สถาปัตยกรรมระบบพลังงานอัจฉริยะแบบคลาวด์ ด้วยการประมวลผลเชิงกระจาย ในคลัสเตอร์ที่ร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์

นายกิตติพัฒน์ แสงแก่นเพ็ชร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Cloud-Based Smart Energy System Architecture with Distributed Computing in Federated Cluster of Container Orchestration

Mr. Kittipat Saengkaenpetch

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering
Department of Electrical Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2021

หัวข้อวิทยานิพนธ์	สถาปัตยกรรมระบบพลังงานอัจฉริยะแบบคลาวด์ ด้วยการประมวลผลเชิงกระจายในคลัสเตอร์ที่ร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์
โดย	นายกิตติพัฒน์ แสงแก่นเพ็ชร์
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.เชาวน์ดิศ อัศวกุล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.ชัยเชษฐ์ สายวิจิตร)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.เชาวน์ดิศ อัศวกุล)

..... กรรมการภายนอกมหาวิทยาลัย
(รองศาสตราจารย์ ดร.มณิศา พิพัฒนสมพร)

..... กรรมการภายนอกมหาวิทยาลัย
(ดร.สุรัตน์ ตันเทอดทิตย์)

กิตติพัฒน์ แสงแก่นเพ็ชร์ : สถาปัตยกรรมระบบพลังงานอัจฉริยะแบบคลาวด์ ด้วยการประมวลผลเชิงกระจายในคลัสเตอร์ที่ร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์. (Cloud-Based Smart Energy System Architecture with Distributed Computingin Federated Cluster of Container Orchestration) อ.ที่ปรึกษาหลัก : รศ. ดร.เชาวน์ดิศ อัศวกุล

วิทยานิพนธ์นี้นำเสนอสถาปัตยกรรมระบบพลังงานอัจฉริยะแบบคลาวด์ ด้วยการประมวลผลเชิงกระจายและใช้เทคโนโลยีการร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์ เทคโนโลยีคอนเทนเนอร์จะช่วยในการย้ายแอพลิเคชันและการปรับปรุงรุ่นแอพลิเคชันได้สะดวก โดยทั่วไปข้อมูลจากอุปกรณ์ไอโอทีต่าง ๆ จะรวมศูนย์การจัดเก็บข้อมูล และประมวลผลอยู่ที่ส่วนกลาง งานวิจัยนี้ได้นำสถาปัตยกรรมระบบการประมวลผลเชิงกระจาย ที่สามารถเร่งความเร็วการประมวลผลให้เร็วกว่าแบบรวมศูนย์ นอกจากนี้วิทยานิพนธ์นี้เสนอสถาปัตยกรรมระบบอื่น ๆ เพื่อเป็นประโยชน์เชิงการศึกษาความเป็นไปได้ของการประยุกต์ในทางปฏิบัติสำหรับการใช้เทคโนโลยีคอนเทนเนอร์ คิวเบอร์เนเทส และการร่วมสมาพันธ์คิวเบอร์เนเทส โดยได้นำเสนอการทดสอบสถาปัตยกรรมแบบต่าง ๆ และใช้ระบบทดสอบในโครงการความร่วมมือในการพัฒนาแพลทฟอร์มรองรับศูนย์กลางข้อมูลไอโอทีคลาวด์ หรือ IoTcloudServe@TEIN นอกจากนี้ได้ต่อยอดสถาปัตยกรรมระบบในโครงการ OF@TEIN++ ในการทดสอบการร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์ระหว่างคลัสเตอร์ในความร่วมมือระหว่างมหาวิทยาลัยได้แก่ จุฬาลงกรณ์มหาวิทยาลัย ประเทศไทย มหาวิทยาลัยมาลายา ประเทศมาเลเซีย และ สถาบันวิทยาศาสตร์และเทคโนโลยีกวางจู ประเทศเกาหลีใต้ จากนั้นได้ต่อยอดสถาปัตยกรรมระบบต่อเนื่องในโครงการ OF@TEIN+++ เพื่อนำเสนอการจัดลำดับชั้นของการแบ่งทรัพยากรในการแยกกลุ่มภาระของคลัสเตอร์ที่ร่วมสมาพันธ์ และท้ายสุดได้นำเสนอแนวทางประยุกต์สถาปัตยกรรมในบริบทของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

ปีการศึกษา 2564 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

6270016021 : MAJOR ELECTRICAL ENGINEERING

KEYWORD: Smart Energy Framework, Data Analytics, Kubernetes

Kittipat Saengkaenpetch : Cloud-Based Smart Energy System Architecture with Distributed Computingin Federated Cluster of Container Orchestration. Advisor: Assoc. Prof. CHAODIT ASWAKUL, Ph.D.

This thesis proposes cloud-based smart energy system architectures with distributed computing in federated cluster of container orchestration. Container technology improves application portability and application revisioning conveniently. In general, the data gathered from IoT devices are stored and computed inside one central database. This research proposes a distributed computing system architecture which accelerates data analytic faster than the architecture relying on a central database and analytics. This thesis also proposes related system architectures for educational feasibility study purpose of container technology, Kubernetes and Kubernetes cluster federation. This research's feasibility study is demonstrated practically by the actual tests of different architectures in network testbed as supported by the data-centric IoT-cloud service platform for smart communities (IoTcloudServe@TEIN) project. Furthermore, this thesis proposes system architecture for cluster federation of container orchestration in OF@TEIN++ project. These collaborations are from Chulalongkorn University from Thailand, University of Malaya from Malaysia, and Gwangju Institute of Science and Technology from South Korea. Lastly, this thesis proposes system architecture for hierarchical resource group in OF@TEIN+++ project to demonstrate resource isolation among federated clusters. The application of these architectures has been discussed finally in the context of Electricity Generating Authority of Thailand (EGAT).

Field of Study:	Electrical Engineering	Student's Signature .....
Academic Year:	2021	Advisor's Signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงได้เพราะได้รับความอนุเคราะห์จาก รศ. ดร.เชาวน์ดิศ อัศวกุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ได้ให้คำปรึกษา แนวคิดในการทำวิทยานิพนธ์ การเขียนบทความทางวิชาการ และโอกาสในการร่วมโครงการพัฒนาแนวทางการใช้คลาวด์ต่าง ๆ วิทยานิพนธ์นี้ได้รับการสนับสนุนจากโครงการความร่วมมือในการพัฒนาแพลทฟอร์มรองรับศูนย์กลางข้อมูลไอโอทีคลาวด์ หรือ IoTcloudServe@TEIN โครงการ OF@TEIN++ และโครงการ OF@TEIN+++ ขอขอบคุณหน่วยงานการไฟฟ้าฝ่ายผลิตแห่งประเทศไทยที่ได้ให้การสนับสนุนทุนการศึกษาระดับปริญญาโทของผู้เขียน ขอขอบคุณสมาชิกผู้ร่วมโครงการในการออกแบบ ติดตั้ง และทดสอบสถาปัตยกรรมระบบ ตลอดจนการนำไปประยุกต์ใช้ในกรณีศึกษาต่าง ๆ ได้

กิตติพัฒน์ แสงแก่นเพ็ชร์

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญ

เทคโนโลยีข้อมูลขนาดใหญ่ (big data) มีส่วนสำคัญในภาคพลังงาน ทั้งการจัดการพลังงานอัจฉริยะ (intelligent energy management) การทำนายการบริโภคพลังงาน (energy consumption prediction) และการดึงประโยชน์จากอินเทอร์เน็ตสรรพสิ่ง (Internet of Things exploitation - IoT) [1] อุปกรณ์ไอโอที คือ เซ็นเซอร์ (sensor) หรืออุปกรณ์ส่งสัญญาณที่สามารถเชื่อมต่ออินเทอร์เน็ตเพื่อส่งข้อมูลได้ ข้อมูลที่ถูกบันทึกจะมีความหลากหลาย (variety) มีความต่อเนื่อง (velocity) และมีปริมาณมาก (volume) คุณลักษณะดังกล่าวเป็นลักษณะของข้อมูลขนาดใหญ่ [2] จึงควรเลือกใช้สถาปัตยกรรมระบบ เทคโนโลยีและโครงสร้างพื้นฐาน (infrastructure) ที่รองรับคุณลักษณะดังนี้ได้ วิทยานิพนธ์นี้จะนำเสนอสถาปัตยกรรมระบบและติดตั้งพิสูจน์การทำงานจริง ได้แก่ สถาปัตยกรรมระบบกรอบการทำงานพลังงานอัจฉริยะ (smart energy framework) ในการวิเคราะห์ข้อมูลจากอุปกรณ์ไอโอทีที่สามารถเพิ่มความเร็วการวิเคราะห์ข้อมูลเมื่อมีทรัพยากรมากขึ้นได้ โดยใช้เทคโนโลยีคอนเทนเนอร์ช่วยในการเคลื่อนย้ายแอพลิเคชันการวิเคราะห์ข้อมูลได้สะดวกยิ่งขึ้น และเทคโนโลยีออร์เคสเตรเตอร์ที่ช่วยบริหารทรัพยากรการประมวลผลของเครื่องเซิร์ฟเวอร์หลายโนดเป็นกลุ่มได้ เพื่อให้เข้าใจลักษณะของปัญหาที่สนใจในวิทยานิพนธ์นี้ หัวข้อ 1.1.1 - 1.1.6 จะนำเสนอองค์ความรู้พื้นฐานสำคัญ ได้แก่ หัวข้อ 1.1.1 นำเสนอความต้องการของระบบ ซึ่งเป็นการนำเสนอทรัพยากรระบบที่ภาระ (workload) ต้องการในการทำงาน หัวข้อ 1.1.2 นำเสนอเวอร์ชัวไลเซชัน (virtualization) ซึ่งเป็นเทคโนโลยีเดิมในการจัดสรรทรัพยากรระบบแบบรวมศูนย์ (pooling) ด้วยการปันทรัพยากรและจำลองทรัพยากรระบบให้กับภาระ หัวข้อ 1.1.3 นำเสนอคอนเทนเนอร์ (container) ซึ่งเป็นเทคโนโลยีการจัดการทรัพยากรระบบที่มีประสิทธิภาพยิ่งขึ้น เนื่องจากไม่มีการจำลองระบบปฏิบัติการแต่ใช้ระบบปฏิบัติการร่วมกัน หัวข้อ 1.1.4 นำเสนอคอนเทนเนอร์ออร์เคสเตรชัน (container orchestration) ซึ่งเป็นเทคโนโลยีการรวมศูนย์การจัดการทรัพยากรระบบแบบคอนเทนเนอร์ของหลายโนดเข้าด้วยกันเป็นคลัสเตอร์ (cluster) หัวข้อ 1.1.5 นำเสนอคลัสเตอร์ร่วมสมาพันธ์ (cluster federation) ซึ่งเป็นเทคโนโลยีการจัดสรรทรัพยากรระบบระหว่างคลัสเตอร์ และหัวข้อ 1.1.6 นำเสนอการเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่ ซึ่งเป็นการประยุกต์เทคโนโลยีการจัดการทรัพยากรระบบในการวิเคราะห์ข้อมูลขนาดใหญ่รองรับเทคโนโลยีพลังงานอัจฉริยะ เทคโนโลยีต่าง ๆ ที่นำเสนอโดยลำดับจะนำไปสู่การออกแบบสถาปัตยกรรมระบบ วิธีการติดตั้ง และทดสอบคุณสมบัติ ในหัวข้อ 1.2 ซึ่งจะนำเสนองานวิจัยที่เกี่ยวข้องในการออกแบบสถาปัตยกรรมระบบพลังงานอัจฉริยะ ประกอบกับงานวิจัยการจัดการคลัสเตอร์ที่ร่วมสมาพันธ์

เนื่องจากเป็นกลไกการจัดการทรัพยากรระบบที่สนใจ โดยเฉพาะเมื่อมีภาระงานทำงานอยู่ในหลายคลัสเตอร์และมีผู้ดูแลระบบหลายกลุ่ม โดยได้สรุปประเด็นปัญหา วัตถุประสงค์ ขอบเขต ขั้นตอนการศึกษา และประโยชน์ที่คาดว่าจะได้รับตามลำดับ

1.1.1 ความต้องการของระบบ (system requirement)

แอพลิเคชัน (application) หนึ่ง ๆ มีความต้องการของระบบแตกต่างกัน เช่น แอพลิเคชันไมโครซอฟต์ออฟฟิต (Microsoft Office) มีความต้องการทรัพยากรขั้นต่ำดังนี้ [3]

หน่วยประมวลผลกลาง (cpu)	1.6 GHz processor
หน่วยความจำ (memory)	4 GB
หน่วยเก็บข้อมูล (hard disk)	4 GB
สถาปัตยกรรม (architecture)	x86
ระบบปฏิบัติการ (operating system)	Windows

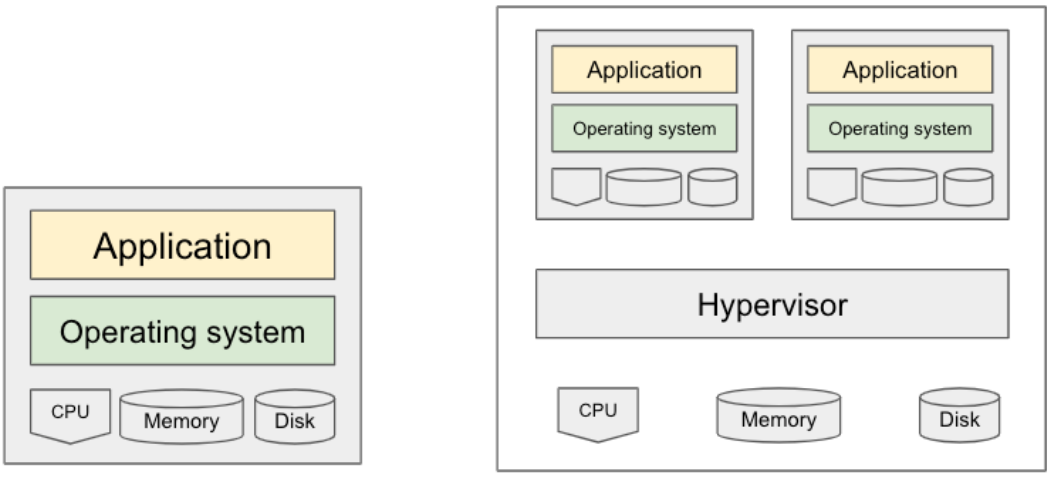
ความต้องการพื้นฐานของแอพลิเคชันนี้ ประกอบด้วย หน่วยประมวลผลกลาง หน่วยความจำ หน่วยเก็บข้อมูล สถาปัตยกรรม ระบบปฏิบัติการ และอื่น ๆ เช่น อินเตอร์เน็ต (internet) เป็นต้น อีกแอพลิเคชันหนึ่งที่มีลักษณะเป็นผู้ให้บริการ จำเป็นต้องติดตั้งอยู่ในสภาพแวดล้อมที่มีทรัพยากรเพียงพอเช่นกัน เช่น แอพลิเคชันฐานข้อมูล โปสเกรสเอสคิวแอล (PostgresSQL) มีความต้องการของระบบขั้นต่ำ [4] ดังนี้

หน่วยประมวลผลกลาง	1 GHz processor
หน่วยความจำ	2 GB
หน่วยเก็บข้อมูล	512 MB
สถาปัตยกรรม	x86
ระบบปฏิบัติการ	Windows

การติดตั้งแอพลิเคชันในลักษณะเป็นผู้ให้บริการนี้ จะติดตั้งในเครื่องที่มีประสิทธิภาพสูงกว่าเครื่องคอมพิวเตอร์ใช้งานทั่วไป เรียกเครื่องคอมพิวเตอร์ลักษณะนี้ว่า “เซิร์ฟเวอร์” (server) เครื่องเซิร์ฟเวอร์จะถูกเปิดใช้งานทิ้งไว้อย่างต่อเนื่อง รองรับผู้ใช้งานที่อาจเรียกใช้ได้ตลอดเวลา จึงเกิดบริการดูแลรักษาเซิร์ฟเวอร์ ให้มีสภาพปกติ มีระบบรักษาความปลอดภัย อุปกรณ์สำรองทางไฟฟ้า และอื่น ๆ เรียกบริการลักษณะนี้ว่า โคโลเคชัน (colocation) คือการนำฝากเซิร์ฟเวอร์ไว้ในสถานที่เดียวกัน

1.1.2 เวอร์ชัวไลเซชัน (virtualization)

เมื่อราคาของเซิร์ฟเวอร์ถูกลง ประสิทธิภาพของเซิร์ฟเวอร์สูงขึ้น สามารถรวมหลายระบบงานให้สามารถทำงานได้บนเซิร์ฟเวอร์เครื่องเดียว จึงเกิดเป็นเทคโนโลยีเวอร์ชัวไลเซชัน คือการจำลองเซิร์ฟเวอร์ทั้งเครื่อง ให้อยู่ในเซิร์ฟเวอร์ขนาดใหญ่ เรียกเซิร์ฟเวอร์ที่ถูกจำลองอยู่ในเซิร์ฟเวอร์ ขนาดใหญ่นี้ว่า เวอร์ชัวแมชชีน (virtual machine) หรือ วีเอ็ม (VM) และเรียกบริการนี้ว่า บริการโครงสร้างพื้นฐาน (infrastructure as a service) [5]

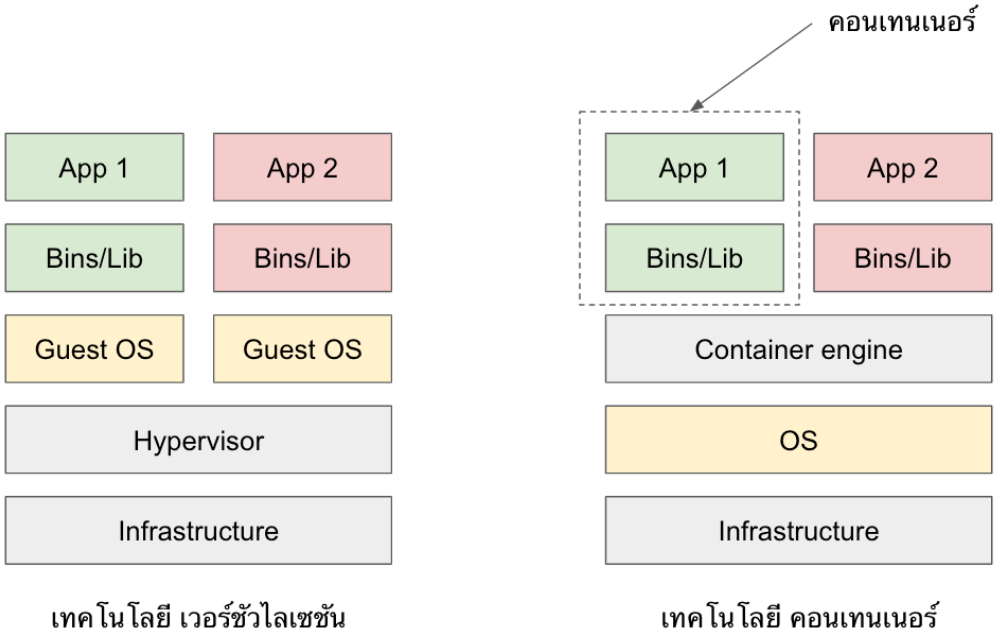


รูปที่ 1 เปรียบเทียบเซิร์ฟเวอร์ทั่วไป (ซ้าย) กับการจำลองเซิร์ฟเวอร์หลายเครื่องในเครื่องเดียว (ขวา)

เซิร์ฟเวอร์ที่มีเทคโนโลยีเวอร์ชัวไลเซชัน จะต้องมีไฮเปอร์ไวเซอร์ (hypervisor) ที่ช่วยแปลงทรัพยากรจำลองไปสู่ทรัพยากรจริง ซึ่งในตลาดมีไฮเปอร์ไวเซอร์หลายชนิดให้เลือกใช้ เช่น วีเอ็มแวร์ วีสเฟียร์ (VMware vSphere) ไมโครซอฟต์ไฮเปอร์วี (Microsoft Hyper-V) เควีเอ็ม (KVM) [6] เป็นต้น เทคโนโลยีเวอร์ชัวไลเซชันประสบความสำเร็จในการย้ายระบบงานทั้งเซิร์ฟเวอร์มาจำลองภายใต้สภาพแวดล้อมได้สมบูรณ์ นอกจากนี้เทคโนโลยีเวอร์ชัวไลเซชัน สามารถรวบรวมทรัพยากรจากหลายตัวเข้ามาเป็นผืนเดียวกัน เป็น “คลัสเตอร์” (cluster) จึงเกิดเทคโนโลยี การย้ายวีเอ็มระหว่างเซิร์ฟเวอร์ไปมาในคลัสเตอร์เดียวกัน ด้วยเหตุผลของการบำรุงรักษาที่ต้องการปิดเซิร์ฟเวอร์บางเครื่องเพื่อซ่อมบำรุง ดังนั้นเทคโนโลยีเวอร์ชัวไลเซชันจะสามารถรักษาความพร้อมใช้ (availability) การให้บริการของระบบได้ดีกว่าเซิร์ฟเวอร์ปกติทั่วไปได้

1.1.3 คอนเทนเนอร์

เทคโนโลยีเวอร์ชัวไลเซชันจะจำลองระบบปฏิบัติการของแต่ละวีเอ็มที่คล้ายกัน ทำให้มีการสูญเสียทรัพยากรไปกับการจำลองระบบปฏิบัติการที่ซ้ำซ้อน [7] เทคโนโลยีคอนเทนเนอร์ (container) จะทำให้การใช้ทรัพยากรมีประสิทธิภาพยิ่งขึ้น โดยการเพิ่มส่วนของคอนเทนเนอร์เอนจิน (container engine) หรือส่วนใหญ่รู้จักในชื่อ “ดอกเกอร์เอนจิน” (docker engine) ที่ช่วยประสานการเรียกใช้ระบบปฏิบัติการเดียวกันเข้าด้วยกัน [7]



รูปที่ 2 ความแตกต่างระหว่างเทคโนโลยีเวอร์ชัวไลเซชันและคอนเทนเนอร์

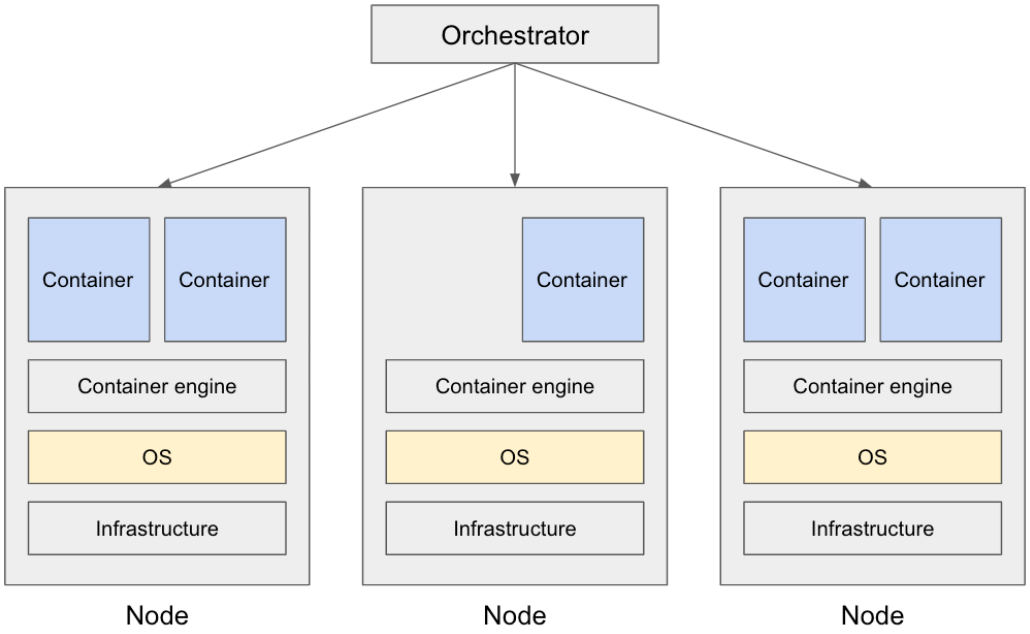
จากรูปที่ 2 ด้านขวาจะเห็นว่า ระบบปฏิบัติการของผู้ใช้งาน (Guest OS) หายไป เหลือแต่แอพลิเคชัน และไลบรารี (library) ในการประมวลผลที่จำเป็น แอพลิเคชันที่จะทำงานในลักษณะคอนเทนเนอร์นี้จำเป็นจะต้องสามารถถูกห่อแอพลิเคชันด้วยกระบวนการคอนเทนเนอร์ไรเซชัน (containerization) ได้

คอนเทนเนอร์มีข้อดีในการเคลื่อนย้ายได้ง่าย สามารถทำงานในสภาพแวดล้อมใดก็ได้ ที่มีคอนเทนเนอร์เอนจิน ระบบปฏิบัติการ และสถาปัตยกรรมเดียวกัน ซึ่งมีความหลากหลายดังนี้ ตัวเลือกของคอนเทนเนอร์เอนจิน ได้แก่ ดอกเกอร์เอนจิน (docker engine) คอนเทนเนอร์ดี (containerd) วินโดว์คอนเทนเนอร์ เป็นต้น ระบบปฏิบัติการ เช่น ลินุกซ์ (linux) หรือ วินโดวส์ เป็นต้น และ สถาปัตยกรรม เช่น เอกซ์86 (x86) หรือ อาร์ม64 (arm64) เป็นต้น

คอนเทนเนอร์เอนจินจะทำงานบนเซิร์ฟเวอร์เครื่องเดียว อย่างไรก็ตาม มีเทคโนโลยีในการรวมเซิร์ฟเวอร์ที่รองรับการทำงานของคอนเทนเนอร์รวมกันเป็น **คลัสเตอร์** โดยมีเครื่องมาสเตอร์โนด

(master node) ทำหน้าที่สั่งการเครื่องเวิร์คเกอร์โนด (worker node) เรียกเทคโนโลยีนี้ว่า คอนเทนเนอร์ออร์เคสเตรชัน (container orchestration)

