 requests/second) รองลงมาคือ Traefik Ingress controller ที่สามารถให้บริการได้ เฉลี่ย 2,600 requests/second, Voyager Ingress controller ที่สามารถให้บริการได้ เฉลี่ย 2,560 requests/second และ Nginx Ingress controller ที่สามารถให้บริการได้ เฉลี่ย 2,200 requests/second ตามลำดับ

ในด้านความเร็วในการตอบสนองเฉลี่ย (Average response time) พบว่า GCE L7 load balancer controller สามารถตอบสนองได้เร็วที่สุด ในทุก ๆ การทดสอบ (ทุก ๆ จำนวน requests ต่อวินาที) อินเกรสคอนโทรลเลอร์ที่ตอบสนองได้เร็วเป็นลำดับถัดมา คือ Traefik Ingress controller, Voyager Ingress controller และ Nginx Ingress controller ตามลำดับและความเร็วในการตอบสนองเฉลี่ยมีแนวโน้มสูงขึ้นเมื่อจำนวนครั้งในการร้องขอข้อมูลต่อหน่วยเวลาเพิ่มมากขึ้นในทุก ๆ อินเกรสคอนโทรลเลอร์

ในด้านการใช้งานทรัพยากรของระบบ (Resources consumption) พบว่า Nginx Ingress Controller ใช้ CPU ขณะทดสอบน้อยที่สุดแต่ใช้หน่วยความจำมากที่สุด ในขณะที่ Traefik Ingress Controller ใช้ CPU ขณะทดสอบมากที่สุดแต่ใช้หน่วยความจำน้อยที่สุด และ Voyager Ingress Controller ใช้ CPU และหน่วยความจำ ในระดับปานกลาง โดยที่ GCE L7 load balancer controller ไม่สามารถตรวจวัดได้ เนื่องจากไม่มีการติดตั้งเพิ่มเติมบนระบบ

และเมื่อนำข้อมูลทั้งหมดมาสรุปผลจะได้แผนภูมิแสดงความสามารถของอินเกรสคอนโทรลเลอร์ แสดงดังภาพที่ 5.1




**ภาพที่ 5.1** แผนภูมิสรุปความสามารถของอินเกรสคอนโทรลเลอร์ในด้านต่าง ๆ

## 5.2 ข้อจำกัดและอุปสรรคของงานวิจัย

- Google Kubernetes Engines L7 load balancer controller (GLBC) ที่ติดตั้งเป็นค่าปริยายบนระบบ Google Kubernetes Engine สามารถทำงานกับรูปแบบของ Service แบบ NodePort ได้เพียงอย่างเดียวเท่านั้น แตกต่างจาก อินเกรสคอนโทรลเลอร์ อื่น ๆ ที่สามารถใช้งานกับ Service แบบ Load Balancer ได้
- บนระบบ Google Kubernetes Engine เมื่อเลือกรูปแบบของ Service เป็น Load Balancer แล้วนั้น ระบบ Google Cloud Platform จะสร้าง Load Balancer ขึ้นให้เองและกำหนด External IP ให้เองโดยอัตโนมัติ ซึ่ง Load Balancer ที่สร้างขึ้นคนละช่วงเวลานี้ อาจจะส่งผลต่อผลการทดลองได้
- ระบบ Google Kubernetes Engine สามารถกำหนดทรัพยากรของเครื่องได้เฉพาะ Worker Node เท่านั้น เครื่อง Master Node ระบบจะจัดเตรียมไว้ให้พร้อมกับ Cluster ที่สามารถเข้าใช้งานได้เพียงอย่างเดียวเท่านั้น

- ระบบเครือข่ายที่ใช้ในการทดสอบ ความคับคั่งของเครือข่าย อาจจะส่งผลต่อผลการทดลองได้ งานวิจัยนี้จึงดำเนินการทำการทดลองในช่วงเวลาที่ใกล้เคียงกันและต่อเนื่องกันเพื่อป้องกันผลกระทบที่อาจจะเกิดขึ้นกับผลการทดลอง

- ผลการทดลองจากโปรแกรม Apache JMeter และ wrk2 ไม่สามารถนำมาวิเคราะห์เปรียบเทียบได้ในทันที ต้องนำเข้าผลการทดลองเพื่อคำนวณและวิเคราะห์ผลด้วยโปรแกรม Microsoft Excel และนำมาจัดทำแผนภูมิเปรียบเทียบด้วยโปรแกรม MATLAB

## 5.3 ข้อเสนอแนะและแนวทางการปรับปรุงในอนาคต

1.) การทดสอบนี้ดำเนินการบน Google Cloud Platform ซึ่งอาจจะมีผลการทดสอบที่คลาดเคลื่อนเมื่อนำไปทดสอบหรือใช้งานบนระบบ Public Cloud อื่น ๆ เนื่องจากประสิทธิภาพของระบบ Cloud load balancer และประสิทธิภาพของระบบเครือข่ายบนระบบ Public Cloud นั้น ๆ แนวทางการดำเนินงานวิจัยต่อไปจะดำเนินการทดลองบน Public Cloud ระบบอื่น ๆ เช่น Amazon web services, Microsoft Azure และ IBM Cloud เป็นต้น