1.1.4 คอนเทนเนอร์ออร์เคสเตรชัน



รูปที่ 3 คอนเทนเนอร์ออร์เคสเตรเตอร์

เซิร์ฟเวอร์แต่ละโนดที่สามารถรันคอนเทนเนอร์ได้ จะถูกบริหารจัดการรวมเป็นคลัสเตอร์เดียวกัน โดยคอนเทนเนอร์ออร์เคสเตรเตอร์ มีหน้าที่บริหารทรัพยากรของเครื่องในคลัสเตอร์ทั้งหมด ทั้งการสั่งให้คอนเทนเนอร์ทำงาน (deployment) การขยายหรือลดขนาด (scaling) จำนวนคอนเทนเนอร์ และการเชื่อมต่อระหว่างคอนเทนเนอร์ (networking) [8] ออร์เคสเตรเตอร์มีให้เลือกใช้หลายตัว ได้แก่ ดอกเกอร์สวอม (docker swarm) ที่เป็นออร์เคสเตรเตอร์ของดอกเกอร์ หรือ คิวเบอร์เนเทส (Kubernetes) ที่เป็นออร์เคสเตรเตอร์ที่รองรับคอนเทนเนอร์เอนจินที่หลากหลายในคลัสเตอร์เดียวกันได้ วิทยานิพนธ์นี้เลือกใช้คิวเบอร์เนเทสเป็นคอนเทนเนอร์ออร์เคสเตรเตอร์ที่ใช้ศึกษา เนื่องจากมีความยืดหยุ่นในการใช้งานมากกว่า [9] เทคโนโลยีคอนเทนเนอร์ออร์เคสเตรเตอร์ มีความสามารถในการรักษาความพร้อมใช้ (availability) ของเซอร์วิสที่ให้บริการโดยคอนเทนเนอร์คล้ายกับเทคนิคของเทคโนโลยีเวอร์ชัวไลเซชัน ที่สามารถย้ายคอนเทนเนอร์ไปทำงานในโนดอื่นเพื่อดำเนินการในงานบำรุงรักษาได้ รวมทั้งรองรับกรณีเมื่อมีบางโนดหยุดทำงาน ก็สามารถย้ายคอนเทนเนอร์ไปรันในโนดอื่นได้ การบริหารจัดการคอนเทนเนอร์ในคลัสเตอร์เดียวกัน สามารถรองรับแอพลิเคชันได้หลายโครงการ อย่างไรก็ตามในองค์กรสามารถมีการจัดการคอนเทนเนอร์แบบหลายคลัสเตอร์ได้เพื่อการแยกเดี่ยวทรัพยากร (resource isolation)

1.1.5 การสมาพันธ์ระหว่างคลัสเตอร์

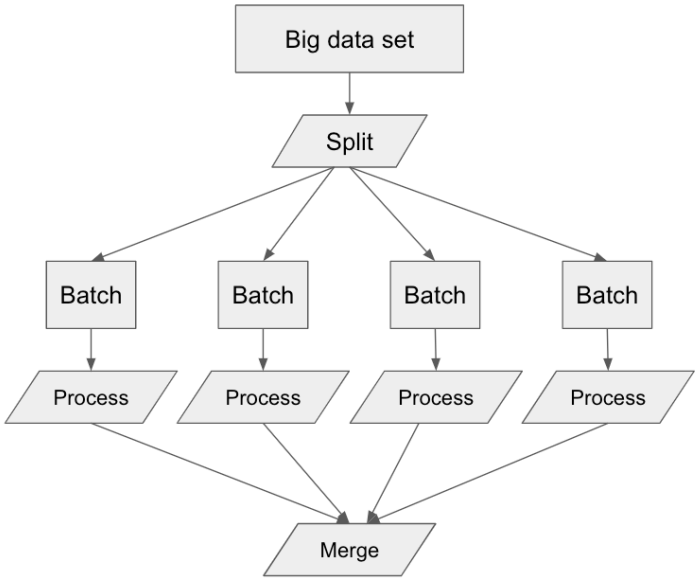
คอนเทนเนอร์ออร์เคสเตรชันที่เกิดขึ้น**ระหว่างคลัสเตอร์** เป็นความร่วมมือระหว่างคลัสเตอร์ร่วมสมาพันธ์ (federated cluster) เนื่องจากมีความต้องการใช้ทรัพยากรข้ามคลัสเตอร์ชั่วคราวในการขยายระบบ [10] การสมาพันธ์ระหว่างคลัสเตอร์ (cluster federation) มีกรณีการใช้งานดังนี้ [11]

1. ความจุล้น (capacity overflow) หรือ ความต้องการใช้คลาวด์สูงขึ้นอย่างฉับพลันเป็นช่วง ๆ (cloud bursting) เนื่องจากการสร้างคลัสเตอร์บนผู้ให้บริการคลาวด์ทั่วไป (public cloud) มีค่าใช้จ่ายสูง จึงมีการสร้างคลัสเตอร์ใช้งานบนคลาวด์ส่วนตัว (private cloud) เพื่อรองรับงานโดยทั่วไป แต่ในบางครั้งที่มีความต้องการใช้ทรัพยากรอย่างเร่งด่วน จึงอาจมีการเพิ่มความจุของคลัสเตอร์อื่นชั่วคราว การทำงานภายใต้การร่วมสมาพันธ์จะช่วยลดความซับซ้อนในการบริหารและลดค่าใช้จ่ายทรัพยากรที่ไม่ได้ใช้ในระยะยาว
2. ระบบงานที่อ่อนไหว (sensitive workload) บางครั้งการจัดการข้อมูลของหลายระบบงาน จะมีทั้งระบบงานที่มีข้อมูลอ่อนไหวไม่ควรนำระบบงานทั้งหมดไปทำงานบนผู้ให้บริการคลาวด์ และระบบงานทั่วไปที่สามารถทำงานบนผู้ให้บริการคลาวด์ใดก็ได้ หรือบางกรณีที่มีข้อตกลงเชิงกฎหมายร่วมกันเกี่ยวกับพื้นที่ที่กำหนดการติดตั้งของระบบ ข้อกำหนดของระบบงานต่าง ๆ ที่อาจเปลี่ยนแปลงได้ จึงจำเป็นต้องมีกระบวนการรองรับนโยบายที่อาจเปลี่ยนแปลงฉับพลัน
3. การหลีกเลี่ยงการผูกขาดของผู้ให้บริการ (avoiding vendor lock-in) การไม่ผูกขาดอยู่กับผู้ให้บริการเป็นสิ่งสำคัญเนื่องจากผู้ให้บริการอาจปิดตัวหรือมีระดับการให้บริการไม่เท่ากัน ดังนั้นการสร้างคลัสเตอร์ร่วมสมาพันธ์ จะทำให้มีแผนสำรอง และป้องกันการผูกขาดของผู้ให้บริการได้
4. ความพร้อมใช้งานระดับสูงในเชิงการกระจายทางภูมิศาสตร์ (geo-distributing high availability) ความพร้อมใช้งานระดับสูง คือการที่ระบบจะคงการให้บริการอยู่ได้แม้จะมีระบบงานบางส่วนเสียหาย เช่นเมื่อเกิดความเสียหายเชิงกายภาพของสถานที่ อาจทำให้ทั้งคลัสเตอร์เสียหาย แต่หากมีการร่วมสมาพันธ์ระหว่างคลัสเตอร์ ระบบจะคงให้บริการในต่างคลัสเตอร์ได้

1.1.6 การเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่

อัลกอริทึมแบ่งแยกและเอาชนะ (divide and conquer) เป็นเทคนิคการแบ่งปัญหาใหญ่เป็นปัญหาย่อย และเมื่อแก้ปัญหาย่อยได้ คำตอบเหล่านั้นจะรวมกันเป็นคำตอบของปัญหาใหญ่ [12] แมพรีดิว (map reduce) คือรูปแบบการประยุกต์อัลกอริทึมแบ่งแยกและเอาชนะแบบหนึ่ง [13]

วิทยานิพนธ์นี้มีแสดงการเขียนโปรแกรมแมพรีดิวเพื่อเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่ ตามโครงสร้างดัง รูปที่ 4 แสดงเทคนิคการประมวลผลข้อมูลขนาดใหญ่



รูปที่ 4 เทคนิคการประมวลผลข้อมูลขนาดใหญ่

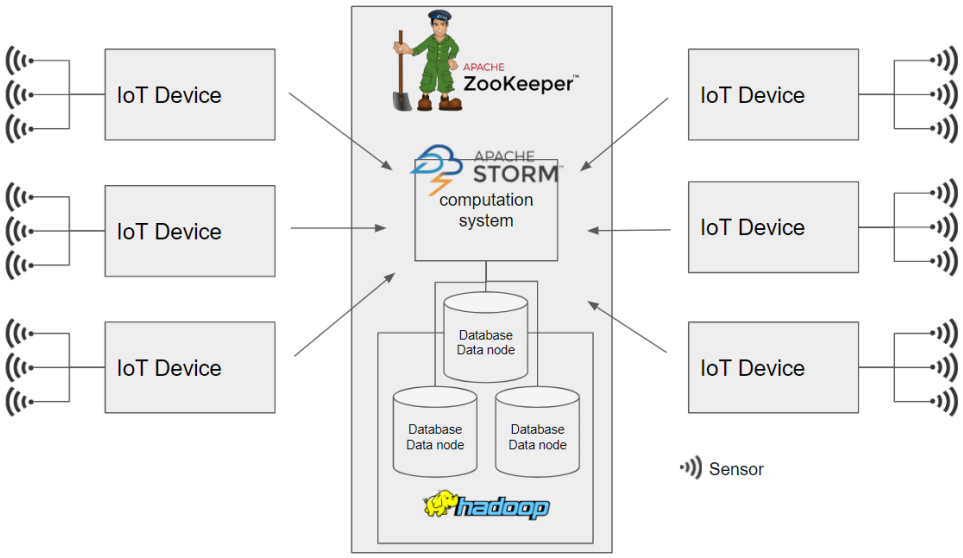
ขั้นตอนการกระจายงาน จะเริ่มตั้งแต่การแบ่งกลุ่มของข้อมูลเป็นส่วน ๆ จากนั้นกำหนดการประมวลผลแบบเป็นชุด (batch) และประมวลข้อมูลด้วยชุดคำสั่งเดียวกัน ผลลัพธ์ที่ได้จะนำมารวมคำตอบกัน จนได้เป็นคำตอบของปัญหาใหญ่ เทคโนโลยีคอนเทนเนอร์ช่วยในการเคลื่อนย้ายแอพลิเคชันในการจัดการข้อมูล และสั่งให้คอนเทนเนอร์ทำงานได้ทันที เทคโนโลยีออร์เคสเตรชันจะช่วยรวมศูนย์การจัดการคอนเทนเนอร์ ในการบริหารทรัพยากรในระดับคลัสเตอร์ และกระจายแอพลิเคชันในการจัดการข้อมูลไปทุกโนดในคลัสเตอร์ ทำให้สามารถประมวลผลข้อมูลได้พร้อมกันในแต่ละคอนเทนเนอร์ และรวมเฉพาะคำตอบในท้ายที่สุด

1.2 งานวิจัยที่เกี่ยวข้องเกี่ยวกับสถาปัตยกรรมระบบพลังงานอัจฉริยะ

ความท้าทายของการประมวลผลข้อมูลขนาดใหญ่ของการจัดการพลังงานอัจฉริยะประกอบด้วยการจัดการข้อมูลจากหลายต้นทาง (multi-source data) การจัดเก็บข้อมูล (data storage) และการประมวลผลข้อมูลให้ทันกาล (real-time data processing) [14] การจัดการพลังงานอัจฉริยะจะมีบทบาทในการปรับใช้การจัดการพลังงานแบบดั้งเดิมไปสู่รูปแบบที่ทันสมัย หัวใจของการเปลี่ยนแปลงไปสู่การจัดการข้อมูลที่ทันสมัยต้องอาศัยเทคนิคการจัดการข้อมูลขนาดใหญ่ และการสื่อสารระหว่างผู้ใช้ไฟฟ้าที่อาจเป็นได้ทั้งผู้ผลิตและผู้บริโภคพลังงานได้ในคนเดียวกัน ความท้าทายของการจัดการข้อมูลต่าง ๆ ให้มีประสิทธิภาพ ประกอบไปด้วย การนำมาให้ได้ซึ่งข้อมูล

(data acquisition) การย้ายข้อมูล (data transmission) การจัดการข้อมูล (data processing) การนำเสนอข้อมูล (data visualization) การตีความข้อมูล (data interpretation) และการใช้ประโยชน์จากคลื่นข้อมูลขนาดใหญ่ที่เชื่อมโยงกับศูนย์กลางการจัดเก็บข้อมูล (utilization of big data streams and combined data lake) [15] กระบวนการทั้งหมดนี้รวมเรียกว่า การจัดการข้อมูลให้พร้อมใช้ (data wrangling) ระบบพลังงานอัจฉริยะจะสามารถจัดเก็บข้อมูลจากตัววัดที่หลากหลายและถูกจัดเก็บในลักษณะที่ง่ายต่อการนำไปวิเคราะห์ [16] โดยทั่วไปสถาปัตยกรรมระบบพลังงานอัจฉริยะที่มีการวิเคราะห์ข้อมูลขนาดใหญ่ ต้องออกแบบกลยุทธ์การจัดการข้อมูลล่วงหน้า [15]

รูปแบบที่ปรากฏจากการค้นคว้าวิจัยในขอบเขตพลังงานอัจฉริยะ มีการนำเสนอเครื่องมือการจัดการข้อมูลขนาดใหญ่ โดยมีซอฟต์แวร์อาทิ Apache Hadoop, Apache Storm และ Apache Drill อยู่เบื้องหลังการทำงานจัดการข้อมูลที่เป็นที่นิยมและเหมาะกับการจัดการพลังงานที่สามารถรองรับการจัดการข้อมูลขนาดใหญ่และการประมวลผลอย่างทันกาล [15] การจัดการข้อมูลของกลไกเหล่านี้อาศัยการประมวลผลข้อมูลขนาดใหญ่พร้อมกัน (large-scale parallel system) ที่มีความท้าทายในการออกแบบสถาปัตยกรรมระบบทั้งข้อจำกัดเชิงกายภาพ และระบบอื่นที่เกี่ยวข้องในการร่วมประสานการนำข้อมูลไปใช้ สถาปัตยกรรมระบบเหล่านี้จึงต้องอาศัย Apache Hadoop ในการเปิดคุณสมบัติการจัดการประมวลผลแบบหลายโนด (multi-node computing) และเครื่องมือในการนำข้อมูลที่จัดเก็บนี้ไปวิเคราะห์ได้แก่ Apache Drill และ Apache Storm อีกทั้งต้องอาศัยซอฟต์แวร์ ZooKeeper ที่สำคัญในการจัดการข้อมูลที่จัดเก็บโดย Hadoop cluster [17], [18] เนื่องด้วยความซับซ้อนในการเปิดคุณสมบัติต่าง ๆ การสร้างสถาปัตยกรรมระบบในการจัดการข้อมูลขนาดใหญ่โดยมากจึงต้องมีการออกแบบอย่างรอบคอบและนำไปใช้ระยะยาว จากสถาปัตยกรรมระบบดังกล่าวเมื่อจัดวางซอฟต์แวร์ Apache Hadoop, Apache Storm และ ZooKeeper จะเป็นดังรูปที่ 5 โดย ZooKeeper จะตรวจสอบความพร้อมใช้ของแต่ละองค์ประกอบ โดยมีส่วนการประมวลผลข้อมูลที่ Apache Storm และมี Apache Hadoop รับงานแบบประมวลผลพร้อมกันและทำซ้ำข้อมูลระหว่างโนดที่ผู้ดูแลระบบสามารถบำรุงรักษาจัดหาโนดทดแทนได้โดยไม่หยุดให้บริการ การจัดเก็บข้อมูลและประมวลผลที่ศูนย์กลาง มีการนำเสนอในงานวิจัยและบทสำรวจสถาปัตยกรรมระบบพลังงานอัจฉริยะอาทิ [14] [15] และ [16]

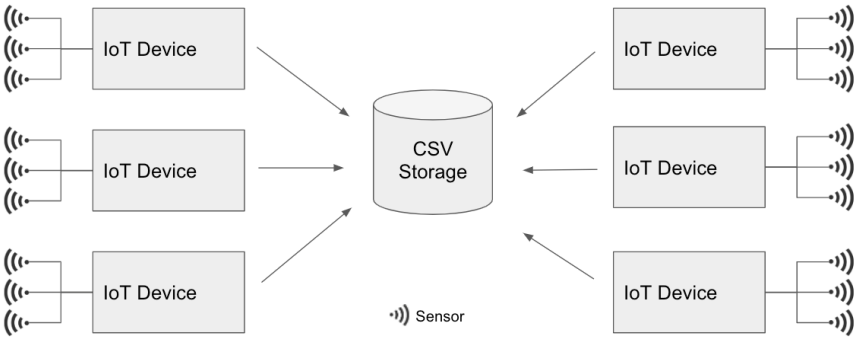


รูปที่ 5 การประยุกต์สถาปัตยกรรมระบบฐานข้อมูลแบบหลายโนดกับซอฟต์แวร์จัดการข้อมูล

การเริ่มต้นศึกษาการจัดการข้อมูลขนาดใหญ่จึงต้องติดตั้งซอฟต์แวร์ต่าง ๆ ที่อยู่ภายใต้สถาปัตยกรรมระบบที่ออกแบบเป็นจำนวนมาก ที่ยังไม่รวมส่วนของการคำนวณ เพื่อลดอุปสรรคขั้นตอนในการสร้างกระบวนการจัดการข้อมูลขนาดใหญ่ วิทยานิพนธ์นี้จะนำเสนอสถาปัตยกรรมระบบพลังงานอัจฉริยะที่มีความยืดหยุ่น กล่าวคือสามารถปรับเปลี่ยนกลยุทธ์ในการจัดการข้อมูลได้ง่าย และนำเสนอการประมวลผลพร้อมกันได้

1.3 ที่มาการจัดการข้อมูลเพื่อประมวลผลในโครงการ CU-BEMS

กรณีศึกษาโครงการ CU-BEMS ของ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย มีการพัฒนาต้นแบบระบบจัดการพลังงานในอาคาร (BEMS – building energy management system) [19] โดยการติดตั้งเซนเซอร์ และจัดเก็บข้อมูล โดยมีรูปแบบการจัดเก็บข้อมูลแบบรวมศูนย์ดังรูปที่ 6 แสดงการจัดเก็บข้อมูลจากอุปกรณ์ไอโอทีในโครงการ CU-BEMS ที่มีการบันทึกเป็นไฟล์ CSV แยกตามรายการอุปกรณ์ที่ศูนย์กลาง



รูปที่ 6 การจัดเก็บข้อมูลอุปกรณ์ไอโอทีในโครงการ CU-BEMS

การประมวลผลข้อมูลไฟล์ CSV หากใช้วิธีการประมวลผลพร้อมกันด้วยเทคนิคการวิเคราะห์ข้อมูลขนาดใหญ่ จะช่วยให้การประมวลผลเร็วขึ้นได้ และเทคโนโลยีคอนเทนเนอร์จะช่วยให้การยืดหดของจำนวนคอนเทนเนอร์ที่ช่วยในการประมวลผลข้อมูลทำได้สะดวกขึ้น มีแนวคิดในการออกแบบดังนี้คือ

1. สามารถขยายขนาดทรัพยากรที่ใช้ในการประมวลผลได้
2. เทคนิคที่ใช้ในการวิเคราะห์ข้อมูล ต้องสามารถแบ่งงานและทำงานพร้อมกันได้
3. สามารถเปลี่ยนสูตรการวิเคราะห์ข้อมูลได้ง่าย

จากแนวคิดดังนี้จึงพัฒนากรอบการทำงาน (framework) เพื่อแสดงตัวอย่างการวิเคราะห์ข้อมูลพลังงานอัจฉริยะบนข้อมูลจริง ด้วยเทคโนโลยีคอนเทนเนอร์ และคอนเทนเนอร์ออร์เคสเตรชัน

1.4 ที่มาการออกแบบสถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง

โครงสร้างพื้นฐานที่รองรับการวิจัยเทคโนโลยีไอโอทีและการทดสอบการเพิ่มความเร็วการประมวลผลข้อมูลจาก CU-BEMS ได้รับการสนับสนุนจากโครงการ IoTcloudServe@TEIN ในการสร้างแพลทฟอร์มเรียนรู้เทคโนโลยีไอโอทีร่วมกัน จึงได้ใช้เทคโนโลยีคอนเทนเนอร์ออร์เคสเตรชันบนคอมพิวเตอร์เซิร์ฟเวอร์ในโครงการทั้งหมด การให้บริการมีความเกี่ยวข้องกับผู้ใช้งานหลายคน และเพื่อให้สามารถให้บริการได้ต่อเนื่อง จึงพัฒนาสถาปัตยกรรมระบบที่สามารถให้บริการจริงได้ (production) โดยพัฒนาคุณสมบัติความพร้อมใช้ระดับสูง (high availability) โดยติดตั้งการทำงานจริงและพิสูจน์คุณสมบัติดังกล่าว

นอกเหนือจากความพร้อมใช้ระดับสูง เมื่อมีผู้ใช้บริการไอโอทีแพลทฟอร์มในการรันคอนเทนเนอร์มากขึ้น จำเป็นต้องกำกับดูแลความเรียบร้อยในการให้บริการ โดยมีผู้ใช้งานทั้งนักพัฒนาแอพลิเคชันและผู้ดูแลระบบ (system admin) ในการบำรุงรักษาแพลทฟอร์ม จึงมีการออกแบบสถาปัตยกรรมระบบให้รองรับการจัดการผู้ใช้งาน (user management) วิทยานิพนธ์นี้จะติดตั้งและพิสูจน์คุณสมบัติการจัดการผู้ใช้งานแบบหลายโครงการให้ปลอดภัย

1.5 ที่มาการออกแบบสถาปัตยกรรมระบบการร่วมสมาพันธ์คลัสเตอร์โดยผู้ดูแลระบบหลายกลุ่ม

นอกเหนือจากในบริบทของการทำงานของคอนเทนเนอร์ในคลัสเตอร์เดียวกัน มีกรณีการสร้างการร่วมสมาพันธ์คลัสเตอร์ (cluster federation) คือการสร้างคอนเทนเนอร์ออร์เคสเตรชันแบบหลายคลัสเตอร์ ซึ่งมีคุณประโยชน์ดังที่ระบุในหัวข้อ 1.1.5 การร่วมสมาพันธ์คลัสเตอร์สามารถยืมทรัพยากรระหว่างคลัสเตอร์ได้ โดยปกติการจัดการหลายคลัสเตอร์สามารถใช้เครื่องมือรานเชอร์ (Rancher) ในการบริหารโครงการแบบหลายคลัสเตอร์ได้ อย่างไรก็ตามในด้านความปลอดภัย กรณีการร่วมสมาพันธ์คลัสเตอร์ระหว่างผู้ดูแลระบบต่างกลุ่ม เครื่องมือรานเชอร์จะไม่สามารถจำกัดสิทธิในการเข้าถึงทรัพยากรระหว่างกลุ่มที่ต้องการควบคุมได้ กล่าวคือ ผู้ดูแลระบบแต่ละกลุ่มจะเข้าควบคุม

ทรัพยากรข้ามหน่วยงานได้ทั้งหมดหากใช้โปรแกรมรานเชอร์ วิทยานิพนธ์นี้จึงค้นคว้าวิจัยการออกแบบคอนเทนเนอร์ออร์เคสเตรเตอร์ของกรณีการร่วมสมาพันธ์ (federation) คอนเทนเนอร์ออร์เคสเตรชันหลายกลุ่ม เช่นในบริบทของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทยที่มีหลายหน่วยงานย่อยในองค์กร แต่ละหน่วยงานมีการจัดซื้อเซิร์ฟเวอร์ใช้งานเฉพาะ เมื่อมีการพัฒนากลไกการร่วมสมาพันธ์คลัสเตอร์จะสามารถยืมทรัพยากรข้ามหน่วยงานเพื่อใช้ประโยชน์ร่วมกัน

งานวิจัยที่นำเสนอเครื่องมือในการจัดการคอนเทนเนอร์ออร์เคสเตรชันหลายกลุ่ม ส่วนมากออกแบบเพื่อเพิ่มประสิทธิภาพเมื่อมีการสร้างการร่วมสมาพันธ์ เช่น งานวิจัย [20] นำเสนอการเปิดคุณสมบัติการเคลื่อนย้าย (portability) ของแอพลิเคชันแบบคลาวด์ [21] เพื่อ**สร้างออร์เคสเตรเตอร์**ที่สามารถรันแอพลิเคชันแบบคลาวด์ที่ขยายขนาดอัตโนมัติได้ พบว่าเมื่อมีการร่วมสมาพันธ์ระหว่างคิวเบอร์เนเทสหลายคลัสเตอร์ สามารถรองรับภาระงานสูงได้มีประสิทธิภาพ เป็นการสนับสนุนประโยชน์การร่วมสมาพันธ์คิวเบอร์เนเทสแต่ไม่พิจารณาประเด็นเรื่องความปลอดภัยกรณีผู้ดูแลระบบหลายหน่วยงานทำงานร่วมกัน

งานวิจัย [22] นำเสนอการพัฒนาหน่วยควบคุม ให้สามารถรองรับคิวเบอร์เนเทสคลัสเตอร์เป็นจำนวนในระดับหลายพันกลุ่มได้ เพื่อรองรับกรณีการประมวลผลแบบเอช (edge computing) งานวิจัยเสนอว่า คิวเฟ็ด [23] (เครื่องมือหนึ่งในการจัดการคิวเบอร์เนเทสแบบร่วมสมาพันธ์) ไม่เหมาะสำหรับการพัฒนาการประมวลแบบเอชเนื่องจากมีลักษณะเป็นจุดเดียวของความล้มเหลว (single point of failure) ซึ่งมีความเสี่ยงในการพัฒนาระบบเชิงกระจายขนาดใหญ่ นอกจากนี้รองรับจำนวนคลัสเตอร์ได้น้อย ซึ่งทำให้เกิดช่องว่างทางเวลา (latency) ระหว่างเอชคลัสเตอร์ (edge cluster) ไปสู่หน่วยควบคุมศูนย์กลาง งานวิจัย [22] ได้นำเสนอวิธีแก้ปัญหาการจัดการคิวเฟ็ดแบบรวมศูนย์เป็นแบบเชิงกระจาย โดยการสร้าง**ฐานข้อมูล**ที่บันทึกทรัพยากรที่ร่วมสมาพันธ์ (federated resource) และสร้าง**หน่วยควบคุม**การร่วมสมาพันธ์กระจายไปในแต่ละคลัสเตอร์ (distributed federation controller) โดยมีการทำซ้ำของข้อมูล (replication) ระหว่างกัน ในลักษณะที่สามารถทำให้เกิดความสอดคล้องกันในท้ายที่สุด (eventually consistency) งานวิจัย [22] มีการพึ่งพาคุณสมบัติของการทำซ้ำกันระหว่างฐานข้อมูลในการกระจายข้อมูลสถานะโดยรวมไปทุกคลัสเตอร์ ทำให้หน่วยควบคุมของแต่ละโนดสามารถเห็นสถานะของโนดอื่น ๆ ในคลัสเตอร์ได้ทั้งหมด อย่างไรก็ตามการตัดสินใจของแต่ละโนดเป็นแบบเชิงกระจาย โดยตัดสินบนสถานะของข้อมูลที่ปรับให้สอดคล้องกันและขึ้นอยู่กับข้อมูลขณะหนึ่งของแต่ละคลัสเตอร์ งานวิจัยมีการอ้างอิงถึงงานวิจัยที่คล้ายกัน [24] ในการสร้างออร์เคสเตรชันเชิงกระจายเพื่อการจัดการคอนเทนเนอร์ในระดับเอชของคลัสเตอร์หลายพันโนด และใช้สร้างหน่วยควบคุมเชิงกระจายเหมือนงานวิจัย [22] ในชื่อ ด็อกมา (DOCMA) ซึ่งเป็นการออกแบบออร์เคสเตรเตอร์ใหม่ที่ทำหน้าที่คล้ายคิวเบอร์เนเทส งานวิจัย [22] เชื่อว่าคิวเบอร์เนเทสจะมีสมรรถนะสูงกว่าในเชิงอุตสาหกรรม มากกว่าการสร้างออร์เคสเตรเตอร์แทนที่

การจัดการความปลอดภัยในการจัดการทรัพยากรระหว่างคลัสเตอร์ งานวิจัยด้านการจัดการคลัสเตอร์ร่วมสมาพันธ์ พบว่าขาดปัจจัยพิจารณาความปลอดภัยในการสร้างความร่วมมือระหว่างคลัสเตอร์โดยผู้ให้บริการหลายกลุ่ม ดังนี้

งานวิจัย [20] มีข้อสังเกตคือ คิวเบอร์เนเทสคลัสเตอร์ทั้งหมดทำงานภายใต้ผู้ให้บริการคลาวด์รายเดียว และ ไม่มีการควบคุมปัจจัยการเข้าถึงทรัพยากรคิวเบอร์เนเทสที่อาจแตกต่างกันแต่ละกลุ่ม แต่สามารถแสดงประโยชน์ของการร่วมสมาพันธ์โดยการขอใช้ทรัพยากรเพื่อรองรับกรณีความจุล้นได้

งานวิจัย [22] นำเสนอการออกแบบหน่วยควบคุมเชิงกระจาย ซึ่งมีการเก็บข้อมูลสถานะของทุกคลัสเตอร์กระจายไปทุกคลัสเตอร์ ทำให้ข้อมูลมีความซ้ำซ้อน และมีการใช้คุณสมบัติของฐานข้อมูลทดแทนการอัพเดทสถานะล่าสุดของทุกคลัสเตอร์ (database replication) ซึ่งอาจไม่เหมาะกับการนำไปใช้ในการตัดสินใจการออร์เคสเตรตคอนเทนเนอร์ระดับกลุ่มคลัสเตอร์ของทุกคลัสเตอร์โดยตรง เนื่องจากควรมีสถานะของทั้งคลัสเตอร์ในขณะหนึ่ง ๆ เพียงแบบเดียว และสามารถตัดสินใจการออร์เคสเตรตคอนเทนเนอร์ที่เหมาะสมที่สุดบนทุกคลัสเตอร์ได้ สำหรับกรณีจุดเดียวของความล้มเหลวอาจกำหนดให้หน่วยควบคุมมีความพร้อมใช้ระดับสูงหรือกระจายอยู่หลายโนด โดยไม่จำเป็นต้องกระจายอยู่ทุกโนด

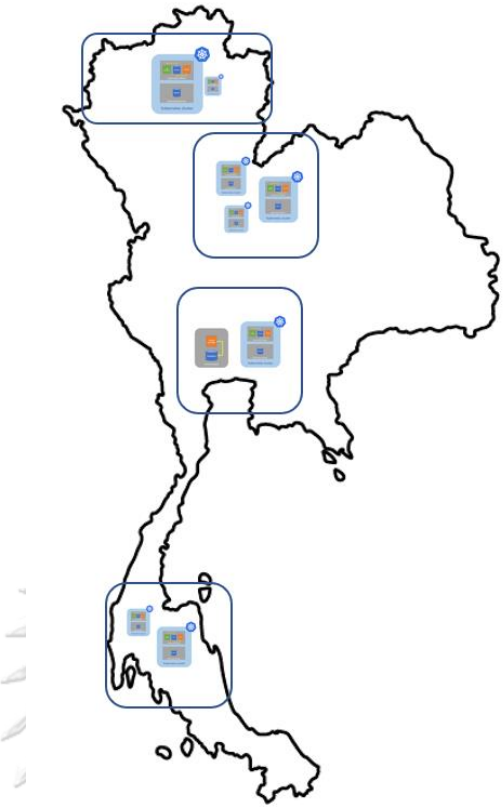
1.6 สรุปปัญหาและข้อจำกัดของงานวิจัยที่ผ่านมา

จากการนำเสนอที่มาในการออกแบบสถาปัตยกรรมระบบต่าง ๆ เพื่อรองรับเทคโนโลยีพลังงานอัจฉริยะ ในการติดตั้งและทดสอบคุณสมบัติของสถาปัตยกรรม จึงขอสรุปปัญหาการออกแบบสถาปัตยกรรมต่าง ๆ ดังนี้

1. การออกแบบและติดตั้งสถาปัตยกรรมระบบในการเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่
2. การออกแบบและติดตั้งสถาปัตยกรรมระบบในการจัดการรุ่นแอพลิเคชันแบบรวมศูนย์ได้
3. การออกแบบและติดตั้งสถาปัตยกรรมระบบการให้บริการคอนเทนเนอร์ออร์เคสเตรชันที่มีความพร้อมใช้งานระดับสูง
4. การออกแบบและติดตั้งสถาปัตยกรรมระบบการจัดการผู้ใช้งานและการแยกกลุ่มทรัพยากร
5. การออกแบบและติดตั้งสถาปัตยกรรมระบบการจัดการคอนเทนเนอร์หลายคลัสเตอร์แบบผู้ใช้งานหลายกลุ่ม

จากการสืบค้นงานวิจัยต่าง ๆ ไม่พบงานวิจัยที่ศึกษาการร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์ กรณี **การร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์ แบบผู้ดูแลระบบหลายกลุ่ม** เป็นการยกระดับความปลอดภัยในการสร้างกลไกการร่วมสมาพันธ์คลัสเตอร์ของผู้ดูแลระบบหลายกลุ่ม

การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย มีหน่วยงานย่อยที่เป็นกลุ่มผู้ดูแลระบบ กล่าวคือมีการจัดการคลาวด์ส่วนตัวเป็นคลัสเตอร์แต่ละคลัสเตอร์แยกหน่วยงานกัน จึงคาดว่าแนวคิดการร่วมสมาพันธ์การออร์เคสเตรตคอนเทนเนอร์จะเกิดประโยชน์ในการดำเนินงาน โดยเฉพาะเทคโนโลยีพลังงานอัจฉริยะต่าง ๆ รูปที่ 7 แสดงคลัสเตอร์เซิร์ฟเวอร์ของกลุ่มผู้ดูแลระบบในการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย โดยทั้งหมดเป็นกลุ่มผู้ดูแลระบบที่ใช้เทคโนโลยีเซิร์ฟเวอร์มาตรฐาน หรือเวอร์ชัวแมชชีน



รูปที่ 7 คลัสเตอร์เซิร์ฟเวอร์ของกลุ่มผู้ดูแลระบบในการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

การใช้เทคโนโลยีเวอร์ชัวไลเซชันของทุกกลุ่มผู้ดูแลระบบใน การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย สามารถพัฒนาบริการคอนเทนเนอร์ด้วยเทคโนโลยีคอนเทนเนอร์ออร์เคสเตรชันได้ไม่ยาก ทำให้สามารถใช้ประโยชน์จากเทคโนโลยีคอนเทนเนอร์ได้ และผู้ดูแลระบบแต่ละกลุ่มจัดสรรทรัพยากรระบบภายในคลัสเตอร์ของตัวเอง การเปิดคุณสมบัติคลัสเตอร์แบบร่วมสมาพันธ์ จะเป็นการยืมใช้ทรัพยากรระหว่างกลุ่มชั่วคราว ทำให้ได้ประโยชน์จากการร่วมสมาพันธ์ของภาระที่ติดตั้งข้ามคลัสเตอร์ได้ และสามารถจำกัดการเข้าถึงทรัพยากรระหว่างกลุ่มในออร์เคสเตรเตอร์ที่ออกแบบได้

1.7 วัตถุประสงค์

วิทยานิพนธ์ฉบับนี้ มีวัตถุประสงค์เพื่อนำเสนอ**สถาปัตยกรรมระบบ**พลังงานอัจฉริยะแบบคลาวด์ ของคลัสเตอร์ที่ร่วมสมาพันธ์**โดยผู้ดูแลระบบหลายกลุ่ม** โดยมีกรณีศึกษาต่าง ๆ ที่เป็นพื้นฐานเพื่อการศึกษาและพัฒนาต่อยอดการประยุกต์ในบริบทของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

1.8 ขอบเขต

วิทยานิพนธ์ฉบับนี้ นำเสนอ**สถาปัตยกรรมระบบ**พลังงานอัจฉริยะแบบคลาวด์ในการติดตั้งและพิสูจน์คุณสมบัติ ได้แก่สถาปัตยกรรมระบบเพื่อเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่ สถาปัตยกรรมระบบการให้บริการคอนเทนเนอร์ออร์เคสเตรชันที่มีความพร้อมใช้งานระดับสูง สถาปัตยกรรมระบบการจัดการผู้ใช้งานและการจัดกลุ่มทรัพยากร และสถาปัตยกรรมระบบการจัดการคอนเทนเนอร์หลายคลัสเตอร์แบบผู้ใช้งานหลายกลุ่ม

1.9 ขั้นตอนการศึกษา

1. ศึกษาทดลองสร้างคิวเบอร์เนเทสแบบหลายคลัสเตอร์
2. ออกแบบและสร้างวิธีการควบคุมคิวเบอร์เนเทสหลายคลัสเตอร์ผ่าน API
3. ออกแบบสถาปัตยกรรมระบบที่มีคุณสมบัติดังนี้
    a. สถาปัตยกรรมระบบในการเพิ่มความเร็วการวิเคราะห์ข้อมูลขนาดใหญ่
    b. สถาปัตยกรรมระบบในการจัดการรุ่นแอพลิเคชันแบบรวมศูนย์ได้
    c. สถาปัตยกรรมระบบที่มีความพร้อมใช้งานระดับสูง
    d. สถาปัตยกรรมระบบการจัดการผู้ใช้งานและจัดกลุ่มทรัพยากร
    e. สถาปัตยกรรมระบบการจัดการคอนเทนเนอร์หลายคลัสเตอร์แบบผู้ใช้งานหลายกลุ่ม
4. เปรียบเทียบข้อดี ข้อเสีย ข้อสังเกตต่าง ๆ ของแต่ละเทคโนโลยี และแนวทางปรับใช้ในหน่วยงานการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย โดยหน่วยงานการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย จะได้ประโยชน์ทั้งส่วนของผู้ให้บริการที่เพิ่มประสิทธิภาพงานบำรุงรักษาโครงสร้างพื้นฐาน และส่วนของผู้พัฒนาระบบในการจัดการแอพลิเคชันพลังงานอัจฉริยะหรือการสนับสนุนการดำเนินงานต่าง ๆ ได้สะดวก

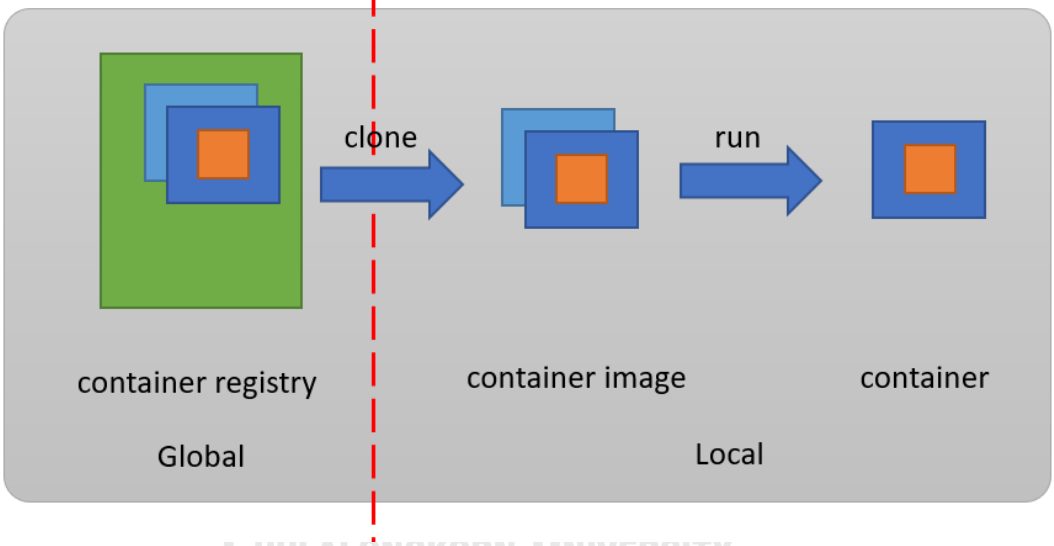
1.10ประโยชน์ที่คาดว่าจะได้รับจากโครงร่างวิทยานิพนธ์

1. ได้แนวทางการสร้างคิวเบอร์เนเทสคลัสเตอร์
2. ได้แนวทางการสร้างการร่วมสมาพันธ์คิวเบอร์เนเทสคลัสเตอร์กรณีผู้ดูแลระบบหลายกลุ่ม
3. ได้แนวทางการเพิ่มความเร็วการประมวลผลเชิงกระจายสำหรับการวิเคราะห์ข้อมูล
4. ได้สถาปัตยกรรมระบบที่รองรับการทำไอโอทีแพลทฟอร์ม และรองรับ
5. ได้แนวทางพัฒนาการให้บริการคอนเทนเนอร์ใน การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

บทที่ 2
พื้นฐานของเทคโนโลยีคอนเทนเนอร์ ออร์เคสเตรเตอร์ การร่วมสมาพันธ์ และไอโอทีแพลทฟอร์มที่ใช้สร้างระบบทดสอบตามสถาปัตยกรรมต่าง ๆ

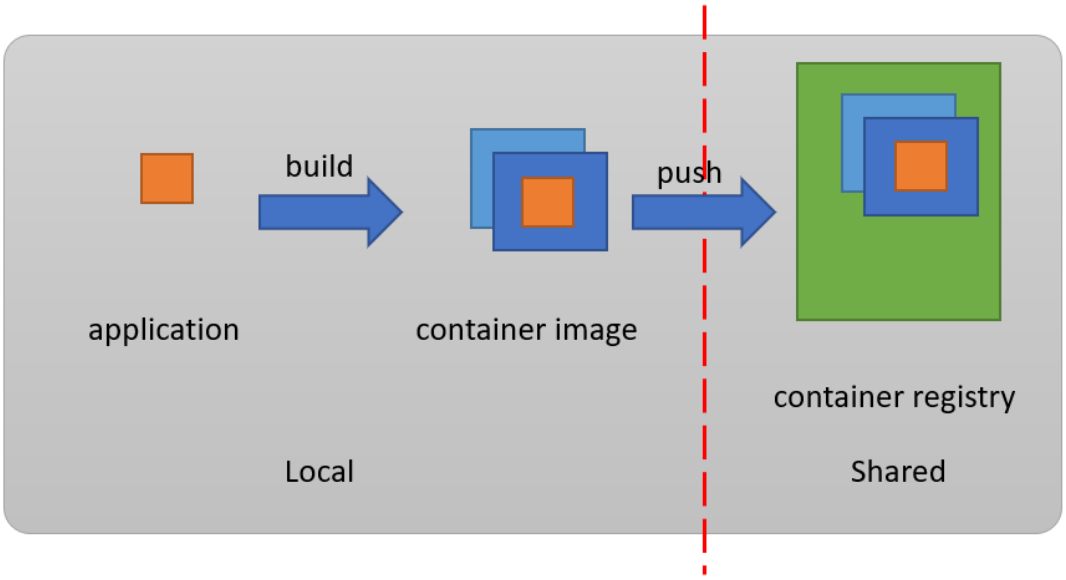
2.1 เทคโนโลยีคอนเทนเนอร์

คอนเทนเนอร์เปรียบเสมือนตู้เก็บแอพลิเคชัน เมื่อจะนำคอนเทนเนอร์ไปใช้ จะต้องโคลน (clone) คอนเทนเนอร์อิมเมจ (container image) จากทะเบียนคอนเทนเนอร์ (container registry) ซึ่งทะเบียนคอนเทนเนอร์จะจัดเก็บคอนเทนเนอร์อิมเมจหลายแอพลิเคชัน หลายรุ่น และอาจเป็นเซิร์ฟเวอร์เครื่องหนึ่งที่แชร์คอนเทนเนอร์อิมเมจในระดับโกลบอล (global) ที่สามารถกำหนดสิทธิ์การเข้าถึงได้ หลังจากโคลนคอนเทนเนอร์อิมเมจมาในระดับท้องถิ่นแล้ว (local) จะสามารถสั่งรัน (run) คอนเทนเนอร์ให้ทำงาน รูปที่ 8 แสดงขั้นตอนการรันคอนเทนเนอร์



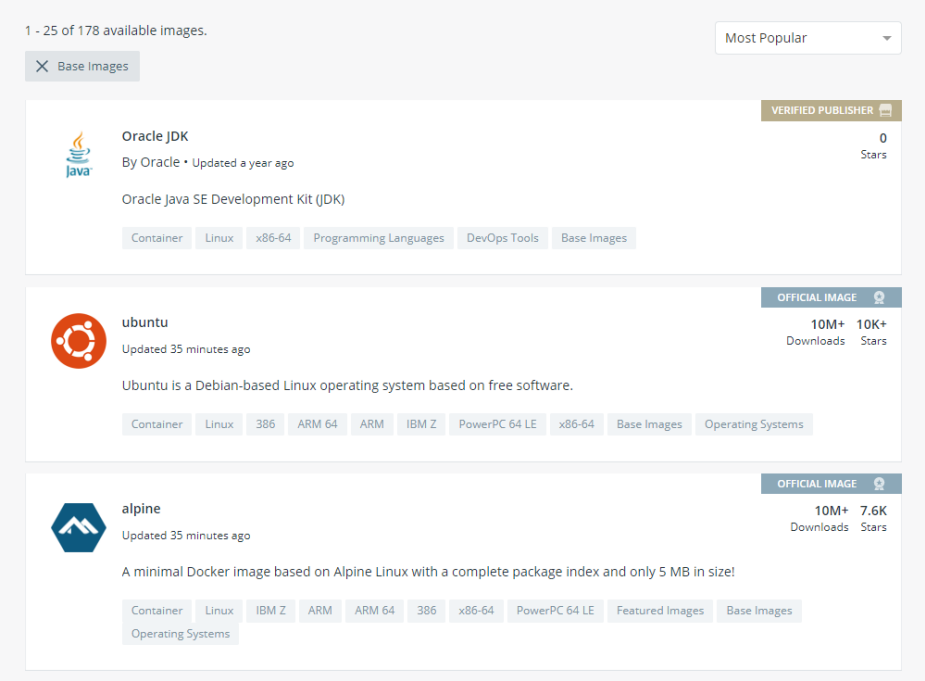
รูปที่ 8 ขั้นตอนการรันคอนเทนเนอร์

การมีคอนเทนเนอร์อิมเมจในทะเบียนคอนเทนเนอร์ แอพลิเคชันต้องผ่านกระบวนการคอนเทนเนอร์ไรซ์เซชัน โดยการกำหนดคอนเทนเนอร์ฐาน (base container) และสคริปต์ขั้นตอนการเตรียมคอนเทนเนอร์ให้พร้อมใช้ จากนั้นจึงใช้คำสั่งสร้างคอนเทนเนอร์ (container build) ให้ได้คอนเทนเนอร์อิมเมจในเครื่องท้องถิ่น และเผยแพร่คอนเทนเนอร์ไประดับโกลบอล ด้วยคำสั่ง ดันคอนเทนเนอร์ (container push) รูปที่ 9 แสดงขั้นตอนการเผยแพร่คอนเทนเนอร์อิมเมจ



รูปที่ 9 ขั้นตอนการเผยแพร่คอนเทนเนอร์อิมเมจ

รายการคอนเทนเนอร์ฐาน สามารถสืบค้นได้ที่ [25] เป็นบริการดอกเกอร์ฮับ (docker hub) เป็นทะเบียนคอนเทนเนอร์ระดับโกลบอลที่เปิดให้ใช้บริการทั่วไป คอนเทนเนอร์ฐานที่ใช้อาจเป็นคอนเทนเนอร์อิมเมจอื่น ๆ อีกชั้นก็ได้ รูปที่ 10 แสดงรายการคอนเทนเนอร์อิมเมจบนดอกเกอร์ฮับ



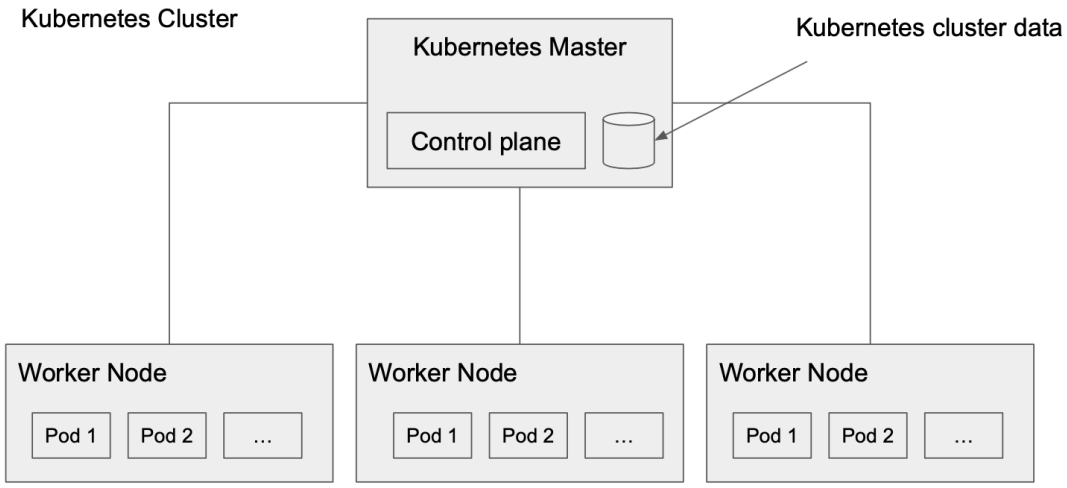
รูปที่ 10 รายการคอนเทนเนอร์อิมเมจ

2.2 องค์ประกอบหลักและการทำงานของออร์เคสเตรเตอร์แบบคิวเบอร์เนเทส

คลัสเตอร์ของเซิร์ฟเวอร์ คือกลุ่มของเซิร์ฟเวอร์ที่ทำงานร่วมกัน คิวเบอร์เนเทส คือ คอนเทนเนอร์ออร์เคสเตรเตอร์ที่จัดการคอนเทนเนอร์ในคลัสเตอร์ โดยมีมาสเตอร์โนดควบคุมสั่งการ คิวเบอร์เนเทสมีการจัดการทรัพยากรให้บริการคอนเทนเนอร์ที่สำคัญดังนี้

1. พอด (pod) คือ กลุ่มของคอนเทนเนอร์ เปรียบเสมือนเซิร์ฟเวอร์เครื่องหนึ่ง
2. ดีพลอยเมนต์ (deployment) คือ การประกาศโครงสร้างและจำนวนสำเนา (replica) ของพอด
3. เซอร์วิส (service) คือ กลไกการเผยแพร่ (expose) การเข้าถึงบริการของพอด
4. อินเกรส (ingress) คือ กลไกการเผยแพร่ (expose) แบบ HTTP หรือ HTTPS จากภายนอกคลัสเตอร์มาเซอร์วิส (service)

คิวเบอร์เนเทส 1 คลัสเตอร์ จะประกอบด้วยคิวเบอร์เนเทสมาสเตอร์โนดอย่างน้อย 1 โนด และประกอบด้วยเวิร์คเกอร์โนดหลายโนดหรือไม่มีก็ได้ รูปที่ 11 แสดงตัวอย่างโครงสร้างคิวเบอร์เนเทสคลัสเตอร์ โดยคิวเบอร์เนเทสคลัสเตอร์ จะมีระนาบควบคุม (control plane) กระจายไปทุกมาสเตอร์โนด และมีฐานข้อมูลที่จัดเก็บสถานะของคลัสเตอร์ ที่อาจอยู่ภายในมาสเตอร์โนด หรืออยู่ภายนอกมาสเตอร์โนดก็ได้