2.) ในการนำอินเกรสคอนโทรลเลอร์ไปใช้งานควรคำนึงถึงคุณสมบัติอื่น ๆ ที่อำนวยความสะดวกในการใช้งานที่นอกเหนือจากขอบเขตของการทดสอบในงานวิจัยนี้ (Throughput และ Response time, Resources Utilization) เช่น การรองรับ Web socket, การรองรับการเข้ารหัส TLS, การทำ SSL Termination, การเลือกหรือปรับแต่ง Load balancing algorithm ขั้นสูง, ระบบ Logging และมีระบบจัดเก็บและแสดงผลข้อมูลสถิติ (Dashboard) เพื่อใช้ประกอบในการตัดสินใจเลือกใช้อินเกรสคอนโทรลเลอร์

# บรรณานุกรม

# บรรณานุกรม

## ภาษาไทย

Chanwit Kaewkasi. (2559). *คอนเทนเนอร์ คืออะไร.* สืบค้น 15 มิถุนายน 2561, จาก https://sites.google.com/site/chanwit/blogs/what-is-container/

## ภาษาต่างประเทศ

Agung B. Prasetijo, Eko D. Widianto & Ersya T. Hidayatullah. (2016). Performance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat. *Proc. of 2016 3rd Int. Conf. on Information Tech., Computer, and Electrical Engineering (ICITACEE)* (pp. 393-396). Semarang, Indonesia: IEEE.

Ameer Abbas. (2018). *Ingress with NGINX controller on Google Kubernetes Engine.* Retrieved June 15, 2018, from https://cloud.google.com/community/tutorials/nginx-ingress-gke/

Apache Software Foundation. *Apache JMeter™.* Retrieved June 15, 2018, from https://jmeter.apache.org/

AppsCode. (2018). *Voyager Secure HAProxy Ingress Controller for Kubernetes.* Retrieved June 15, 2018, from https://appscode.com/products/voyager/

DigitalOcean Inc. (2017). *What is Load Balancing?* Retrieved June 15, 2018, from https://www.digitalocean.com/community/tutorials/what-is-load-balancing/

Docker Inc. *Docker overview.* Retrieved June 15, 2018, from https://www.docker.com/what-container/

Docker Inc. *What is a Container? A standardized unit of software.* Retrieved June 15, 2018, from https://www.docker.com/what-container/

Elias Konidis, Panagiotis Kokkinos & Emmanouel Varvarigos. (2016). *Evaluating Traffic Redirection Mechanisms for High Availability Servers.* University Campus. Rion, Greece: IEEE.

Gil Tene (giltene). *A constant throughput, correct latency recording variant of wrk.* Retrieved June 15, 2018, from https://github.com/giltene/wrk2/

Google Inc. *Stackdriver Monitoring Console.* Retrieved July 18, 2018, from https://app.google.stackdriver.com/

José Emmanuel Cruz de la Cruz & Christian Augusto Romero Goyzueta. (2017). *Design of a High Availability System with HAProxy and Domain Name Service for Web Services.* Universidad Nacional del Altiplano. Puno, Perú: IEEE.

Kubernetes. (2018). *Ingress controller for Google Cloud.* Retrieved June 15, 2018, from https://github.com/kubernetes/ingress-gce/

Kubernetes. (2018). *NGINX Ingress Controller.* Retrieved June 15, 2018, from https://kubernetes.github.io/ingress-nginx/

National Institute of Standards and Technology. (2011). *The NIST Definition of Cloud Computing* (pp. 6-7). Gaithersburg, MD: National Institute of Standards and Technology.

Red Hat, Inc. *What is a Linux container?* Retrieved June 15, 2018, from https://www.redhat.com/en/topics/containers/whats-a-linux-container/

Red Hat, Inc. *What is Kubernetes?* Retrieved June 15, 2018, from https://www.redhat.com/en/topics/containers/what-is-kubernetes/

Tahira Islam & Mohammad S. Hasan. (2017). *A Performance Comparison of Load Balancing Algorithms for Cloud Computing.* University of Dhaka. Dhaka, Bangladesh: IEEE.

The Linux Foundation. *Concepts Underlying the Cloud Controller Manager.* Retrieved June 15, 2018, from https://kubernetes.io/docs/concepts/architecture/cloud-controller/

The Linux Foundation. *Kubernetes: Production-Grade Container Orchestration.* Retrieved June 15, 2018, from https://kubernetes.io/

The Linux Foundation. *Using a Service to Expose Your App.* Retrieved June 15, 2018, from https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/

The Linux Foundation. *Using kubectl to Create a Deployment.* Retrieved June 15, 2018, from https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

Traefik. (2018). *Kubernetes Ingress Controller.* Retrieved June 15, 2018, from https://docs.traefik.io/

## ประวัติผู้เขียน

| | |
|---|---|
| ชื่อ-นามสกุล | นายอาธิป พวงลำใย |
| ประวัติการศึกษา | ปริญญาตรี |
| | วศ.บ. วิศวกรรมทรัพยากรน้ำ |
| | มหาวิทยาลัยเกษตรศาสตร์ |
| | พ.ศ.2551 |
| ตำแหน่งและสถานที่ทำงานปัจจุบัน | วิศวกร ระดับ 5 |
| | ฝ่ายระบบส่งน้ำดิบ การประปานครหลวง |