รูปที่ 11 ตัวอย่างคิวเบอร์เนเทสคลัสเตอร์

2.3 โปรแกรมรานเชอร์

รานเชอร์คือโปรแกรมที่ใช้ส่งคำสั่งควบคุมไปคิวเบอร์เนเทสคลัสเตอร์ ผลิตโดยบริษัทรานเชอร์ [26] โดยปกติคิวเบอร์เนเทสสามารถสั่งการได้โดยตรงผ่านเครื่องมือ คิวซีทีแอล (kubectl) ซึ่งเป็นเครื่องมือสั่งการคิวเบอร์เนเทสด้วยคำสั่งควบคุมคลัสเตอร์โดยตรง รานเชอร์จะนำเสนอองค์ประกอบต่าง ๆ ในคิวเบอร์เนเทสผ่านเว็บแอพลิเคชัน ช่วยให้ผู้ดูแลระบบสามารถควบคุมคลัสเตอร์ผ่านเว็บแอพลิเคชันและบริหารจัดการคลัสเตอร์ได้สะดวกขึ้น การรันโปรแกรมรานเชอร์จะรันแยกต่างหากจากคิวเบอร์เนเทสคลัสเตอร์ ความพร้อมใช้ของโปรแกรมรานเชอร์จึงไม่เกี่ยวโยงกับความพร้อมใช้ของคิวเบอร์เนเทสคลัสเตอร์ หากโปรแกรมรานเชอร์ไม่สามารถใช้งานได้ผู้ดูแลระบบจะสามารถสั่งการผ่านเครื่องมือคิวซีทีแอลบนมาสเตอร์โนดได้ดั่งเดิม รานเชอร์โปรแกรมหนึ่งสามารถควบคุมคิวเบอร์เนเทสหลายคลัสเตอร์ได้ แต่ไม่สามารถย้ายคอนเทนเนอร์ข้ามคลัสเตอร์ผ่านโปรแกรมรันเชอร์ได้ โปรแกรมรานเชอร์มีส่วนสำคัญในการออกแบบสถาปัตยกรรมระบบจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร

2.4 คำสำคัญที่เกี่ยวกับเทคโนโลยีคลาวด์

คลาวด์สาธารณะ (public cloud) หมายถึง บริการคลาวด์ที่ให้บริการทั่วไป โดยคิดค่าบริการตามปริมาณใช้งานและครอบคลุมค่าใช้จ่ายทั้งหมด ที่ผู้ใช้บริการไม่ต้องบำรุงรักษาโครงสร้างพื้นฐานเอง มีข้อดีคือมีบริการหลากหลาย และมีค่าใช้จ่ายตามจำนวนที่ใช้งาน เหมาะกับระบบงานใหม่ที่จำนวนผู้ใช้งานไม่ชัดเจน หรือระบบที่ต้องการคุณสมบัติการขยายระดับสูง (scalability) อีกทั้งมีทรัพยากรรองรับมหาศาล มีข้อเสียคือมีค่าใช้จ่ายสูงในระยะยาว และอาจมีความมั่นคงปลอดภัยไม่เพียงพอกับความต้องการ

คลาวด์ส่วนตัว (private cloud) หมายถึง การสร้างคลาวด์ส่วนตัวในองค์กร ที่ผู้ดูแลระบบบำรุงรักษาโครงสร้างพื้นฐานเอง จะมีค่าใช้จ่ายทั้งในส่วนการจัดซื้อโครงสร้างพื้นฐานและงานบำรุงรักษา มีข้อดีคือ มีความปลอดภัยสูง เหมาะสำหรับระบบที่รู้ขนาดและจำนวนผู้ใช้งานชัดเจน ทำให้มีค่าใช้จ่ายระยะยาวถูกกว่าแบบคลาวด์สาธารณะ มีข้อเสียคือ มีบริการไม่หลากหลาย มีงานบำรุงรักษา และอาจไม่คุ้มค่ากรณีติดตั้งระบบใหม่ที่จำนวนผู้ใช้งานไม่ชัดเจน

คลาวด์ลูกผสม (hybrid cloud) หมายถึง การใช้คลาวด์ส่วนตัวควบคู่กับคลาวด์สาธารณะ มีหลายรูปแบบการใช้งาน เช่นการใช้คลาวด์สาธารณะเป็นทรัพยากรสำรอง หรือเป็นทรัพยากรชั่วคราว ซึ่งเมื่อต้องการใช้ทรัพยากรกะทันหัน ทำให้สามารถโยกภาระงานไปคลาวด์สาธารณะได้ เมื่อภาระงานเสร็จสิ้นจึงหยุดใช้งาน และใช้คลาวด์ส่วนตัวเป็นหลัก คลาวด์ลูกผสมถูกนำเสนอในหลายบริบทในประเด็นการเปิดคุณสมบัติความพร้อมใช้งานระดับสูงยิ่งขึ้น หรือการทดสอบความสามารถของคลาวด์สาธารณะเป็นต้น การสร้างคลาวด์ลูกผสมพึงระวังกรณีความมั่นคงปลอดภัย เนื่องจากมีการเชื่อมต่อคลาวด์ส่วนตัวอย่างเป็นทางการ

คิวเบอร์เนเทสคลัสเตอร์ (Kubernetes cluster) คือ ผลิตภัณฑ์หนึ่งของเทคโนโลยีคอนเทนเนอร์ออร์เคสเตรชัน เป็นคำเรียกทั่วไปของการเรียกกลุ่มของโนดที่ติดตั้งคิวเบอร์เนเทสเอนจิน ในการออร์เคสเตรตคอนเทนเนอร์แบบรวมศูนย์ มีความสะดวกในการบำรุงรักษาเนื่องจากสามารถจัดการภาระ (workload) ได้มีประสิทธิภาพ

คิวเบอร์เนเทสคลัสเตอร์แบบหลายมาสเตอร์โนด (Kubernetes multi-master) คือ คิวเบอร์เนเทสคลัสเตอร์ที่มีมาสเตอร์โนดในคลัสเตอร์เดียวกันมากกว่า 1 โนด เพื่อเปิดคุณสมบัติความพร้อมใช้งานระดับสูงบนโครงสร้างพื้นฐาน ทำให้สามารถบำรุงรักษาเครื่องมาสเตอร์ทีละโนดได้หรือมีมาสเตอร์โนดบางโนดทำงานผิดปกติได้โดยไม่กระทบความต่อเนื่องการให้บริการ

คิวเบอร์เนเทสคลัสเตอร์แบบหลายผู้ให้บริการคลาวด์ (Kubernetes multi-cloud) คือ คิวเบอร์เนเทสคลัสเตอร์ที่มีโนดติดตั้งบนคลาวด์สาธารณะหรือคลาวด์ส่วนตัว 2 คลาวด์ขึ้นไป ซึ่งอาจเป็นได้ทั้งเวิร์คเกอร์โนดและมาสเตอร์โนด ทำให้สามารถอพยพภาระงานได้สะดวกขึ้น

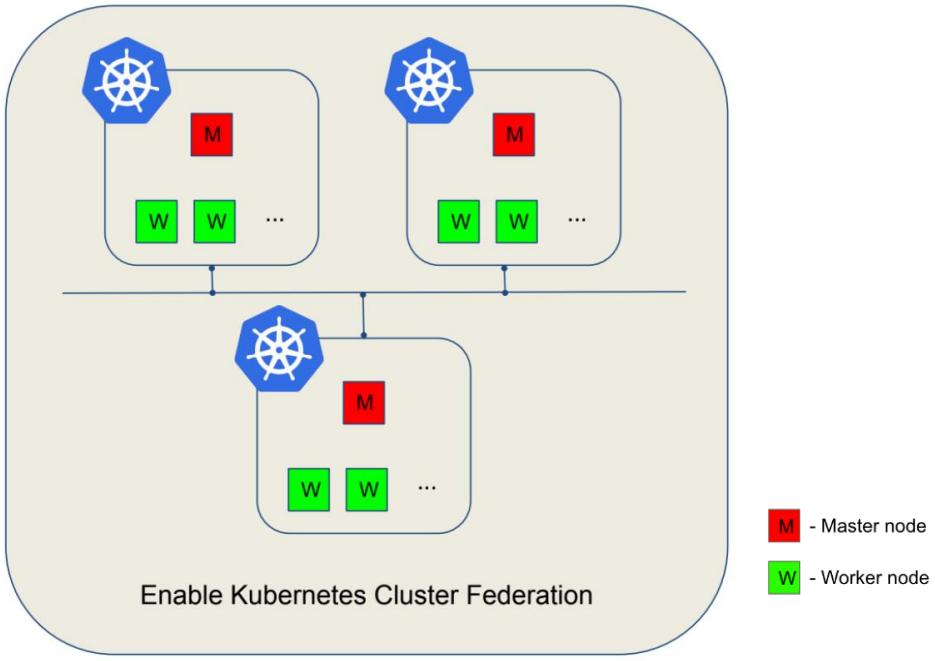
คิวเบอร์เนเทสคลัสเตอร์แบบหลายคลัสเตอร์ (Kubernetes multi-cluster) คือ คิวเบอร์เนเทส 2 คลัสเตอร์ขึ้นไป ที่อาจแบ่งภาระตามลักษณะงาน วัตถุประสงค์ การทดสอบ พื้นที่ ข้อตกลงทางกฎหมาย เป็นต้น

คิวเบอร์เนเทสคลัสเตอร์แบบหลายคลัสเตอร์ที่ร่วมสมาพันธ์ (Kubernetes cluster federation) คือ คิวเบอร์เนเทส 2 คลัสเตอร์ขึ้นไปที่มีปฏิสัมพันธ์กัน เช่น มีการย้ายภาระงานระหว่างกัน การขอใช้ทรัพยากรชั่วคราว การติดตั้งภาระงานควบ การตั้งเป็นศูนย์สำรอง เป็นต้น คิวเบอร์เนเทสคลัสเตอร์ที่ร่วมสมาพันธ์อาจมีผู้ดูแลระบบต่างกลุ่มหรือต่างหน่วยงานกัน มีประเด็นที่น่าสนใจ เช่น การจำกัดสิทธิการเข้าถึงทรัพยากรต่างกลุ่ม การหาเส้นทางระหว่างภาระงานต่างกลุ่ม กลยุทธ์การให้ยืมทรัพยากรต่างกลุ่ม ผู้ใช้งานกลางระหว่างหน่วยงาน เป็นต้น

โครงสร้างคิวเบอร์เนเทส อาจมีทั้งแบบที่ติดตั้งอยู่บนหลายผู้ให้บริการคลาวด์ แบบลูกผสม แบบมีผู้ดูแลระบบหลายกลุ่ม หรือแบบมีหลายคลัสเตอร์ก็ได้

2.5 การร่วมสมาพันธ์ของคิวเบอร์เนเทสคลัสเตอร์ (Kubernetes cluster federation)

การร่วมสมาพันธ์คิวเบอร์เนเทสคลัสเตอร์ มีแนวคิดและตัวอย่างเช่น คิวเฟ็ด (Kubefed) [23] เป็นกลไกที่ทำให้คิวเบอร์เนเทสคลัสเตอร์ทำงานร่วมกันได้ รูปที่ 12 แสดงการร่วมสมาพันธ์ของคิวเบอร์เนเทสคลัสเตอร์แบบคิวเฟ็ดที่มีการรวมระนาบควบคุมเข้าด้วยกัน



รูปที่ 12 การร่วมสมาพันธ์ของคิวเบอร์เนเทสคลัสเตอร์

การร่วมสมาพันธ์ของคิวเบอร์เนเทสคลัสเตอร์ มีองค์ประกอบสำคัญที่ควรพิจารณาดังนี้ [10]

1. มีการถ่ายทอดสิทธิ์การใช้งานของผู้ใช้งานแต่ละคลัสเตอร์
2. นโยบายที่กำหนดการเข้าถึงประเภททรัพยากรของคิวเบอร์เนเทส
3. การจัดการดีเอ็นเอส (dns) เพื่อให้เข้าถึงบริการ (service) ที่กระจายอยู่ได้
4. การจัดการเนมสเปซ (namespace) ภายในคลัสเตอร์ของคลัสเตอร์ร่วมสมาพันธ์

2.6 โครงการ IoTcloudServe@TEIN

เทคโนโลยีคอนเทนเนอร์และคอนเทนเนอร์ออร์เคสเตรเตอร์ได้นำมาใช้ออกแบบสถาปัตยกรรมระบบและติดตั้งบนโครงสร้างพื้นฐานจริงในโครงการ IoTcloudServe@TEIN (data-centric IoT-cloud service platform for smart communities) [27] ซึ่งเป็นโครงการความร่วมมือระหว่างประเทศ ในการสร้างแซนด์บอกซ์ (sandbox) เพื่อพัฒนาไอโอทีแพลทฟอร์ม โดยมีการศึกษาเทคโนโลยีคอนเทนเนอร์ และคิวเบอร์เนเทส ในการรวมเซิร์ฟเวอร์จำนวน 10 โนดเข้าด้วยกันเป็นคิวเบอร์เนเทสคลัสเตอร์ โดยติดตั้งอยู่ที่จุฬาลงกรณ์มหาวิทยาลัยจำนวน 3 โนด สำนักงานบริหารเทคโนโลยีเพื่อพัฒนาการศึกษา (Uninet) จำนวน 4 โนด ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) จำนวน 1 โนด มหาวิทยาลัยแห่งชาติลาว (National University of Laos) ประเทศลาวจำนวน 1 โนด และสถาบันเทคโนโลยีกัมพูชา (Institute of Technology Cambodia) ประเทศกัมพูชาจำนวน 1 โนด การออร์เคสเตรตคอนเทนเนอร์ในคลัสเตอร์สามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ มีการคอนเทนเนอร์ไรซ์ย้ายระบบงานเดิมและพัฒนาระบบงานใหม่ และจดทะเบียนโดเมนในชื่อ iotcloudserve.net ให้บริการต่อบุคลากรในโครงการ โดยมีการจัดการภาระงานและกลุ่มผู้ใช้งานผ่านซอฟท์แวร์รานเชอร์ ให้บริการเรื่อยมาจนถึงปัจจุบัน

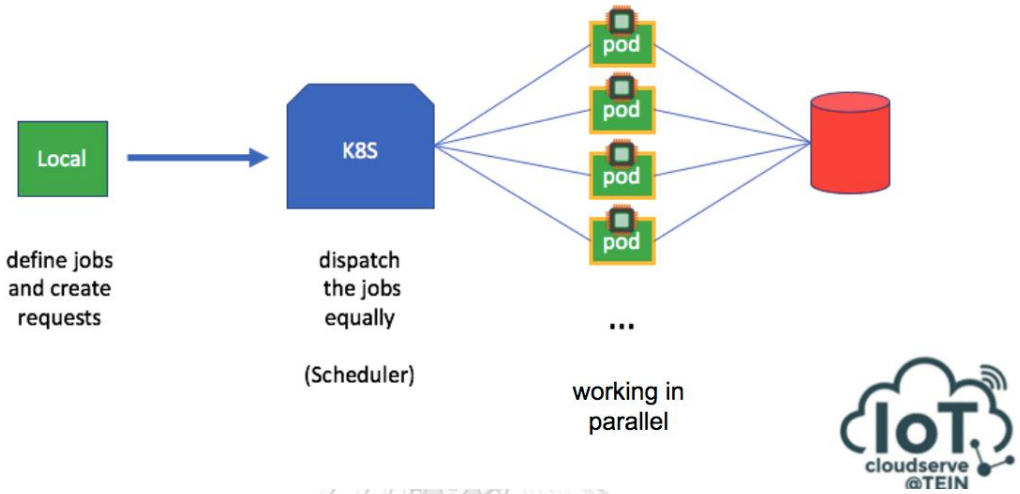
2.7 โครงการ CU-BEMS

จากโครงสร้างพื้นฐานที่มีการพัฒนาไอโอทีแพลทฟอร์มโดยการรวมศูนย์คอมพิวเตอร์เซิร์ฟเวอร์จากหลายภูมิภาคเข้าด้วยกัน จึงได้พัฒนากรณีศึกษาระบบจัดการพลังงานอัจฉริยะ โดยการพัฒนากรอบการทำงานเพื่อเพิ่มความเร็วการวิเคราะห์ข้อมูล ในบริบทนี้ผู้เขียนมีความประสงค์นำเสนอการใช้คุณสมบัติของคอนเทนเนอร์ในการเคลื่อนย้ายแอพลิเคชัน (portability) ที่สะดวกรวดเร็วและปรับเปลี่ยนกลยุทธ์การวิเคราะห์ข้อมูลได้ง่าย โดยมีกรณีศึกษาว่า หากมีทรัพยากรมากขึ้นสามารถเพิ่มความเร็วการวิเคราะห์ข้อมูล โดยใช้เทคนิคอัลกอริทึมการแบ่งแยกและเอาชนะได้

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย มีการพัฒนาต้นแบบระบบจัดการพลังงานในอาคาร (BEMS – building energy management system) [19] โดยการติดตั้งเซ็นเซอร์ และนำข้อมูลที่ได้ไปบันทึก วิเคราะห์และทำนายพฤติกรรมการใช้พลังงานในอนาคต จึงได้นำข้อมูลย้อนหลังมาพัฒนากรอบการทำงานการวิเคราะห์ข้อมูลพลังงานอัจฉริยะ บนโครงสร้างพื้นฐาน IoTcloudServe@TEIN แนวคิดการออกแบบมีดังนี้

จากการสืบค้นประวัติการใช้พลังงานย้อนหลัง พบว่ามีข้อมูลถูกบันทึกเป็นจำนวนมาก (ตั้งแต่เมษายนปี 2014 ถึง สิงหาคม 2017 ของ Sensor 588 ตัว รวมประมาณ 9GB) การคำนวณพลังงานสูญเสียจากข้อมูลทั้งหมดที่มีใช้เวลามาก จึงพิจารณาแนวทางการประมวลผลพร้อมกัน (parallel computing) เพื่อทดสอบประสิทธิภาพการออร์เคสเตรตคอนเทนเนอร์ ตามจำนวนหน่วยประมวลผลกลางที่มีในคลัสเตอร์

รูปที่ 13 แสดงแนวทางการวิเคราะห์ข้อมูลด้วยการประมวลผลพร้อมกันโดยการเขียนโปรแกรมจาวาสคริปต์ และกระจายการวิเคราะห์ไฟล์ CSV พบว่า ลดระยะเวลาการประมวลผลได้ตามจำนวนพอดที่มากขึ้นได้ แต่หากมีการกระจายงานเกินกว่าจำนวนหน่วยประมวลผลกลาง จะทำให้การวิเคราะห์ข้อมูลช้าลง [28]

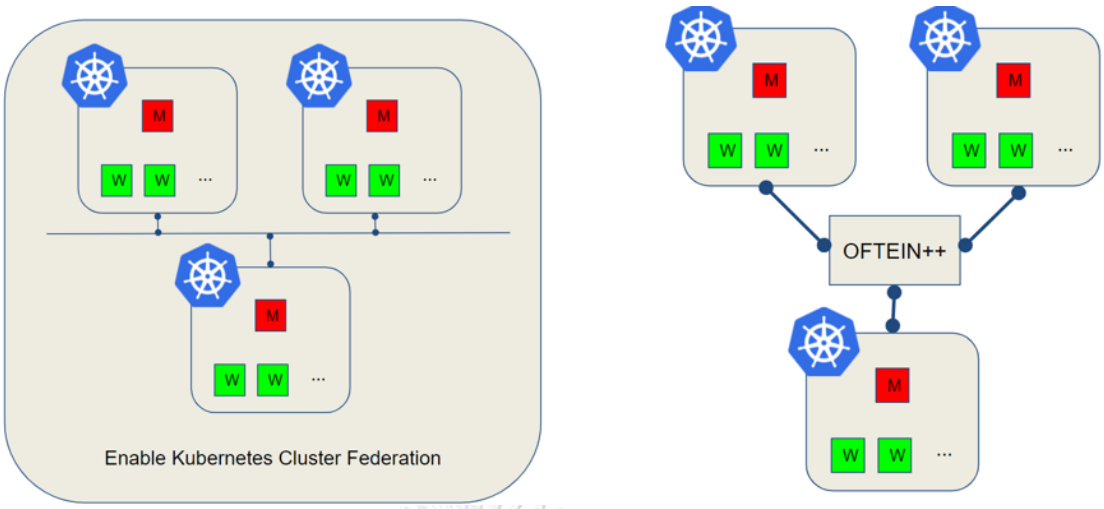


รูปที่ 13 แนวทางการวิเคราะห์ข้อมูลด้วยการประมวลผลพร้อมกัน

2.8 โครงการ OF@TIEN++

นอกเหนือจากสถาปัตยกรรมระบบแบบไอโอทีคลาวด์เซิร์ฟเดิม ได้นำเสนอสถาปัตยกรรมระบบที่บริหารจัดการเซิร์ฟเวอร์หลายโนดแบบรวมศูนย์ ด้วยเทคโนโลยีคอนเทนเนอร์และสร้างเป็นคิวเบอร์เนเทส 1 คลัสเตอร์จากกลุ่มผู้พัฒนาระบบกลุ่มเดียวกัน ปรากฏมีคิวเบอร์เนเทสคลัสเตอร์ต่างกลุ่ม โดยเป็นกลุ่มผู้พัฒนาจากมหาวิทยาลัย Gwangju Institute of Science and Technology จากประเทศเกาหลีใต้ และมหาวิทยาลัย University of Malaya จากประเทศมาเลเซีย จึงเกิดแนวคิดการพัฒนาแนวทางการใช้ทรัพยากรข้ามคลัสเตอร์ร่วมกัน

โครงการ OF@TEIN++ เป็นโครงการต่อยอดจากโครงการ IoTcloudServe@TEIN เพื่อควบรวมคิวเบอร์เนเทสคลัสเตอร์ของแต่ละหน่วยงานเข้าด้วยกัน ตามแนวทางการร่วมสมาพันธ์ (federation) โดยมีคลัสเตอร์ของมหาวิทยาลัย GIST จากประเทศเกาหลีใต้ คลัสเตอร์ของมหาวิทยาลัย UM จากประเทศมาเลเซีย และคลัสเตอร์ของจุฬาลงกรณ์มหาวิทยาลัย จากประเทศไทย โดยตัวแทนจุฬาฯ ได้ร่วมพัฒนากลไกการร่วมสมาพันธ์ดังกล่าว มีแนวคิดการออกแบบดังนี้



รูปที่ 14 การเปรียบเทียบแนวคิดการออกแบบวิธีการร่วมสมาพันธ์คิวเบอร์เนเทส

ปัญหาการจัดการคิวเบอร์เนเทสหลายคลัสเตอร์ในวิทยานิพนธ์นี้จะแตกต่างจากการกรณีการร่วมสมาพันธ์คิวเบอร์เนเทสทั่วไป โดยปกติการร่วมสมาพันธ์คิวเบอร์เนเทสจะบริหารระบบโดยผู้ดูแลระบบกลุ่มเดียว ที่มีการจัดการคิวเบอร์เนเทสหลายคลัสเตอร์ แต่ในกรณีของโครงการ OF@TEIN++ มีลักษณะเป็น**ผู้ดูแลระบบหลายกลุ่ม** และคำนึงถึงสิทธิในการควบคุม**ทรัพยากรที่จะร่วมสมาพันธ์** รูปที่ 14 ด้านซ้ายเป็นการร่วมสมาพันธ์คิวเบอร์เนเทสแบบคิวเฟ็ด [23] ซึ่งเป็นการรวมระนาบควบคุมเข้าด้วยกัน ส่วนด้านขวาเป็นต้นแบบระบบ OF@TEIN++ โครงสร้างดังกล่าวมีลักษณะดังนี้

1. มีการแยกโครงสร้างพื้นฐานออกจากกัน
2. มีการแยกกลุ่มผู้ใช้งานแต่ละคลัสเตอร์ และกลุ่มผู้ใช้งาน OF@TEIN++ ออกจากกัน
3. มีการรวมศูนย์การตัดสินใจการออร์เคสเตรชันของการใช้ทรัพยากรที่ร่วมสมาพันธ์
4. มีกลไกการเลือกคลัสเตอร์ที่จะออร์เคสเตรตคอนเทนเนอร์อัตโนมัติ
5. ผู้ดูแลระบบแต่ละคลัสเตอร์สามารถเปลี่ยนนโยบายการร่วมสมาพันธ์ได้ตลอด

บทที่ 3
การออกแบบสถาปัตยกรรมการเร่งความเร็วการวิเคราะห์ข้อมูล ด้วยการประมวลผลแบบขนานเชิงกระจาย

สถาปัตยกรรมระบบที่นำเสนอในบทนี้ได้ให้ความสำคัญในด้านสมรรถนะ (performance) โดยนำเสนอแนวทางการรีดสมรรถนะจากการควบรวมทรัพยากรในการวิเคราะห์ข้อมูลขนาดใหญ่ เช่น การวิเคราะห์ไฟล์ประเภท CSV โดยใช้คุณสมบัติของคอนเทนเนอร์และความสามารถในการยืดหดทรัพยากร ที่สามารถเร่งความเร็วการประมวลผลมากขึ้นได้ตามจำนวนทรัพยากรที่มี ในหัวข้อ 3.1 จะนำเสนอรูปแบบของข้อมูลดิบที่ได้จากเซนเซอร์ของโครงการ CU-BEMS ส่วนในหัวข้อ 3.2 จะเป็นการออกแบบโปรแกรมวิเคราะห์ข้อมูล และในหัวข้อ 3.3 จะเป็นการนำเสนอสถาปัตยกรรมระบบที่ออกแบบ

3.1 รูปแบบข้อมูลดิบจากเซนเซอร์ของโครงการ CU-BEMS

โครงการ CU-BEMS มีการพัฒนากล่องรวมเซนเซอร์พิเศษที่สามารถวัดข้อมูลหลายอย่าง ได้แก่ อุณหภูมิ ความชื้น ความสว่าง และเซนเซอร์ตรวจจับความเคลื่อนไหว PIR (passive infrared) การเก็บข้อมูลดังกล่าวเพียงพอต่อการวิเคราะห์การประมาณพลังงานสูญเสียในช่วงเวลาหนึ่งได้ แนวคิดคือหากไม่มีความเคลื่อนไหวภายในห้องในช่วงระยะเวลาหนึ่ง แต่มีการบริโภคพลังงาน จะนับว่าเป็นพลังงานสูญเสีย [29]

19.csv

```
1  TIME,VALUE
2  2014-04-15 20:40:16,0.0
3  2014-04-15 20:40:45,1.0
4  2014-04-15 20:41:14,1.0
5  2014-04-15 20:41:43,0.0
6  2014-04-15 20:42:13,0.0
7  2014-04-15 20:42:42,0.0
8  2014-04-15 20:43:11,0.0
9  2014-04-15 20:43:41,0.0
10 2014-04-15 20:44:10,0.0
11 2014-04-15 20:44:40,0.0
12 2014-04-15 20:45:09,0.0
13 2014-04-15 20:45:39,0.0
14 2014-04-15 20:46:08,0.0
```

Possible Values	Person in room
0.0 or OFF	False
1.0 or ON	True

รูปที่ 15 ตัวอย่างข้อมูลที่บันทึกได้จากเซนเซอร์ PIR

รูปที่ 15 แสดงข้อมูลจากเซนเซอร์ PIR ซึ่งปรากฏมีค่าที่เป็นไปได้ 4 แบบคือ 0.0, 1.0, OFF, และ ON โดยแปลผลได้ดังต่อไปนี้ ถ้าเป็น 0.0 หรือ OFF หมายถึงไม่มีความเคลื่อนไหวในโซนนั้น และ 1.0 หรือ ON หมายถึงมีความเคลื่อนไหวในโซนนั้น จากรูปมุมซ้ายบนแสดงชื่อของไฟล์ซึ่งเป็นหมายเลขไอดีของชุดข้อมูล และนำไปเทียบกับข้อมูลการบริโภคพลังงานของไฟล์ CSV ในโซนเดียวกัน เช่นตัวอย่างข้อมูลจากสถานที่ ตึก 4 ชั้น 13 โซน z1 มีการบันทึกข้อมูลการใช้พลังงานในไฟล์ 0.csv

1.csv 2.csv เป็นการใช้ไฟโดยเครื่องปรับอากาศ 3.csv เป็นการใช้ไฟส่องสว่าง 4.csv เป็นการใช้ไฟของเต้าเสียบ และไฟล์ 7.csv เป็นข้อมูลเซนเซอร์ PIR วัดความเคลื่อนไหว ตารางที่ 1 แสดงรายการตัวอย่างข้อมูลชื่อไฟล์และอุปกรณ์ที่ถูกเก็บข้อมูล

ตารางที่ 1 ตัวอย่างข้อมูลชื่อไฟล์และอุปกรณ์ที่ถูกเก็บข้อมูล

ชื่อไฟล์	อุปกรณ์	ชนิด
0.csv	aircon_3ph1	energy_r
1.csv	aircon_3ph1	energy_s
2.csv	aircon_3ph1	energy_t
3.csv	light1	energy
4.csv	outlet1	energy
7.csv	sensor1	pir

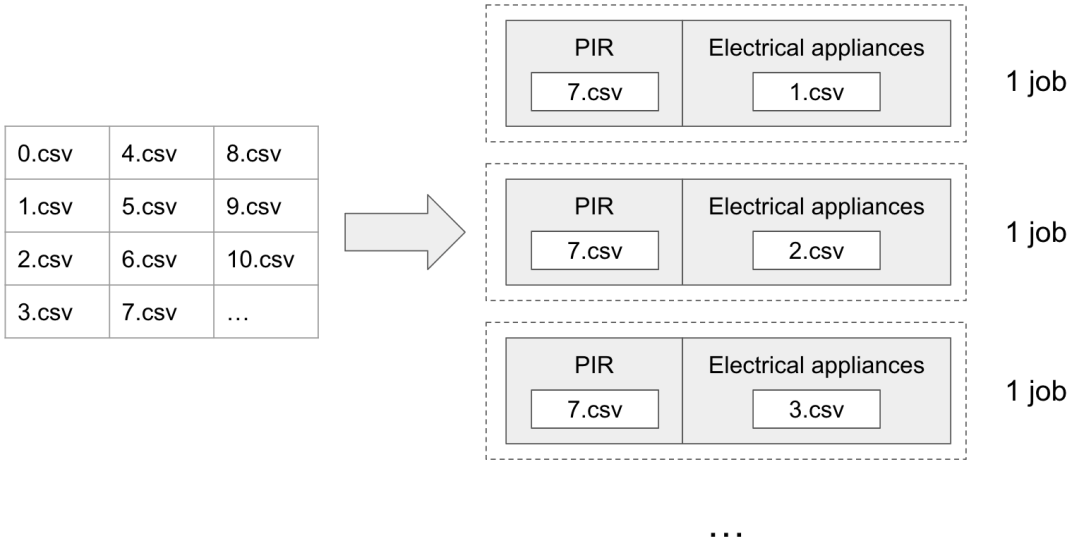
3.2 การออกแบบโปรแกรมวิเคราะห์ข้อมูล CU-BEMS

ข้อมูลของแต่ละไฟล์ในช่วงเวลาเดียวกัน จะนำมาเปรียบเทียบกันดังตัวอย่างตารางที่ 2 ในการเปรียบเทียบจะตัดเวลาหลักวินาทีออก ให้เปรียบเทียบในหลักนาที การประเมินว่า มีการใช้พลังงานสูญเสียหรือไม่ จะประเมินจากหากในช่วงเวลา 15 นาทีที่ผ่านมาไม่มีความเคลื่อนไหว จะประเมินว่าในช่วงเวลาขณะนั้น เป็นการใช้พลังงานสูญเสีย เช่น จากตารางที่ 2 ช่วงเวลาวันที่ 29 พฤษภาคม 2014 เวลา 10.33 น. ตรวจสอบพบความเคลื่อนไหวใต้เซนเซอร์ PIR มีการใช้เครื่องปรับอากาศ และไฟส่องสว่างในช่วงเวลาเดียวกันที่ตรวจพบความเคลื่อนไหว ในช่วงเวลาจากนี้ 15 นาทีเป็นต้นไป จะไม่นับเป็นพลังงานสูญเสีย

ตารางที่ 2 ตัวอย่างข้อมูล Pointid ในช่วงเวลาเดียวกัน

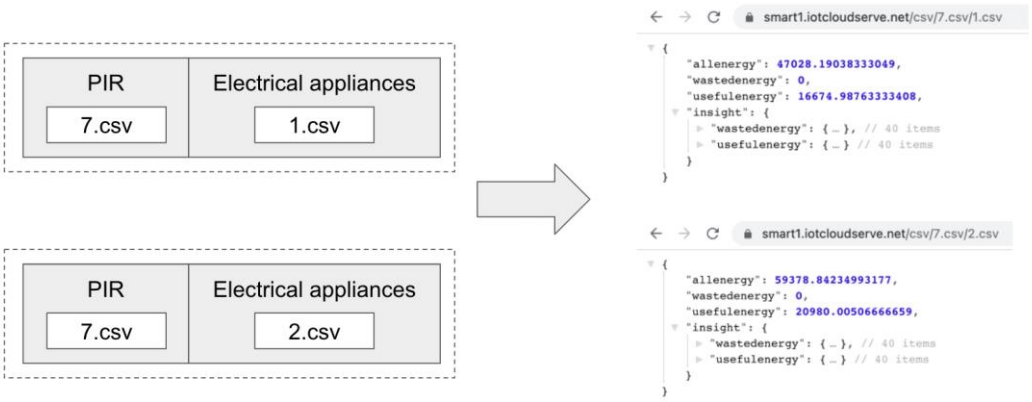
PIR (point id 7)	Aircon (point id 1)	Light (point id 3)
2014-05-29 10:33:49,1.0	2014-05-29 10:33:00,14.299	2014-05-29 10:33:00,9.468
2014-05-29 10:34:07,0.0	2014-05-29 10:34:00,15.396	2014-05-29 10:34:00,9.468
2014-05-29 10:35:17,0.0	2014-05-29 10:35:00,13.543	2014-05-29 10:35:00,9.347
...	...	...

จากรายการข้อมูลทั้งหมดจากตัวอย่างในตารางที่ 2 จึงนำมาออกแบบการกระจายงาน โดยข้อมูลจากในโซนเดียวกันจะมีข้อมูล PIR และอุปกรณ์ใช้ไฟฟ้าอื่น ๆ เช่น เครื่องปรับอากาศ หลอดไฟ เต้าเสียบ รูปที่ 16 แสดงการเชื่อมโยงข้อมูล PIR และเครื่องใช้ไฟฟ้าในโซนเดียวกัน จัดเป็นชุดของงานในการคำนวณพลังงานสูญเสีย



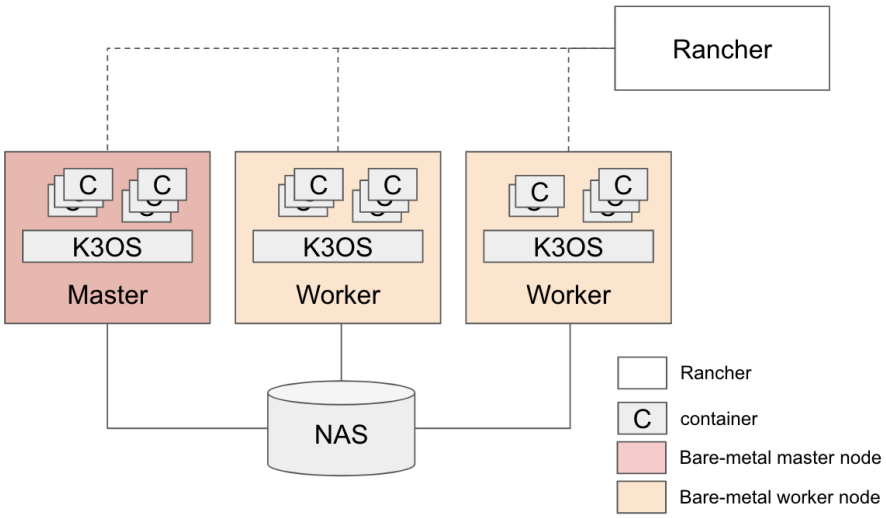
รูปที่ 16 การเชื่อมโยงข้อมูล PIR และเครื่องใช้ไฟฟ้าในโซนเดียวกัน

ใน 1 งานประกอบด้วยการอ่านไฟล์ CSV ข้อมูล PIR ที่ตรวจจับความเคลื่อนไหว และการอ่านไฟล์ CSV ข้อมูลการใช้พลังงานของเครื่องใช้ไฟฟ้าในโซนเดียวกัน จึงเขียนโปรแกรมรับข้อมูลนำเข้าเป็นชื่อไฟล์ CSV ที่เป็น PIR และชื่อไฟล์ CSV ที่เป็นเครื่องใช้ไฟฟ้านำมาเทียบกันและคำนวณการใช้พลังงานแบ่งเป็นพลังงานสูญเสีย และพลังงานที่ใช้ทั้งหมด ดังโค้ดอ่านไฟล์ CSV ในภาคผนวก การคำนวณโดยปกติจะคำนวณทีละ 1 คู่ไฟล์ไปจนครบจำนวนคู่ของรายการไฟล์ CSV จำนวนทั้งหมด 584 รายการ จึงพัฒนาสถาปัตยกรรมระบบโดยใช้เทคโนโลยีคอนเทนเนอร์ให้สามารถประมวลผลหลายงานพร้อมกัน รูปที่ 17 แสดงตัวอย่างผลลัพธ์เมื่อให้โปรแกรมคำนวณพลังงานสูญเสียทีละ 1 คู่ไฟล์จำนวน 2 งาน



รูปที่ 17 ตัวอย่างผลลัพธ์การคำนวณพลังงานสูญเสีย 2 งานพร้อมกัน

3.3 สถาปัตยกรรมระบบที่นำเสนอ



รูปที่ 18 สถาปัตยกรรมระบบแบบ 1 คลัสเตอร์เพื่อเร่งการประมวลผลแบบพร้อมกัน

สถาปัตยกรรมระบบต้องการนำเสนอการใช้เทคโนโลยีคอนเทนเนอร์ และติดตั้งออร์เคสเตรเตอร์ช่วยกระจายภาระงานไปทุกโนดในคลัสเตอร์ วิทยานิพนธ์นี้ได้เลือกติดตั้งออร์เคสเตรเตอร์เป็นคิวเบอร์เนเทสบนคอมพิวเตอร์เซิร์ฟเวอร์ 3 โนด และเชื่อมต่อกับแนส (NAS คืออุปกรณ์จัดเก็บข้อมูล) ในการแชร์ข้อมูลร่วมกัน รูปที่ 18 แสดงสถาปัตยกรรมระบบ ประกอบด้วยคิวเบอร์เนเทสมาสเตอร์ 1 โนด และคิวเบอร์เนเทสเวิร์คเกอร์ 2 โนด การติดตั้งระบบปฏิบัติการคอนเทนเนอร์เอนจิน และทรัพยากรที่รองรับเป็นดังตารางที่ 3 ซึ่งแสดงทรัพยากรและการตั้งค่าเครือข่ายของแต่ละโนดโดยรายละเอียดและวิธีการติดตั้ง ได้แสดงวิธีการออกแบบและขั้นตอนการติดตั้งโดยละเอียดในภาคผนวก

ตารางที่ 3 การติดตั้งระบบปฏิบัติการ คอนเทนเนอร์เอนจิน และการตั้งค่าเครือข่ายตาม สถาปัตยกรรมระบบ

Hostname	OS	Architecture	Role	Container engine	Orchestrator
master1	K3OS	amd64	Master	containerd	K3s
worker1	K3OS	amd64	Worker	containerd	K3s
worker2	K3OS	amd64	Worker	containerd	K3s

Hostname	CPU	Memory	Network interface
master1	32 cores	251 GB	ethernet_ac1f6bbb9de8_cable
worker1	32 cores	251 GB	ethernet_ac1f6bbb9b34_cable
worker2	32 cores	251 GB	ethernet_ac1f6bbb9f4c_cable

Hostname	IP-Address	Subnet mask	Gateway IP
master1	202.28.193.102	255.255.255.224	202.28.193.98
worker1	202.28.193.100	255.255.255.224	202.28.193.98
worker2	202.28.193.101	255.255.255.224	202.28.193.98

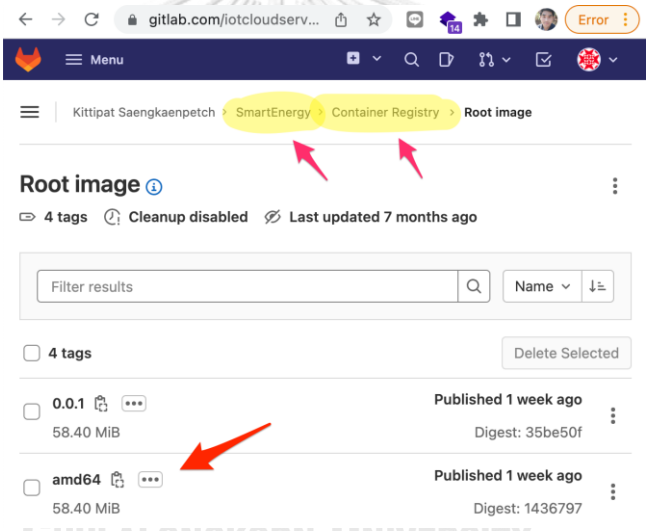
3.4 การทดสอบสถาปัตยกรรมระบบ

3.4.1 การห่อโปรแกรมวิเคราะห์ข้อมูล CU-BEMS

โปรแกรมที่ได้ในหัวข้อ 3.2 จะกำหนดสคริปต์เพื่อการทำคอนเทนเนอร์ไรเซชันตามแนวทางในหัวข้อ 2.1 และรันคอนเทนเนอร์บนคิวเบอร์เนเทสตามสถาปัตยกรรมระบบที่ออกแบบ การทำคอนเทนเนอร์ไรเซชันทำโดยการกำหนดสคริปการคอนเทนเนอร์ไรซ์ดังตารางที่ 4 และสร้างคอนเทนเนอร์อิมเมจบันทึกบนเครื่องท้องถิ่น จากนั้นจึงแชร์คอนเทนเนอร์อิมเมจไปบน registry.gitlab.com/iotcloudserve/smartenergy หรือในระดับโกลบอล เพื่อให้คิวเบอร์เนเทสสามารถดึงคอนเทนเนอร์มารันในคลัสเตอร์ได้ ผลการอัพโหลดคอนเทนเนอร์อิมเมจไป gitlab.com แสดงดังรูปที่ 19

ตารางที่ 4 สคริปดอกเกอร์ไฟล์กำหนดขั้นตอนการคอนเทนเนอร์ไรซ์

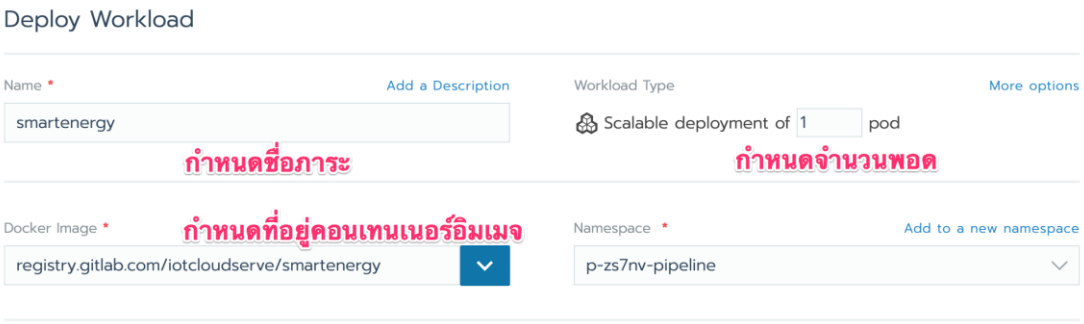
คำสั่งในสคริปต์	คำอธิบาย
FROM node:12-slim	กำหนดคอนเทนเนอร์ฐาน
WORKDIR /usr/src/app	กำหนดไดเรคทอรีที่ทำงานภายในคอนเทนเนอร์
COPY package*.json ./	คัดลอกแพคเกจไลบรารีที่โปรแกรมใช้
RUN npm install	ติดตั้งไลบรารีตามแพคเกจในคอนเทนเนอร์
EXPOSE 8080	เปิดพอร์ต 8080 บนคอนเทนเนอร์
COPY . .	คัดลอกโปรแกรมทั้งหมดลงคอนเทนเนอร์
CMD ["node", "index.js"]	รันโปรแกรมบนคอนเทนเนอร์



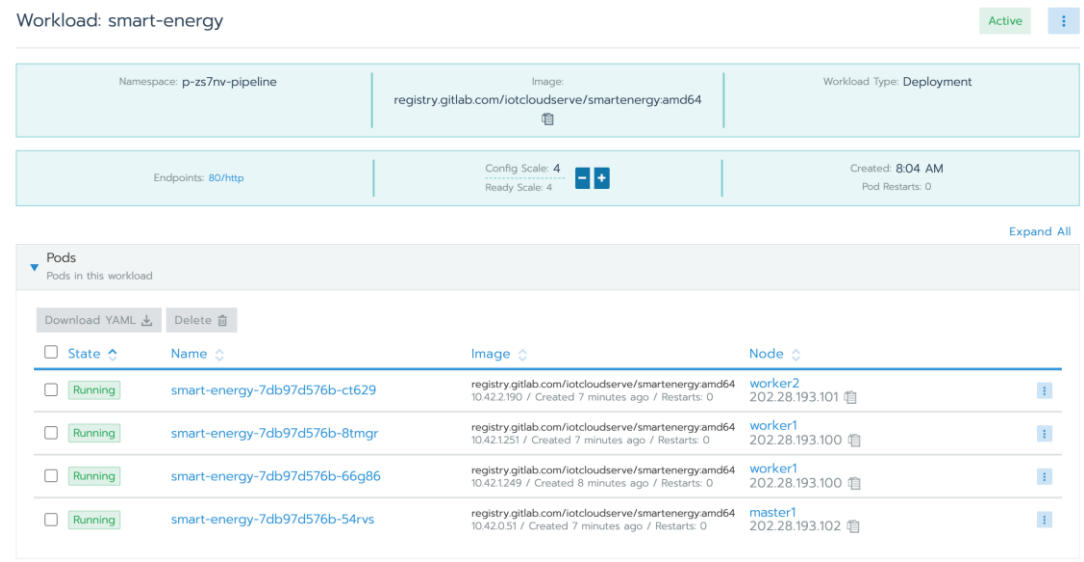
รูปที่ 19 คอนเทนเนอร์อิมเมจบน gitlab.com เพื่อจัดเก็บโปรแกรมวิเคราะห์ข้อมูล CU-BEMS

3.4.2 การทดสอบการรันคอนเทนเนอร์บนคิวเบอร์เนเทส

ทดสอบการรันคอนเทนเนอร์บนคิวเบอร์เนเทสผ่านโปรแกรมรานเชอร์ โดยสร้างภาระบนรานเชอร์ ดังรูปที่ 20 จากนั้นเพื่อทดสอบว่าคอนเทนเนอร์สามารถกระจายการรันไปได้ทุกโนดในคลัสเตอร์ จึงเพิ่มจำนวนพอดของคอนเทนเนอร์เป็น 4 คอนเทนเนอร์ให้กระจายบนโนดในคลัสเตอร์ที่มี 3 โนด ผลการทดสอบการรันคอนเทนเนอร์กระจายไปในคลัสเตอร์เป็นดังรูปที่ 21



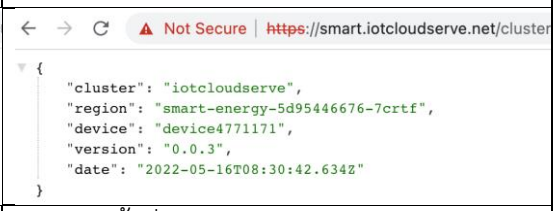
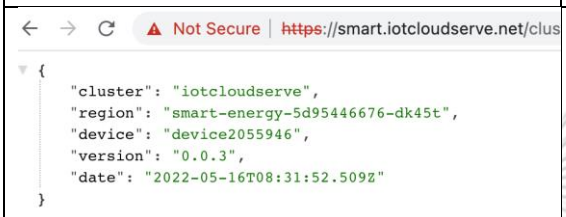
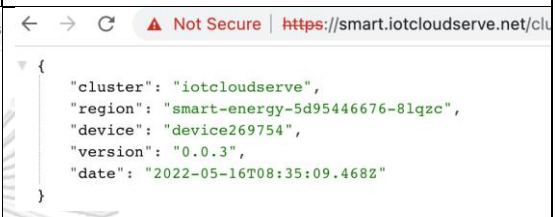
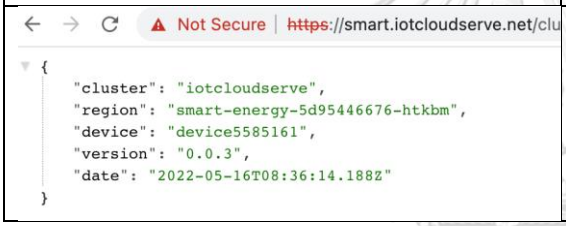
รูปที่ 20 การสร้างภาระบนคิวเบอร์เนเทส



รูปที่ 21 การกระจายคอนเทนเนอร์ไปรันแบบหลายโนดในคลัสเตอร์

รูปที่ 21 แสดงการกระจายคอนเทนเนอร์ไปรันที่มาสเตอร์โนดและเวิร์คเกอร์โนด 2 โนดได้ครบถ้วน จากนั้นจึงสร้าง อินเกรซ เพื่อสร้างการเข้าถึงคอนเทนเนอร์จากภายนอก โดยกำหนดที่อยู่ https://smart.iotcloudserve.net บน DNS ให้ส่งต่อคำร้องมาที่ไอพีของเครื่องมาสเตอร์โนด โดยอินเกรซจะสร้างกลไกการดุลภาระ (load balancing) อัตโนมัติ ผลการทดสอบการเข้าถึงคอนเทนเนอร์แสดงดังตารางที่ 5 โดยเป็นการแสดงผลตอบกลับของแต่ละคอนเทนเนอร์ในคลัสเตอร์ โดยมีชื่อของพอด แสดงการดุลภาระของคิวเบอร์เนเทสว่าในแต่ละครั้งที่เข้าถึงคอนเทนเนอร์ คิวเบอร์เนเทสจะส่งคำร้องวนรอบกระจายไปทั้ง 4 คอนเทนเนอร์ จนเมื่อครบทั้ง 4 คอนเทนเนอร์ ก็จะเวียนกลับมาคอนเทนเนอร์ที่ 1 ใหม่ตามลำดับ

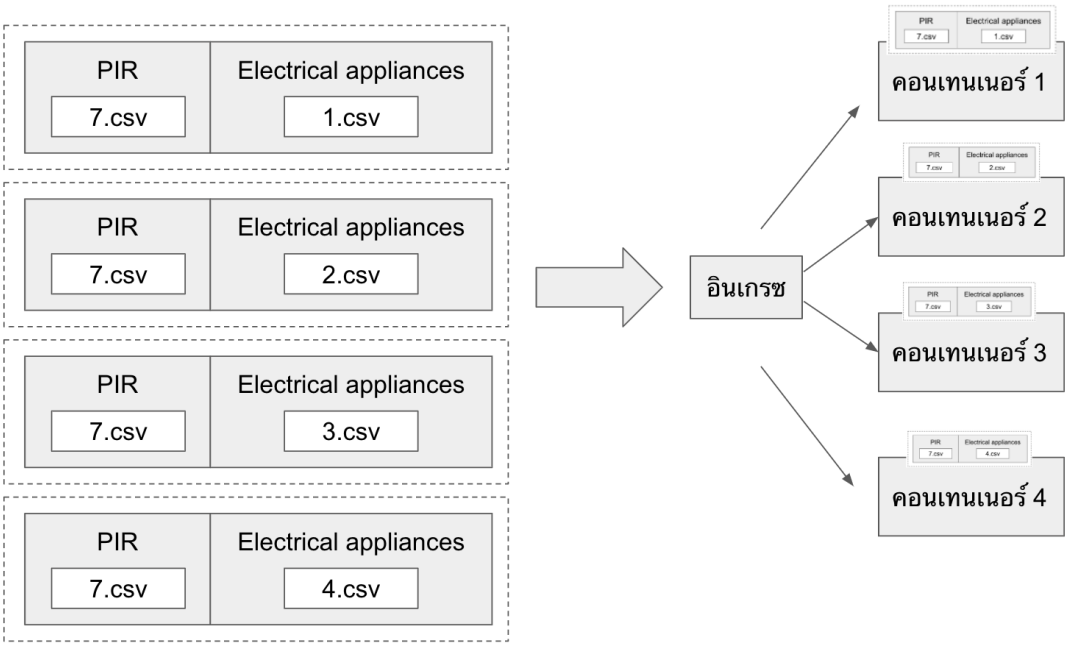
ตารางที่ 5 พฤติกรรมดุลภาระของอินเกรซบนคิวเบอร์เนเทส

<table>
<tr><td>การเข้าครั้งที่ 1 เป็นพอด device5585161</td><td>การเข้าครั้งที่ 2 เป็นพอด device4771171</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>การเข้าครั้งที่ 3 เป็นพอด device2055946</td><td>การเข้าครั้งที่ 4 เป็นพอด device269754</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>การเข้าครั้งที่ 5 เป็นพอด device5585161</td><td>การเข้าครั้งที่ 6 เป็นพอด device4771171</td></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

ตารางที่ 5 แสดงพฤติกรรมดุลภาระของอินเกรซบนคิวเบอร์เนเทส จากคอนเทนเนอร์จำนวน 4 คอนเทนเนอร์ที่กระจายไป 3 โนดในคลัสเตอร์ ผลการทดสอบอธิบายได้ดังนี้ เมื่อเข้าอินเกรซครั้งแรก อินเกรซจะส่งคำร้องไปพอดรหัส device5585161 จากนั้นเมื่อเข้าอินเกรซครั้งที่ 2 อินเกรซได้ส่งคำร้องไปพอดรหัส device4771171 จากนั้นในครั้งถัดไป อินเกรซได้ส่งคำร้องไป device2055946 device269754 device5585161 และ device4771171 ตามลำดับ บนคลัสเตอร์ที่รันคอนเทนเนอร์อิมเมจเดียวกันจำนวน 4 คอนเทนเนอร์ จากการเข้าอินเกรซในครั้งที่ 5 และ 6 อินเกรซได้ดุลภาระวนกลับมาเข้าถึง device5585161 และ device4771171 ตามลำดับที่เข้าอินเกรซในครั้งที่ 1 และ 2 ผลการทดสอบคอนเทนเนอร์ที่สร้างสามารถรันและเข้าถึงคอนเทนเนอร์กระจายไปทุกโนดในคลัสเตอร์ได้

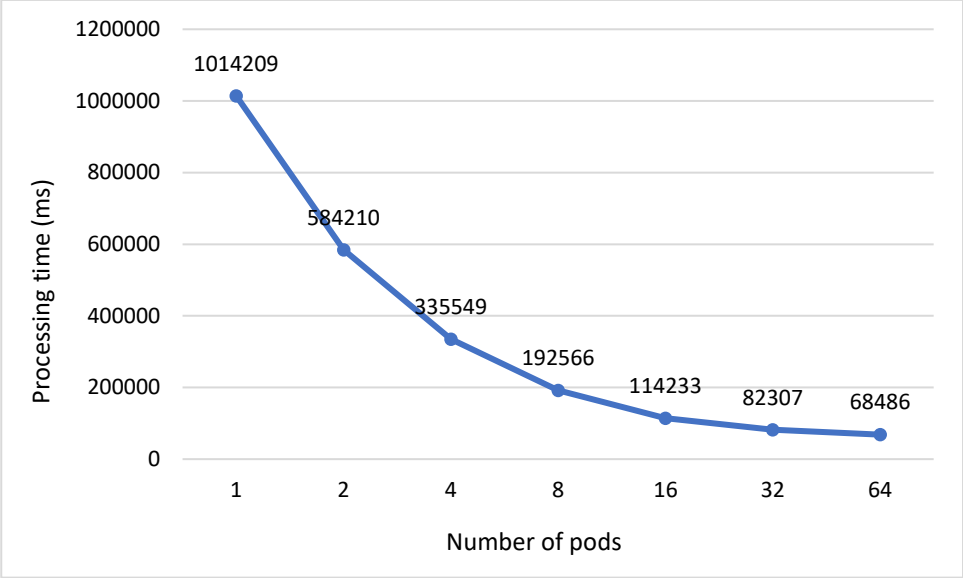
3.4.3 การเร่งความเร็วการวิเคราะห์ข้อมูลพลังงานอัจฉริยะ

จากกลไกอินเกรซการดุลภาระไปคอนเทนเนอร์ต่าง ๆ ในคลัสเตอร์อย่างทั่วถึงอัตโนมัติ จึงออกแบบการเร่งความเร็วการวิเคราะห์ข้อมูลโดย เมื่อส่งข้อมูลนำเข้าให้คอนเทนเนอร์อ่านไฟล์ CSV ของ PIR และเครื่องใช้ไฟฟ้า ในแต่ละครั้งที่เข้าถึงอินเกรซจะสามารถกระจายภาระไปคอนเทนเนอร์ต่าง ๆ ได้ให้ทำงานพร้อม ๆ กัน จึงเขียนโปรแกรมการสร้างภาระที่ยิงไปอินเกรซ แนวคิดเป็นไปดังรูปที่ 22 ได้สร้างโปรแกรมเพื่อสรุปงานระหว่างไฟล์ PIR และเครื่องใช้ไฟฟ้าในโซนเดียวกันเป็นชุด และกระจายงานตามจำนวนคอนเทนเนอร์ให้ประมวลผลพร้อมกัน โปรแกรมจะสรุปงานไฟล์ CSV ของ PIR และเครื่องใช้ไฟฟ้าในโซนเดียวกันเตรียมไว้เป็นคู่ และสร้างคำร้องไปอินเกรซที่บอกชื่อไฟล์ CSV ทั้ง 2 ไฟล์ให้โปรแกรมวิเคราะห์ เมื่อโปรแกรมคำนวณเสร็จสิ้น จึงบันทึกผลลัพธ์ไว้ และส่งคำร้องประมวลผลงานเพิ่มเติม ในท้ายที่สุด ผลลัพธ์แต่ละงานของโปรแกรม จะรวบรวมคำตอบย่อยของทุกงานเป็นคำตอบใหญ่เพียงคำตอบเดียว



รูปที่ 22 แนวคิดการประมวลผลข้อมูล CU-BEMS พร้อมกันบนคิวเบอร์เนเทส

3.4.4 ผลการทดสอบการเร่งความเร็วการวิเคราะห์ข้อมูล CU-BEMS



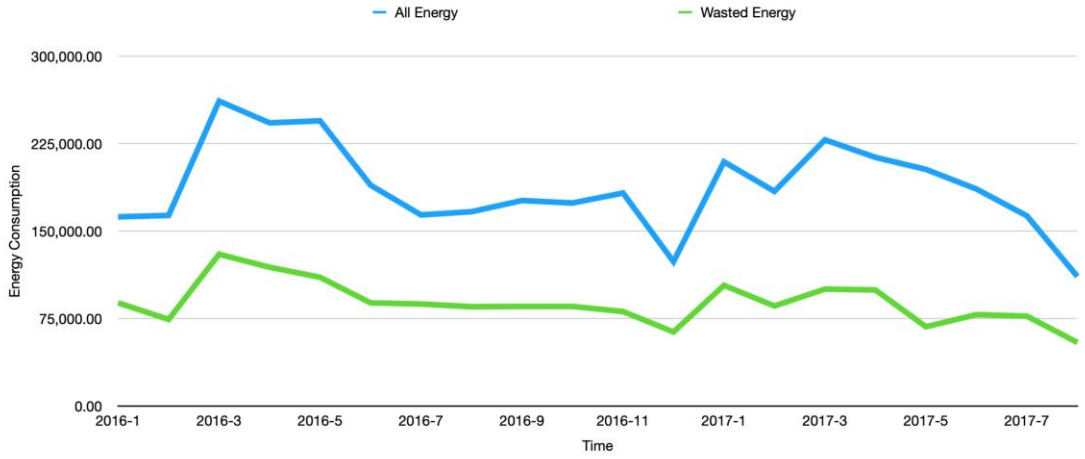
รูปที่ 23 ระยะเวลาที่ใช้ประมวลผลเทียบกับจำนวนพอด

ผลการทดสอบแสดงดังรูปที่ 23 แสดงระยะเวลาที่ใช้ประมวลผลในแกน y เทียบกับจำนวนพอดในแกน x (เทียบเท่ากับจำนวนคอนเทนเนอร์) พอดประมวลผลจำนวน 1 พอด ใช้เวลาในการวิเคราะห์ข้อมูลประมาณ 1,014,209 ms หรือประมาณ 17 นาที นับเป็นกรณีที่ไม่มีการประมวลผลพร้อมกัน

เมื่อเพิ่มจำนวนพอดเป็นจำนวน 2 พอด จะใช้เวลาประมวลผลลดลงเหลือประมาณ 9 นาทีครึ่ง และหากเพิ่มจำนวนพอดอีกเท่าตัวเป็น 4 พอด จะใช้เวลาประมวลผลลดลงเหลือ 6 นาทีตามลำดับ ผลการทดสอบเมื่อจำนวนพอดมากขึ้น จะสามารถประมวลผลข้อมูลทั้งหมดได้เร็วขึ้น กรณีวิเคราะห์สาเหตุที่ระยะเวลาประมวลผลที่ลดลงไม่เป็นครึ่งหนึ่งพอดีของจำนวนพอดที่มากขึ้นทีละเท่าตัว เนื่องจากกำหนดให้มีการรอกันระหว่างรอบการประมวลผลพร้อมกัน ซึ่งในบางคอนเทนเนอร์ได้รับงานชุดข้อมูลขนาดเล็กและใหญ่ไม่เท่ากัน การกระจายงานไปพอดที่ประมวลผลไม่เสร็จจะทำให้การประมวลผลช้าลงเพราะเกิดการบล็อคกันของโปรแกรมจาวาสคริปต์ [28] จึงกำหนดให้แต่ละรอบมีการประมวลผลจนสำเร็จทุกพอด จึงกระจายงานต่อในรอบถัดไป ทำให้พอดที่ทำงานเสร็จเร็วว่างงานจนกว่าจะรับงานใหม่ รายละเอียดโค้ดโปรแกรมอยู่ในภาคผนวก

นอกจากนี้ได้แก้ไขโปรแกรมการวิเคราะห์ข้อมูล ให้วิเคราะห์และจัดกลุ่มข้อมูลเพื่อศึกษาแนวโน้มการใช้พลังงานในแต่ละเดือน การพัฒนาโปรแกรมใช้เวลาประมาณครึ่งชั่วโมงและปรับเทคนิคการวิเคราะห์ข้อมูลใหม่ ดำเนินการคอนเทนเนอร์ไรซ์เซชัน และกระจายคอนเทนเนอร์ใหม่แทนที่คอนเทนเนอร์เดิม การวิเคราะห์ข้อมูลใช้เวลาใกล้เคียงกับการคำนวณพลังงานสูญเสีย โดยใช้เวลาประมวลผลน้อยลงเมื่อจำนวนพอดมากขึ้นเช่นกัน

รูปที่ 24 แสดงผลการวิเคราะห์ข้อมูลแนวโน้มการบริโภคพลังงานทั้งหมดเทียบกับพลังงานสูญเสีย แกน y เป็นข้อมูลการใช้พลังงานในหน่วย Kw/h แกน x เป็นเวลา เส้นสีน้ำเงินแสดงการใช้พลังงานทั้งหมด เทียบกับเส้นสีเขียวที่แสดงการใช้พลังงานสูญเสีย มีข้อสังเกตคือในช่วงระหว่างเดือนกุมภาพันธ์ถึงมีนาคมปี 2016 มีการบริโภคพลังงานสูงขึ้น และจากมีนาคมถึงมิถุนายนปี 2016 มีปริมาณพลังงานสูญเสียลดลงอย่างต่อเนื่อง



รูปที่ 24 แนวโน้มการบริโภคพลังงานทั้งหมดเทียบกับพลังงานสูญเสีย

3.5 บทสรุป

ข้อมูลไฟล์ CSV จำนวนมากจาก CU-BEMS สามารถถูกนำมาประมวลผลแบบเร่งความเร็วการประมวลผลได้ด้วยการเพิ่มจำนวนพอดที่รันคอนเทนเนอร์ให้มากขึ้น และมากที่สุดได้ตามจำนวนทรัพยากรที่มีของทุกโนดในคลัสเตอร์รวมกัน แนวทางการวิเคราะห์ข้อมูล CSV ด้วยวิธีการบรรจุโปรแกรมในคอนเทนเนอร์หลายคอนเทนเนอร์ สามารถกระจายงานและทำงานพร้อมกันได้ โดยสามารถเปลี่ยนสูตรการวิเคราะห์ข้อมูลได้ง่ายด้วยการคอนเทนเนอร์ไรซ์โปรแกรมและรันคอนเทนเนอร์บนพอดใหม่ ข้อดีของการวิเคราะห์ข้อมูลในลักษณะนี้อีกด้านหนึ่งคือ สามารถทำลายภาระงานที่ใช้เสร็จและคืนทรัพยากรที่ใช้ได้ทันที ที่ต่างจากกรณีการติดตั้งซอฟต์แวร์การจัดการข้อมูลขนาดใหญ่ Hadoop ในหัวข้อ 1.2 ที่ต้องสงวนทรัพยากรให้กับซอฟต์แวร์ตลอดระยะเวลาการใช้งาน

บทที่ 4
การออกแบบสถาปัตยกรรมระบบในการจัดการรุ่นของคอนเทนเนอร์

สถาปัตยกรรมระบบในบทนี้ให้ความสำคัญเรื่องการรันคอนเทนเนอร์ (configuration) ให้สามารถบริหารจัดการรุ่นของคอนเทนเนอร์แบบรวมศูนย์และสามารถรันคอนเทนเนอร์ไปโนดในพื้นที่ห่างไกลได้ ปัญหาที่พิจารณาในการจัดการคอนเทนเนอร์และรุ่นของคอนเทนเนอร์ในงานบำรุงรักษามีดังนี้

1. โนดเซิร์ฟเวอร์กระจายอยู่ในพื้นที่ห่างไกลจากมาสเตอร์โนด
2. คอนเทนเนอร์ที่รันในโนดเซิร์ฟเวอร์สามารถมีการย้อนกลับรุ่นของคอนเทนเนอร์ได้

4.1 การกำหนดแท็กของคอนเทนเนอร์อิมเมจ

คอนเทนเนอร์สามารถถูกกำหนดแท็ก (tag) เพื่อประโยชน์ในการรันคอนเทนเนอร์เฉพาะรุ่นหรือลักษณะเฉพาะอย่างอื่น แท็กของคอนเทนเนอร์จะถูกกำหนดในระหว่างก่อนการสร้างคอนเทนเนอร์เพื่อสร้างคอนเทนเนอร์อิมเมจใหม่บนทะเบียนคอนเทนเนอร์ จากนั้นเมื่อนำมาใช้จึงให้กำหนดก่อนรันคอนเทนเนอร์เพื่อเจาะจงรุ่นหรือลักษณะเฉพาะของคอนเทนเนอร์นั้น ๆ ตารางที่ 6 แสดงตัวอย่างรายการคอนเทนเนอร์อิมเมจและการกำหนดแท็กที่สื่อความหมายต่างกัน

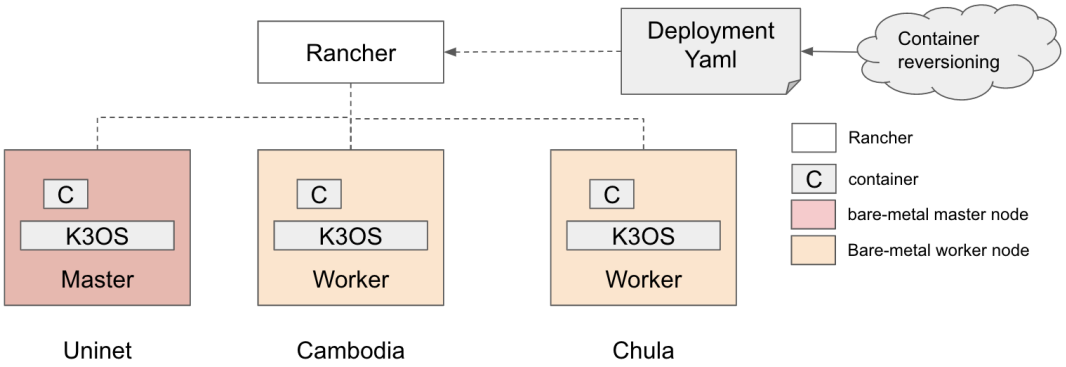
ตารางที่ 6 ตัวอย่างรายการคอนเทนเนอร์อิมเมจและการกำหนดแท็ก

ชื่อคอนเทนเนอร์อิมเมจและแท็ก	ความหมาย
smartenergy	คอนเทนเนอร์อิมเมจ smartenergy รุ่นปัจจุบัน
smartenergy:latest	คอนเทนเนอร์อิมเมจ smartenergy รุ่นล่าสุด
smartenergy:0.0.1	คอนเทนเนอร์อิมเมจ smartenergy รุ่น 0.0.1
smartenergy:arm64	คอนเทนเนอร์อิมเมจ smartenergy แบบ arm64

4.2 ความสำคัญของการปรับรุ่นคอนเทนเนอร์

แอพลิเคชันที่รันในเซิร์ฟเวอร์แต่ละโนด อาจมีการใช้งานต่างเวอร์ชันกัน เช่นความแตกต่างของฮาร์ดแวร์ หรือตามความต้องการของผู้ใช้งาน โดยปกติในงานบำรุงรักษาการปรับรุ่นของแอพลิเคชันบนเซิร์ฟเวอร์คล้ายกับการอัพเกรดเฟิร์มแวร์ (firmware) กล่าวคือต้องมีการวางแผนก่อนการดำเนินการ ทั้งการสำรองข้อมูลก่อนการอัพเกรด การอัพเกรด (upgrade) การดาวน์เกรดกรณีดำเนินการไม่สำเร็จ (downgrade) โดยกิจกรรมทั้งหมดมีลำดับขั้นตอนที่ต้องดำเนินการบนเซิร์ฟเวอร์แต่ละโนด การเปลี่ยนแนวทางบำรุงรักษาจากการใช้งานแอพลิเคชันแบบเดิมไปสู่การห่อแอพลิเคชันด้วยเทคโนโลยีคอนเทนเนอร์และติดแท็ค จะช่วยให้งานบำรุงรักษาสะดวกขึ้นได้

4.3 สถาปัตยกรรมระบบที่นำเสนอ



รูปที่ 25 สถาปัตยกรรมระบบในการจัดการรุ่นของคอนเทนเนอร์

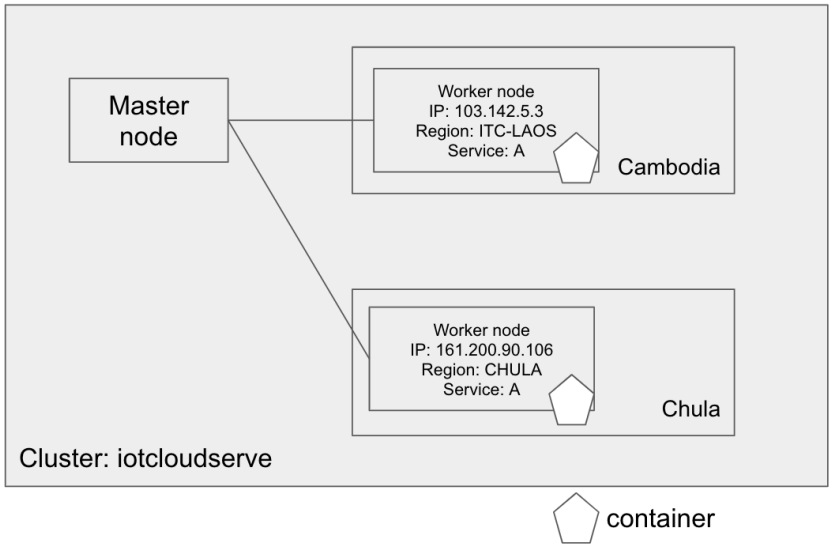
รูปที่ 25 แสดงสถาปัตยกรรมระบบในการจัดการรุ่นของคอนเทนเนอร์ เพื่อนำเสนอคุณสมบัติการปรับรุ่นของคอนเทนเนอร์ในโนดที่อยู่ห่างไกล ทำได้โดยการกำหนดดีพลอยเมนต์ของภาระโดยระบุให้รันคอนเทนเนอร์ในโนดที่เจาะจง จะสามารถกำหนดโนดเซิร์ฟเวอร์และรุ่นของคอนเทนเนอร์ในแต่ละภาระที่แตกต่างกันได้ ตารางที่ 7 แสดงการติดตั้งคิวเบอร์เนเทสในโนดแต่ละโนดตามการออกแบบสถาปัตยกรรม

ตารางที่ 7 รายการติดตั้งคิวเบอร์เนเทสตามสถาปัตยกรรมระบบในการจัดการรุ่นของคอนเทนเนอร์

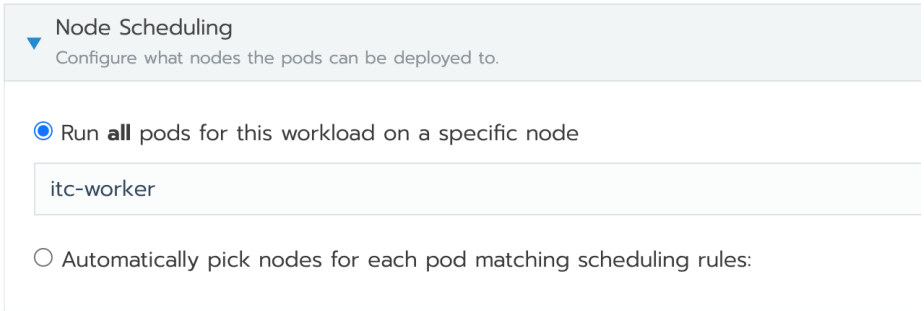
Hostname	Role	IP-Address	OS	Location
master1	master	202.28.193.102	K3OS	UNINET ประเทศไทย เขตพญาไท
k3os-2030	worker	161.200.90.106	K3OS	CHULA ประเทศไทย
itc-worker	worker	103.142.5.3	K3OS	ITC ประเทศกัมพูชา

4.4 การทดสอบสถาปัตยกรรมระบบ

การทดสอบการปรับรุ่นแอพลิเคชันแบบรวมศูนย์ทำโดยสร้างภาระด้วยคอนเทนเนอร์อิมเมจ smartenergy และรันบนโนดที่ห่างไกล แนวคิดเป็นดังรูปที่ 26 แสดงการรันคอนเทนเนอร์ที่โนดจุฬาลงกรณ์มหาวิทยาลัย ประเทศไทย และสถาบัน ITC ประเทศกัมพูชา ในรูปแสดงการรันคอนเทนเนอร์ของ 2 โนดในคลัสเตอร์ โดยกำหนดที่อยู่คอนเทนเนอร์อิมเมจเป็น registry.gitlab.com/iotcloudserve/smartenergy รูปที่ 27 แสดงการกำหนดโนดปลายทางที่รันคอนเทนเนอร์ในระหว่างการสร้างภาระบนรานเชอร์

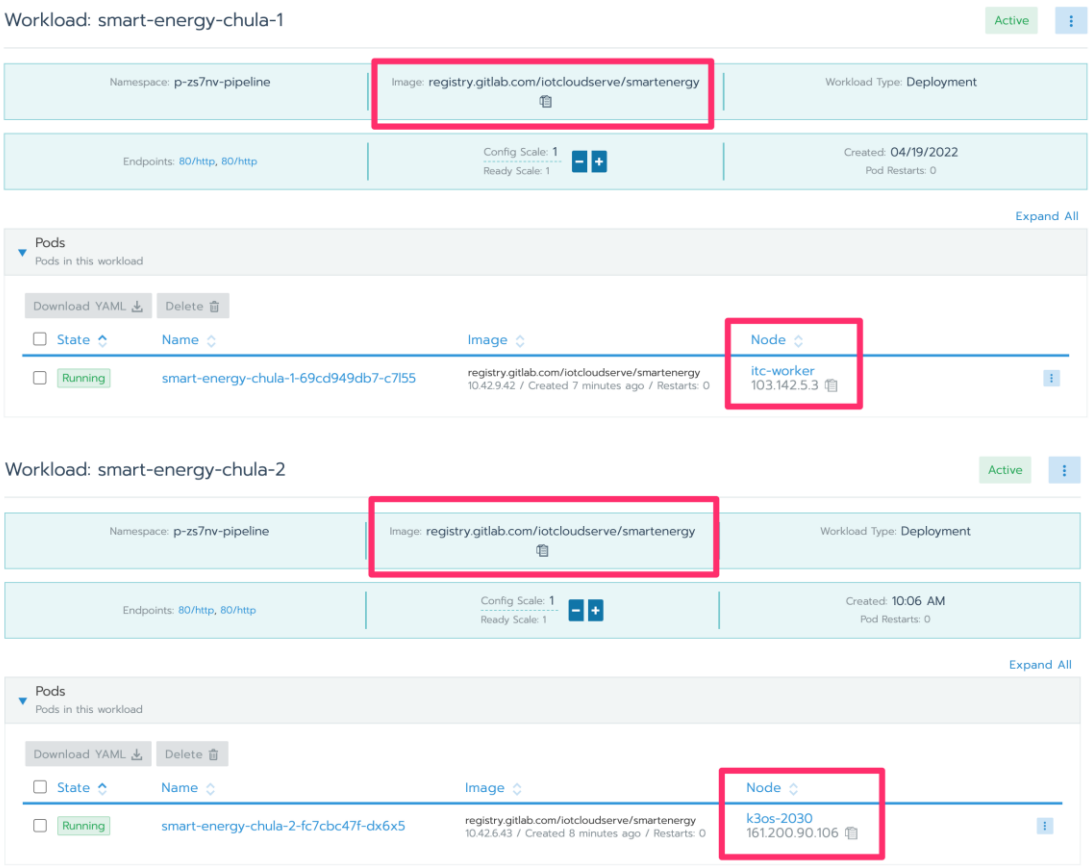


รูปที่ 26 แนวคิดการรันคอนเทนเนอร์กระจายไปแต่ละโนดที่อยู่ห่างไกล



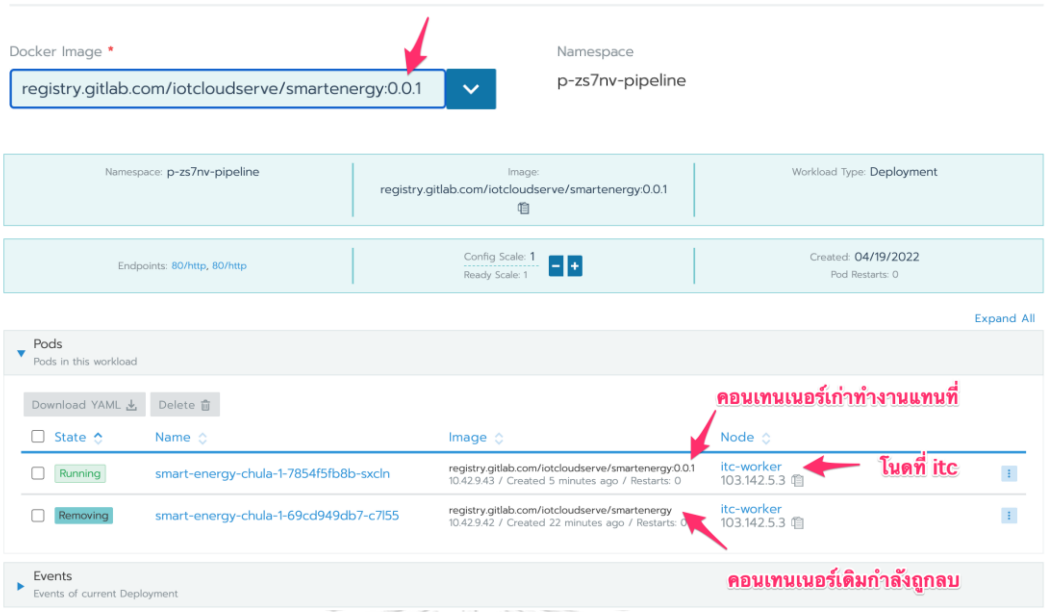
รูปที่ 27 การกำหนดโนดที่ติดตั้งภาระบนโปรแกรมรานเชอร์

การรันคอนเทนเนอร์ smartenergy บนโนด itc-worker ที่อยู่ประเทศกัมพูชา และบนโนด k3os-2030 ที่อยู่ในประเทศไทย มีผลการรันดังรูปที่ 28 คิวเบอร์เนเทสสามารถสั่งรันคอนเทนเนอร์บนโนดทีอยู่ระยะไกลได้

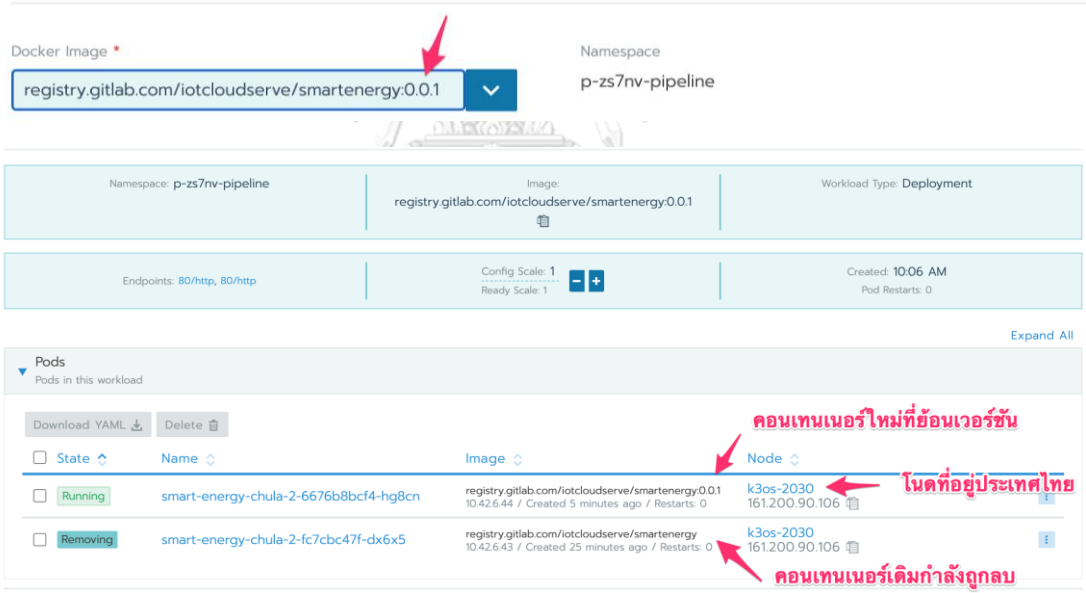


รูปที่ 28 การรันคอนเทนเนอร์แบบรวมศูนย์ที่โนดระยะไกล

จากการทดสอบรันคอนเทนเนอร์ในโนดที่ห่างไกล สามารถส่งคอนเทนเนอร์ไปรันได้ จากนั้นจึงทดสอบหากเกิดกรณีต้องการย้อนเวอร์ชันของคอนเทนเนอร์ จากในรูปที่ 28 เป็นการรันคอนเทนเนอร์เวอร์ชันปัจจุบัน จึงทดสอบกำหนดรันคอนเทนเนอร์ย้อนเวอร์ชันไปที่ 0.0.1 การทดสอบทำโดยกำหนดแท็กของคอนเทนเนอร์โดยการแก้ไขภาระบนรานเชอร์ รูปที่ 29 แสดงการตั้งค่าแท็กและผลการตั้งค่าเพื่อย้อนเวอร์ชันของคอนเทนเนอร์ไป 0.0.1 ของคอนเทนเนอร์ที่รันในโนดที่ประเทศกัมพูชา และผลการทดสอบในทำนองเดียวกันบนโนดที่รันในประเทศไทยดังรูปที่ 30 ผลการทดสอบผู้ใช้งานสามารถย้อนเวอร์ชันของคอนเทนเนอร์บนโนดในประเทศกัมพูชาและประเทศไทยแบบรวมศูนย์ได้

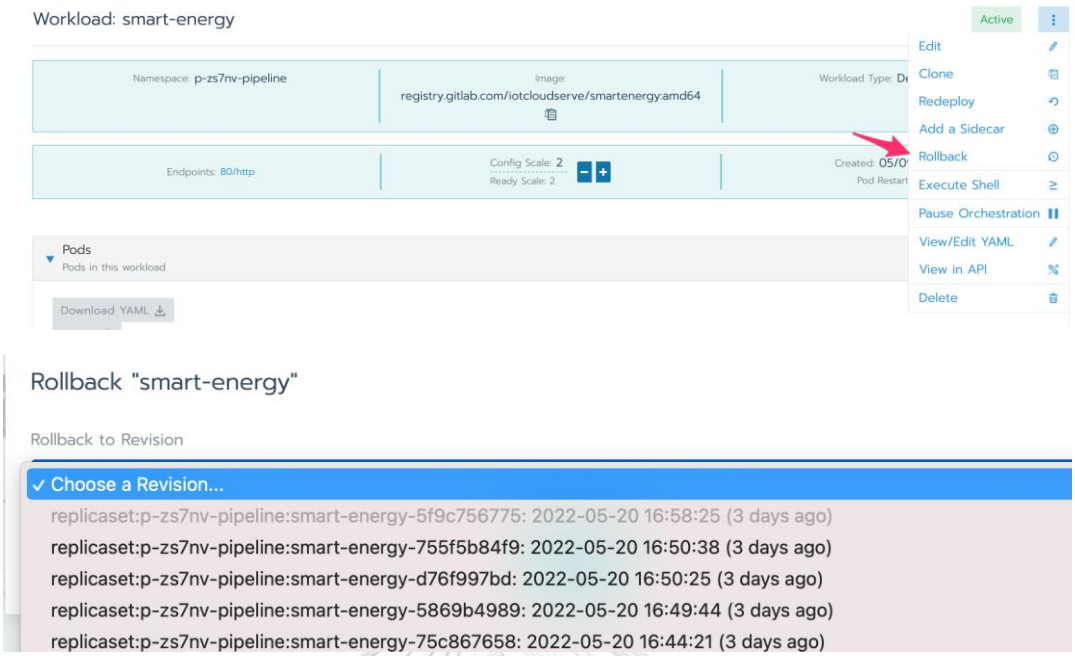


รูปที่ 29 การกำหนดแท็กเพื่อย้อนเวอร์ชันของคอนเทนเนอร์บนภาระของโนดที่รันในประเทศกัมพูชา



รูปที่ 30 การกำหนดแท็กเพื่อย้อนเวอร์ชันของคอนเทนเนอร์บนภาระของโนดที่รันในประเทศไทย

กรณีการย้อนเวอร์ที่นำเสนอเป็นการย้อนเวอร์ชันแบบทำมือ (manual) ซึ่งในการดำเนินการแต่ละครั้งที่มีการเปลี่ยนแปลงการตั้งค่าของภาระ คิวเบอร์เนเทสมีการบันทึกการตั้งค่าในอดีตไว้ จึงสามารถย้อนเวอร์ชันการตั้งค่าภาระผ่านคำสั่งย้อนกลับได้ (rollback) [30] รูปที่ 31 แสดงการใช้คำสั่งย้อนกลับการตั้งค่าในอดีตของภาระบนรานเชอร์



รูปที่ 31 การย้อนกลับการตั้งค่าในอดีตด้วยคำสั่งย้อนกลับบนรานเชอร์

4.5 บทสรุป

จากรูปแบบสถาปัตยกรรมระบบเดิมที่มีการใช้เครื่องมือคิวเบอร์เนเทสในการออร์เคสเตรทคอนเทนเนอร์และติดตั้งคิวเบอร์เนเทสภายในพื้นที่เดียวกัน ในบทนี้ได้ขยายการพิจารณาเป็นสถาปัตยกรรมระบบเพื่อศึกษาการรันคิวเบอร์เนเทสจากหลายโนดในพื้นที่ห่างไกล อีกทั้งการจัดการรุ่นของคอนเทนเนอร์ในคลัสเตอร์ซึ่งสามารถตอบโจทย์ในเรื่องการรัน (configuration) คอนเทนเนอร์แบบรวมศูนย์ สามารถนำไปประยุกต์การจัดการคอนเทนเนอร์ในงานพลังงานอัจฉริยะได้

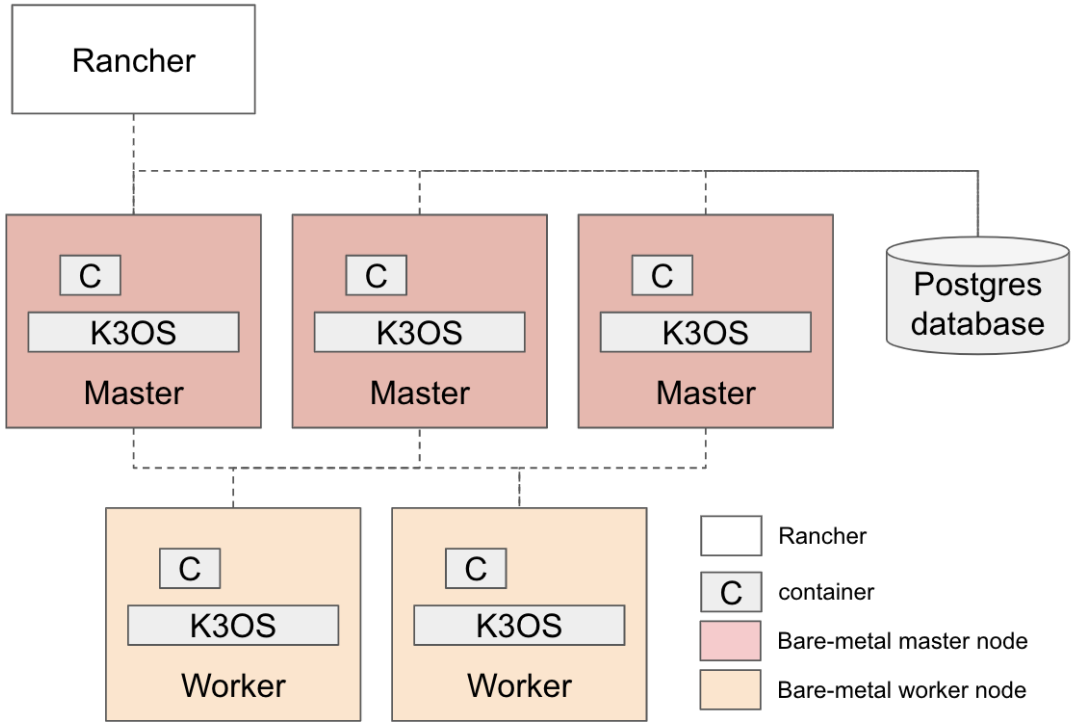
เมื่อผู้พัฒนาปรับปรุงแอพลิเคชันและออกคอนเทนเนอร์เวอร์ชันใหม่ การบำรุงรักษาจะสามารถใช้คอนเทนเนอร์รุ่นเดิมต่อไปได้ จนกว่าจะมีการกำหนดแท็กเพื่อเจาะจงเวอร์ชันของคอนเทนเนอร์ใหม่ หรือหากต้องการย้อนเวอร์ชันของคอนเทนเนอร์ก็สามารถทำได้ การอัพเกรดนี้สามารถสั่งการได้จากศูนย์กลางได้โดยไม่ต้องเดินทางไปบำรุงรักษาเซิร์ฟเวอร์ที่อยู่ห่างไกล สามารถลดขั้นตอนการอัพเกรดและดาวน์เกรดของแอพลิเคชันได้

บทที่ 5
การออกแบบสถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง

โครงสร้างพื้นฐานที่รองรับเทคโนโลยีไอโอทีและการทดสอบการเพิ่มความเร็วการประมวลผลข้อมูลจาก CU-BEMS ได้รับการสนับสนุนจากโครงการ IoTcloudServe@TEIN ในการสร้างแพลทฟอร์มเรียนรู้เทคโนโลยีไอโอทีร่วมกัน การให้บริการมีความเกี่ยวข้องกับผู้ใช้งานหลายคน และเพื่อให้สามารถให้บริการได้ต่อเนื่อง บทนี้จะนำเสนอสถาปัตยกรรมระบบที่สามารถให้บริการจริงได้ (production) โดยพัฒนาคุณสมบัติความพร้อมใช้ระดับสูง (high availability) โดยทำการติดตั้งและพิสูจน์คุณสมบัติดังกล่าว การบำรุงรักษาคลัสเตอร์ (maintenance) ในบางครั้งมีการหยุดใช้งานโนดบางโนดในคลัสเตอร์ การพัฒนาคิวเบอร์เบอร์เนเทสที่มีความพร้อมใช้ระดับสูงขึ้น (high availability) ที่ยังทำให้สามารถรักษาความต่อเนื่องในการให้บริการได้ จึงมีความจำเป็นอย่างยิ่งในทางปฏิบัติ

ในบทนี้สถาปัตยกรรมระบบที่นำเสนอจะให้ความสำคัญด้านการทนต่อความผิดพร่อง (fault tolerance) โดยปกติคิวเบอร์เนเทสคลัสเตอร์ที่มีมาสเตอร์โนดจำนวน 1 โนด และมีเวิร์คเกอร์โนดจำนวนหลายโนดรับงานจากมาสเตอร์โนด หากมีเวิร์คเกอร์โนดบางตัวหยุดทำงาน มาสเตอร์โนดจะรับรู้และสั่งให้โนดอื่นรันคอนเทนเนอร์แทนได้ แต่กรณีที่มาสเตอร์โนดที่มีเพียงหนึ่งเดียวหยุดทำงาน คอนเทนเนอร์ที่ทำงานอยู่ในเวิร์คเกอร์โนดจะทำงานได้ปกติ แต่ผู้ดูแลระบบจะไม่สามารถตรวจสอบสถานะ (monitor) และควบคุมคอนเทนเนอร์ต่าง ๆ ในคลัสเตอร์ได้จนกว่าจะซ่อมแซมเครื่องมาสเตอร์โนดให้กลับมาทำงานปกติ ในการใช้งานจริงจึงอาจมีเหตุที่ทำให้มาสเตอร์โนดหยุดทำงาน จึงต้องพัฒนาคิวเบอร์เนเทสแบบหลายมาสเตอร์โนด โดยเมื่อเกิดเหตุขึ้นระบบจะต้องสามารถใช้งานติดตามสถานะ หรือควบคุมคอนเทนเนอร์ต่อไปได้

5.1 สถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง



รูปที่ 32 สถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง

รูปที่ 32 แสดงสถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง ประกอบด้วยคิวเบอร์เนเทสมาสเตอร์โนดจำนวน 3 โนด และคิวเบอร์เนเทสเวิร์คเกอร์จำนวน 2 โนด จำนวนของคิวเบอร์เนเทสมาสเตอร์โนดเป็นไปตามข้อกำหนดของ [31] และ [32] โดยมีฐานข้อมูลโปสเกรสจัดเก็บสถานะของคิวเบอร์เนเทสคลัสเตอร์แยกออกมาจากคลัสเตอร์ การสร้างคิวเบอร์เนเทสความพร้อมใช้ระดับสูงแบ่งออกเป็น 2 ชนิดตามรูปแบบการจัดเก็บข้อมูลสถานะคลัสเตอร์ กล่าวคือได้แก่ แบบฐานข้อมูลภายใน และแบบฐานข้อมูลภายนอก แบบฐานข้อมูลภายในจะไม่มีฐานข้อมูลจัดเก็บสถานะแยกออกมาจากคลัสเตอร์ แต่จะจัดเก็บกระจายไปทุกมาสเตอร์โนด ทำให้หากมีมาสเตอร์หยุดให้บริการเป็นจำนวนโนดมากเกินไป จะทำให้จำนวนโนดไม่เพียงพอต่อการทำงานของฐานข้อมูลจัดเก็บสถานะ ทำให้ไม่สามารถควบคุมคลัสเตอร์ได้ [31] ฐานข้อมูลภายในส่วนใหญ่จะใช้ฐานข้อมูลอีทีซีดี (etcd [33]) ในการจัดเก็บสถานะ ส่วนแบบฐานข้อมูลภายนอก มีข้อสำคัญคือทำให้ระดับความพร้อมใช้ของคลัสเตอร์ขึ้นอยู่กับความพร้อมใช้ของฐานข้อมูลด้วย เพราะหากฐานข้อมูลภายนอกหยุดทำงานหรือเข้าถึงไม่ได้ จะเป็นเหตุให้ไม่สามารถควบคุมและตรวจสอบสถานะคลัสเตอร์ได้เช่นกัน แต่เนื่องจากไม่มีการจัดเก็บสถานะกระจายไปแต่ละโนด ทำให้มีความสะดวกในการบำรุงรักษา

มาสเตอร์โนดที่สามารถมีจำนวนมาสเตอร์โนดหยุดทำงานได้มากกว่าแบบฐานข้อมูลภายใน โดยไม่กระทบกับข้อมูลที่จัดเก็บสถานะของคลัสเตอร์

5.2 การทดสอบความพร้อมใช้ระดับสูง

การทดสอบความพร้อมใช้งานระดับสูงทำโดยหยุดการทำงานเวิร์คเกอร์โนดและมาสเตอร์โนด และทดสอบความพร้อมใช้ 2 กรณีดังนี้

1. คอนเทนเนอร์สามารถทำงานอยู่ในเวิร์คเกอร์โนดหรือมาสเตอร์โนดได้หรือไม่
2. ผู้ดูแลระบบสามารถติดตามสถานะคลัสเตอร์และควบคุมคอนเทนเนอร์ได้หรือไม่

ในการทดสอบนี้ ได้ออกแบบและติดตั้งคิวเบอร์เนเทส 3 ลักษณะดังนี้

- คลัสเตอร์ที่ 1 ประกอบด้วยมาสเตอร์โนด 1 โนด และเวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายใน
- คลัสเตอร์ที่ 2 ประกอบด้วยมาสเตอร์โนด 3 โนด และเวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายใน
- คลัสเตอร์ที่ 3 ประกอบด้วยมาสเตอร์โนด 3 โนด และเวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก

และออกแบบกรณีการหยุดการทำงานของโนดดังนี้

- หยุดการทำงานของเวิร์คเกอร์โนดทีละโนด แล้วตรวจสอบความพร้อมใช้
- หยุดการทำงานของมาสเตอร์โนดทีละโนด แล้วตรวจสอบความพร้อมใช้

เพื่อความสะดวกในการทดสอบคุณสมบัติ จึงใช้เครื่องมือเคสามดีในการสร้างคิวเบอร์เนเทสคลัสเตอร์ และเพื่อทดสอบการเข้าถึงบริการของคอนเทนเนอร์ที่รันในคลัสเตอร์จึงสร้างคอนเทนเนอร์ nginx อย่างง่ายบนคลัสเตอร์โดยรันคำสั่งตามลำดับในภาคผนวก คำสั่งที่ใช้สร้างคลัสเตอร์เพื่อทดสอบความพร้อมใช้สรุปได้ดังตารางที่ 8 ผลการติดตั้งเป็นดังตารางที่ 9 และผลการทดสอบการเข้าถึงบริการแต่ละบริการในสถานการณ์ปกติเป็นดังตารางที่ 10 และตารางที่ 11 ตามลำดับ

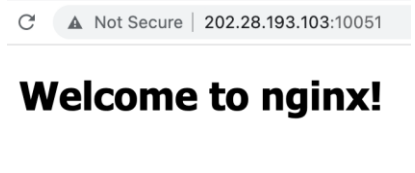
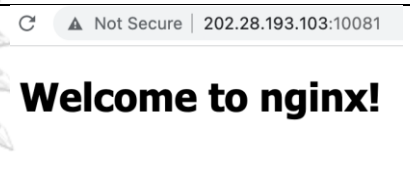
ตารางที่ 8 คำสั่งการสร้างคิวเบอร์เนเทสคลัสเตอร์ด้วยเคสามดี

ลักษณะคลัสเตอร์	คำสั่งสร้างคลัสเตอร์
คลัสเตอร์ชื่อ k3d-c1 ประกอบด้วย มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด เชื่อมโฮสพอร์ต 10051 ไปคลัสเตอร์พอร์ต 80	k3d cluster create c1 -i rancher/k3s:v1.21.1-k3s1 \ -s 1 \ -a 2 \ -p "10051:80@loadbalancer"
คลัสเตอร์ชื่อ k3d-c2 ประกอบด้วย มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด เชื่อมโฮสพอร์ต 10061 ไปคลัสเตอร์พอร์ต 80	k3d cluster create c2 -i rancher/k3s:v1.21.1-k3s1 \ -s 3 \ -a 2 \ -p "10061:80@loadbalancer"
คลัสเตอร์ชื่อ k3d-c3 ประกอบด้วย มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด เชื่อมโฮสพอร์ต 10071 ไปคลัสเตอร์พอร์ต 80 ที่มาสเตอร์โนด 1 เชื่อมต่อฐานข้อมูลภายนอก ที่มาสเตอร์โนด 2 เชื่อมต่อฐานข้อมูลภายนอก ที่มาสเตอร์โนด 3 เชื่อมต่อฐานข้อมูลภายนอก	k3d cluster create c3 -i rancher/k3s:v1.21.1-k3s1 \ -s 3 \ -a 2 \ -p "10071:80@loadbalancer" \ --k3s-arg "--datastore-endpoint=db@server:0" \ --k3s-arg "--datastore-endpoint=db@server:1" \ --k3s-arg "--datastore-endpoint=db@server:2"

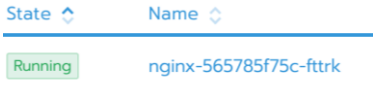
ตารางที่ 9 ผลการสร้างคิวเบอร์เนเทสคลัสเตอร์ด้วยเคสามดี

ลักษณะคลัสเตอร์	Hostname	IP-address	Role	Exposed port
คลัสเตอร์ชื่อ k3d-c1 มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c1-server-0	172.28.0.2	server	10051
	k3d-c1-agent-0	172.28.0.4	worker	
	k3d-c1-agent-1	172.28.0.3	worker	
คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	172.29.0.2	server	10061
	k3d-c2-server-1	172.29.0.3	server	
	k3d-c2-server-2	172.29.0.4	server	
	k3d-c2-agent-0	172.29.0.6	worker	
	k3d-c2-agent-1	172.29.0.5	worker	
คลัสเตอร์ชื่อ k3d-c3 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก	k3d-c3-server-0	172.30.0.2	server	10081
	k3d-c3-server-1	172.30.0.3	server	
	k3d-c3-server-2	172.30.0.4	server	
	k3d-c3-agent-0	172.30.0.5	worker	
	k3d-c3-agent-1	172.30.0.6	worker	

ตารางที่ 10 ผลการทดสอบการเข้าถึงเซอร์วิสบนคลัสเตอร์ในสถานการณ์ปกติ

ลักษณะคลัสเตอร์	Hostname	Role	ทดสอบการเข้าถึงบริการ
คลัสเตอร์ชื่อ k3d-c1 มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c1-server-0	server	
	k3d-c1-agent-0	worker	
	k3d-c1-agent-1	worker	
คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	server	
	k3d-c2-server-1	server	
	k3d-c2-server-2	server	
	k3d-c2-agent-0	worker	
	k3d-c2-agent-1	worker	
คลัสเตอร์ชื่อ k3d-c3 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก	k3d-c3-server-0	server	
	k3d-c3-server-1	server	
	k3d-c3-server-2	server	
	k3d-c3-agent-0	worker	
	k3d-c3-agent-1	worker	

ตารางที่ 11 สถานะคอนเทนเนอร์และโนดที่ทำงานในสถานการณ์ปกติ

ลักษณะคลัสเตอร์	Hostname	Role	สถานะคอนเทนเนอร์	โนดที่ทำงาน
คลัสเตอร์ชื่อ k3d-c1 มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c1-server-0	server		
	k3d-c1-agent-0	worker		
	k3d-c1-agent-1	worker		
คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	server		
	k3d-c2-server-1	server		
	k3d-c2-server-2	server		
	k3d-c2-agent-0	worker		
	k3d-c2-agent-1	worker		
คลัสเตอร์ชื่อ k3d-c3 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูล ภายนอก	k3d-c3-server-0	server		
	k3d-c3-server-1	server		
	k3d-c3-server-2	server		
	k3d-c3-agent-0	worker		
	k3d-c3-agent-1	worker		

ตารางที่ 12 การทดสอบการหยุดการทำงานบนโนดที่คอนเทนเนอร์ทำงานและสถานะเซอร์วิสหลังโนดที่รันคอนเทนเนอร์เดิมหยุดทำงาน

ลักษณะคลัสเตอร์	Hostname	Role	สถานะโนด	สถานะบริการ
กรณีที่ 1 คลัสเตอร์ชื่อ k3d-c1 มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c1-server-0	server	เปิด	202.28.193.103:10051 **to nginx!**
	k3d-c1-agent-0	worker	เปิด	
	k3d-c1-agent-1	worker	ปิด	
กรณีที่ 2 คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	server	เปิด	202.28.193.103:10061 **to nginx!**
	k3d-c2-server-1	server	เปิด	
	k3d-c2-server-2	server	เปิด	
	k3d-c2-agent-0	worker	เปิด	
	k3d-c2-agent-1	worker	ปิด	
กรณีที่ 3 คลัสเตอร์ชื่อ k3d-c3 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก	k3d-c3-server-0	server	เปิด	202.28.193.103:10081 **to nginx!**
	k3d-c3-server-1	server	ปิด	
	k3d-c3-server-2	server	เปิด	
	k3d-c3-agent-0	worker	ปิด	
	k3d-c3-agent-1	worker	ปิด	
กรณีที่ 4 คลัสเตอร์ชื่อ k3d-c1 มาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c1-server-0	server	ปิด	202.28.193.103:10051 **to nginx!**
	k3d-c1-agent-0	worker	เปิด	
	k3d-c1-agent-1	worker	ปิด	

สถานะคอนเทนเนอร์ กรณีที่ 1

State	Name	Node
Unknown	nginx-565785f75c-tttxc	k3d-c1-agent-1 172.28.0.3
Running	nginx-565785f75c-29dw4	k3d-c1-agent-0 172.28.0.4

สถานะคอนเทนเนอร์ กรณีที่ 2

State	Name	Node
Unknown	nginx-565785f75c-2qzzz	k3d-c2-agent-1 172.29.0.5
Running	nginx-565785f75c-k6lsz	k3d-c2-agent-0 172.29.0.6

สถานะคอนเทนเนอร์ กรณีที่ 3

State	Name	Node
Unknown	nginx-565785f75c-mf6w4	k3d-c3-server-1 172.30.0.3
Unknown	nginx-565785f75c-fttrk	k3d-c3-agent-0 172.30.0.5
Running	nginx-565785f75c-dnhml	k3d-c3-server-0 172.30.0.2

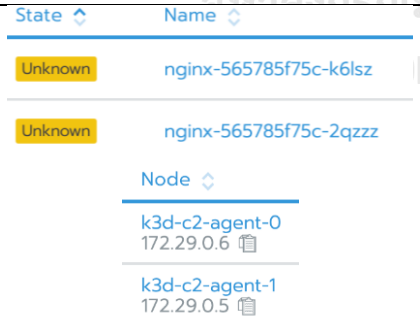
สถานะคอนเทนเนอร์ กรณีที่ 4

คอนเทนเนอร์ทำงานที่โนด k3d-c1-agent-0

ตารางที่ 13 การทดสอบการหยุดการทำงานบนโนดที่คอนเทนเนอร์ทำงานและสถานะเซอร์วิสหลังโนดที่รันคอนเทนเนอร์เดิมหยุดทำงาน (ต่อ)

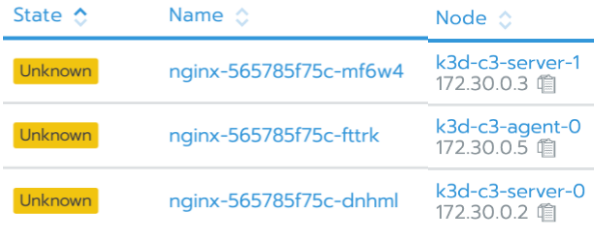
ลักษณะคลัสเตอร์	Hostname	Role	สถานะโนด	สถานะบริการ
กรณีที่ 5 คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	server	ปิด	Unavailable
	k3d-c2-server-1	server	เปิด	
	k3d-c2-server-2	server	เปิด	
	k3d-c2-agent-0	worker	ปิด	
	k3d-c2-agent-1	worker	ปิด	
กรณีที่ 6 คลัสเตอร์ชื่อ k3d-c2 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด	k3d-c2-server-0	server	ปิด	Unavailable
	k3d-c2-server-1	server	เปิด	
	k3d-c2-server-2	server	ปิด	
	k3d-c2-agent-0	worker	ปิด	
	k3d-c2-agent-1	worker	ปิด	
กรณีที่ 7 คลัสเตอร์ชื่อ k3d-c3 มาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก	k3d-c3-server-0	server	เปิด	Unavailable
	k3d-c3-server-1	server	ปิด	
	k3d-c3-server-2	server	ปิด	
	k3d-c3-agent-0	worker	ปิด	
	k3d-c3-agent-1	worker	ปิด	

สถานะคอนเทนเนอร์ กรณีที่ 5



สถานะคอนเทนเนอร์ กรณีที่ 6

สถานะคอนเทนเนอร์ กรณีที่ 7



ตารางที่ 12 และ ตารางที่ 13 แสดงการทดสอบการหยุดการทำงานบนโนดที่คอนเทนเนอร์ทำงานและสถานะบริการหลังโนดที่รันคอนเทนเนอร์เดิมหยุดทำงาน ดังนี้

กรณีที่ 1 คลัสเตอร์มีมาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายใน เมื่อมี 1 เวิร์คเกอร์โนดหยุดทำงานที่แต่เดิมมีคอนเทนเนอร์ทำงานอยู่ในโนดนั้น คลัสเตอร์จะสั่งให้โนดอื่นทำงานแทนได้ ดังตารางที่ 12 กรณีที่ 1 การทดสอบนี้แสดงว่า ระดับความพร้อมใช้ไม่ขึ้นอยู่กับเวิร์คเกอร์โนด สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายในแบบมีมาสเตอร์โนด 1 โนด

กรณีที่ 2 คลัสเตอร์มีมาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายใน เมื่อมี 1 เวิร์คเกอร์โนดหยุดทำงานที่แต่เดิมมีคอนเทนเนอร์ทำงานอยู่ในโนดนั้น คลัสเตอร์จะสั่งให้โนดอื่นทำงานแทนได้ ดังตารางที่ 12 กรณีที่ 2 การทดสอบนี้แสดงว่า ระดับความพร้อมใช้ไม่ขึ้นอยู่กับเวิร์คเกอร์โนด สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายในแบบมีมาสเตอร์โนดหลายโนด

กรณีที่ 3 คลัสเตอร์มีมาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายนอก เมื่อมี 2 เวิร์คเกอร์โนดหยุดทำงานที่แต่เดิมมีคอนเทนเนอร์ทำงานอยู่หนึ่งในโนดนั้น และมี 1 มาสเตอร์โนดหยุดทำงาน คลัสเตอร์จะยังคงสั่งให้โนดอื่นทำงานแทนได้ ดังตารางที่ 12 กรณีที่ 3 การทดสอบนี้แสดงว่า สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายนอกแบบมีมาสเตอร์โนดหลายโนด ดังในกรณีนี้ 3 โนด สามารถมีมาสเตอร์โนดหยุดทำงานได้ 1 โนด

กรณีที่ 4 คลัสเตอร์มีมาสเตอร์โนด 1 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายใน จากกรณีที่ 1 เมื่อมี 1 เวิร์คเกอร์โนดหยุดทำงาน และ 1 มาสเตอร์โนดหยุดทำงานเพิ่มขึ้น จะส่งผลให้คลัสเตอร์ไม่พร้อมใช้งาน ไม่สามารถติดตามสถานะได้ แต่คอนเทนเนอร์ที่รันอยู่บนเวิร์คเกอร์โนดเดิมยังคงให้บริการได้ ดังตารางที่ 12 กรณีที่ 4 การทดสอบนี้แสดงว่า สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายใน แบบมีมาสเตอร์โนด 1 โนด เมื่อมาสเตอร์โนดนี้โนดเดียวหยุดทำงาน จะไม่สามารถควบคุมคลัสเตอร์ได้

กรณีที่ 5 คลัสเตอร์มีมาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายใน จากกรณีที่ 2 มีเวิร์คเกอร์ 1 โนดหยุดทำงาน และมี 1 มาสเตอร์โนดหยุดทำงาน ทดสอบโดยหยุดมาสเตอร์โนดเพิ่ม 1 โนด และหยุดเวิร์คเกอร์เพิ่ม 1 โนด เป็นกรณีเหลือ 2 มาสเตอร์โนด จะทำให้คลัสเตอร์ยังควบคุมอยู่ได้ แต่คลัสเตอร์จะไม่สามารถตัดสินใจย้ายคอนเทนเนอร์ไปในโนดอื่นอัตโนมัติได้ (ไม่การันตีความอัตโนมัติ) ผู้ดูแลระบบต้องย้ายคอนเทนเนอร์เอง ดังตารางที่ 13 กรณีที่ 5 การทดสอบนี้แสดงว่า สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายในแบบมีมาสเตอร์โนดหลายโนด ดังในกรณีนี้ 3 โนด สามารถมีมาสเตอร์โนดหยุดทำงานได้ 1 โนดและเหลือ 2 มาสเตอร์โนดทำงาน โดยจะยังสามารถควบคุมคลัสเตอร์ได้

กรณีที่ 6 คลัสเตอร์มีมาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด มีฐานข้อมูลจัดเก็บสถานะภายใน จากกรณีที่ 5 เมื่อมี 2 เวิร์คเกอร์โนดหยุดทำงาน และมี 1 มาสเตอร์โนดหยุดทำงานแล้ว ทดสอบให้ 1 มาสเตอร์โนดหยุดทำงานเพิ่ม เป็นกรณีเหลือ 1 มาสเตอร์โนด คลัสเตอร์จะไม่พร้อมใช้งาน ดังตารางที่ 13 กรณีที่ 6 การทดสอบนี้แสดงว่า สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายใน แบบมีมาสเตอร์โนดหลายโนด ดังในกรณีนี้ 3 โนด เมื่อมาสเตอร์โนดหยุดทำงานครบ 2 โนด จะไม่สามารถควบคุมคลัสเตอร์ได้

กรณีที่ 7 คลัสเตอร์มีมาสเตอร์โนด 3 โนด เวิร์คเกอร์โนด 2 โนด แบบฐานข้อมูลภายนอก จากกรณีที่ 3 ที่หยุดการทำงานทั้ง 2 เวิร์คเกอร์โนด และ 1 มาสเตอร์โนด แล้วคลัสเตอร์ยังสามารถสั่งการให้โนดอื่นทำงานแทนได้ จึงให้หยุดการทำงานเพิ่มอีก 1 มาสเตอร์โนด และเหลือ 1 มาสเตอร์โนดเดียวในคลัสเตอร์ พบว่าคลัสเตอร์ยังสามารถตรวจสอบสถานะและควบคุมได้ ดังตารางที่ 13 กรณีที่ 7 การทดสอบนี้แสดงว่า สำหรับกรณีคิวเบอร์เนเทสคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายนอกแบบมีมาสเตอร์โนดหลายโนด ดังในกรณีนี้ 3 โนด สามารถมีมาสเตอร์โนดหยุดทำงานได้มากถึง 2 โนด

กรณีที่ 6 และกรณีที่ 7 ต่างเหลือ 1 มาสเตอร์โนดเหมือนกัน แต่กรณีที่ 6 คลัสเตอร์ไม่พร้อมใช้งาน ไม่สามารถควบคุมคลัสเตอร์ได้ ในขณะกรณีที่ 7 ยังสามารถตรวจสอบสถานะและควบคุมคลัสเตอร์ได้ เนื่องจากคลัสเตอร์ในกรณีที่ 7 มีการใช้ฐานข้อมูลภายนอก จะสามารถรองรับโนดมาสเตอร์หยุดทำงานได้มากกว่าแบบฐานข้อมูลภายใน

5.3 แนวทางการประเมินความพร้อมใช้ระดับสูง

แนวทางการประเมินความพร้อมใช้ระดับสูง มีประเด็น 2 ส่วนที่พิจารณาคือ ความพร้อมใช้ (availability) และ ความพร้อมใช้ระดับสูง (high availability)

- หากระบบมีความพร้อมใช้ระดับสูง จะทำให้ ระบบมีความพร้อมใช้ด้วย
- หากระบบมีความพร้อมใช้ จะไม่จำเป็นว่า ระบบมีความพร้อมใช้ระดับสูง
- ความพร้อมใช้ของคิวเบอร์เนเทสคลัสเตอร์ ขึ้นอยู่กับ ความพร้อมใช้ของฐานข้อมูลจัดเก็บสถานะ และ ความพร้อมใช้ของระนาบควบคุม (control plane) หากขาดอย่างใดอย่างหนึ่งคลัสเตอร์จะไม่มีความพร้อมใช้
- ความพร้อมใช้ระดับสูงของคิวเบอร์เนเทสคลัสเตอร์ ขึ้นอยู่กับ
    - มีโนดใดก็ได้ในคลัสเตอร์สามารถตายได้ (ทั้งมาสเตอร์โนด และเวิร์คเกอร์โนด)
    - มีจำนวนระนาบควบคุม 2 แกนขึ้นไป
    - ฐานข้อมูลจัดเก็บสถานะมีความพร้อมใช้ระดับสูง
- กรณีคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายใน ส่วนใหญ่เป็นฐานข้อมูลแบบ อีทีซีดี จะทำให้จำนวนมาสเตอร์โนด เท่ากับจำนวนโนดที่มี อีทีซีดี และเท่ากับจำนวนระนาบควบคุม
- กรณีคลัสเตอร์แบบฐานข้อมูลจัดเก็บสถานะภายนอก ให้พิจารณาแยกทั้ง ระดับความพร้อมใช้ของฐานข้อมูลจัดเก็บสถานะ และ จำนวนระนาบควบคุม
- ความพร้อมใช้ของระนาบควบคุม ขึ้นอยู่กับ จำนวนระนาบควบคุม
    - จำนวนระนาบควบคุม เท่ากับ 0 ทำให้ควบคุมคลัสเตอร์ไม่ได้ ระนาบควบคุมไม่มีความพร้อมใช้
    - จำนวนระนาบควบคุม มากกว่าหรือเท่ากับ 1 ทำให้ระนาบควบคุมมีความพร้อมใช้
    - จำนวนระนาบควบคุม มากกว่าหรือเท่ากับ 2 ทำให้ระนาบควบคุมมีความพร้อมใช้ระดับสูง
    - จำนวนระนาบควบคุม มากกว่าหรือเท่ากับ 2 และเป็นเลขคู่ ทำให้ไม่การันตีว่าระนาบควบคุมจะสามารถตัดสินใจเองได้อัตโนมัติ (อาจเกิดปัญหาการเลือกผู้นำ (Leader selection) [34])
    - จำนวนระนาบควบคุม มากกว่าหรือเท่ากับ 1 และเป็นเลขคี่ จะการันตีว่าระนาบควบคุมจะสามารถตัดสินใจเองได้อัตโนมัติ [34]
- ความพร้อมใช้ของฐานข้อมูลจัดเก็บสถานะ แบบ อีทีซีดี เป็นไปตามตารางที่ 14 [35]

ตารางที่ 14 จำนวนโนดที่มี อีทีซีดี ที่มีผลต่อจำนวนการทนต่อความผิดพร่อง

จำนวนโนดที่มีอีทีซีดี	จำนวนการทนต่อความผิดพร่อง
1	0
2	0
3	1
4	1
5	2
6	2
7	3
8	3
9	4

อีทีซีดี ใช้ราฟอัลกอริทึม (raft algorithm) ในการคัดเลือกโนดผู้นำ [36] ดังนั้นในขณะหนึ่งคลัสเตอร์จะมีความพร้อมใช้ก็ต่อเมื่อ จำนวนโนดที่มีอีทีซีดีที่พร้อมใช้มีจำนวน**มากกว่าครึ่งหนึ่ง**ของจำนวนโนดที่มีอีทีซีดีทั้งหมด

- จำนวนการทนต่อความผิดพร่อง คือ จำนวนโนดที่มีอีทีซีดี ที่ตายได้สูงสุด
    - หากมีค่ามากกว่าศูนย์ แปลว่า ฐานข้อมูลจัดเก็บสถานะมีความพร้อมใช้ระดับสูง
    - หากมีค่าเท่ากับศูนย์ แปลว่า ฐานข้อมูลจัดเก็บสถานะมีความพร้อมใช้
- หากจำนวนโนดที่มี อีทีซีดี มากขึ้นเป็นจำนวนคี่ จะทำให้จำนวนการทนต่อความผิดพร่องเพิ่มขึ้น แต่หากมากขึ้นเป็นจำนวนคู่ จะไม่ทำให้จำนวนการทนต่อความผิดพร่องมากขึ้น
- ระดับความพร้อมใช้ ไม่ขึ้นอยู่กับจำนวนเวิร์คเกอร์โนด

5.4 สรุปผลการทดสอบและการประเมินระดับความพร้อมใช้

ตารางที่ 15 แสดงการประเมินระดับความพร้อมใช้เทียบกับผลการทดสอบ สรุปผลการทดสอบการประเมินระดับความพร้อมใช้สอดคล้องกับผลการทดสอบทุกกรณีตั้งแต่กรณีที่ 1 ถึง 7 ตัวอย่างลำดับที่ 7 ถึง 9 เป็นตัวอย่างการใช้คิวเบอร์เนเทสที่มีความพร้อมใช้ระดับสูง และตัวอย่างลำดับที่ 12 ถึง 14 เป็นตัวอย่างที่มีระดับความพร้อมใช้ขึ้นกับระดับความพร้อมใช้ของฐานข้อมูลจัดเก็บสถานะภายนอก

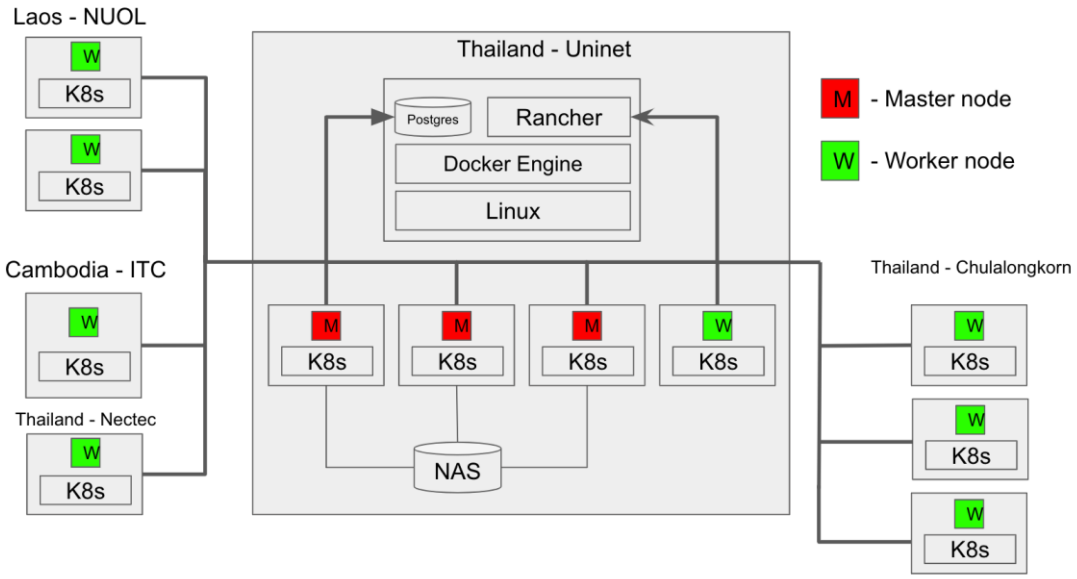
ตารางที่ 15 การประเมินระดับความพร้อมใช้เทียบกับผลการทดสอบ

ลำดับ	w1	w2	m1	m2	m3	ฐานข้อมูล	การย้ายคอนเทนเนอร์	ระดับความพร้อมใช้	ผลการทดสอบ
1	/	/	/	-	-	ภายใน	อัตโนมัติ	พร้อมใช้	สถานการณ์ปกติ
2	x	/	/	-	-	ภายใน	อัตโนมัติ	พร้อมใช้	กรณีที่ 1
3	x	x	/	-	-	ภายใน	อัตโนมัติ	พร้อมใช้	-
4	x	/	x	-	-	ภายใน	-	ไม่พร้อมใช้	กรณีที่ 4
5	-	-	/	/	-	ภายใน	-	พร้อมใช้	สถานการณ์ปกติ
6	-	-	x	/	-	ภายใน	-	ไม่พร้อมใช้	-
7	/	/	/	/	/	ภายใน	อัตโนมัติ	พร้อมใช้ระดับสูง	สถานการณ์ปกติ
8	x	/	/	/	/	ภายใน	อัตโนมัติ	พร้อมใช้ระดับสูง	กรณีที่ 2
9	x	x	/	/	/	ภายใน	อัตโนมัติ	พร้อมใช้ระดับสูง	-
10	x	x	x	/	/	ภายใน	ไม่การันตี	พร้อมใช้	กรณีที่ 5
11	x	x	x	x	/	ภายใน	-	ไม่พร้อมใช้	กรณีที่ 6
12	/	/	/	/	/	ภายนอก	อัตโนมัติ	ขึ้นกับฐานข้อมูล	สถานการณ์ปกติ
13	x	/	/	/	/	ภายนอก	อัตโนมัติ	ขึ้นกับฐานข้อมูล	-
14	x	x	/	/	/	ภายนอก	อัตโนมัติ	ขึ้นกับฐานข้อมูล	-
15	x	x	x	/	/	ภายนอก	ไม่การันตี	ขึ้นกับฐานข้อมูล	กรณีที่ 3
16	x	x	x	x	/	ภายนอก	อัตโนมัติ	ขึ้นกับฐานข้อมูล	กรณีที่ 7

คอลัมน์ 1-5 แสดงสัญลักษณ์ w1 w2 m1 m2 และ m3 หมายถึง เวิร์คเกอร์โนด 1 เวิร์คเกอร์โนด 2 มาสเตอร์โนด 1 มาสเตอร์โนด 2 และ มาสเตอร์โนด 3 ตามลำดับ และ สัญลักษณ์ / หมายถึง โนดใช้งานได้ปกติ x หมายถึง โนดหยุดทำงาน และ - หมายถึง ไม่มีโนดนี้ในคลัสเตอร์

5.5 ตัวอย่างการประยุกต์โครงการ IoTcloudServe@TEIN

สถาปัตยกรรมระบบของโครงการ IoTcloudServe@TEIN เป็นดังรูปที่ 33 ซึ่งแสดงโครงสร้างสถาปัตยกรรมที่มีการออกแบบมาสเตอร์โนดจำนวน 3 โนดเพื่อเปิดคุณสมบัติความพร้อมใช้ระดับสูง ชนิดฐานข้อมูลจัดเก็บสถานะแบบภายนอก (ในรูปแสดงเป็นฐานข้อมูล Postgres) รายการติดตั้งคิวเบอร์เนเทสของสถาปัตยกรรมเป็นตารางที่ 16 สถาปัตยกรรมนี้สามารถมีคิวเบอร์เนเทสมาสเตอร์โนดหยุดทำงานได้สูงสุด 2 มาสเตอร์โนด และมีเวิร์คเกอร์โนดหยุดทำงานเป็นจำนวนโนดเท่าใดก็ได้ ที่ยังคงสามารถติดตามสถานะและควบคุมคลัสเตอร์ได้



รูปที่ 33 สถาปัตยกรรมระบบโครงการไอโอทีคลาวด์เซิร์ฟ

ตารางที่ 16 รายการติดตั้งคิวเบอร์เนเทสตามสถาปัตยกรรมระบบความพร้อมใช้ระดับสูง

Hostname	IP-Address	OS	Architecture	Role	Container engine	Location
master1	202.28.193.102	K3OS	amd64	Master	containerd	Uninet, Thailand
master2	202.28.193.100	K3OS	amd64	Master	containerd	Uninet, Thailand
master3	202.28.193.101	K3OS	amd64	Master	containerd	Uninet, Thailand
k3os-2030	161.200.90.106	K3OS	amd64	Worker	containerd	Chula, Thailand
k3os-4689	161.200.90.110	K3OS	amd64	Worker	containerd	Chula, Thailand
k3os-8550	161.200.90.111	K3OS	amd64	Worker	containerd	Chula, Thailand
alpine	202.28.193.105	K3OS	amd64	Worker	containerd	Uninet, Thailand
itc-worker	103.142.5.3	K3OS	amd64	Worker	containerd	ITC, Cambodia
-	-	K3OS	amd64	Worker	containerd	Nectec, Thailand
-	-	K3OS	amd64	Worker	containerd	NUOL, Laos

5.6 บทสรุป

สถาปัตยกรรมระบบที่มีความพร้อมใช้ระดับสูง ตอบโจทย์ด้านการพัฒนาระบบให้มีการทนต่อความผิดพร่อง (fault tolerance) สถาปัตยกรรมระบบนี้มีความสำคัญเมื่อต้องการรักษาความต่อเนื่องของการให้บริการเมื่อไปใช้งานจริง การเปิดคุณสมบัติความพร้อมใช้ระดับสูงของคิวเบอร์เนเทส ด้วยการมีมาสเตอร์โนดจำนวน 3 โนด และมีรูปแบบฐานข้อมูลจัดเก็บสถานะแบบภายนอก จะทนทานต่อการทำงานผิดพลาดของโนดได้ เมื่อมาสเตอร์โนดใดมีปัญหา ผู้ดูแลระบบสามารถเข้าตรวจสอบความผิดปกติของโนดโดยมีมาสเตอร์โนดอื่นทำงานแทนได้โดยไม่หยุดให้บริการ อย่างไรก็ตามเมื่อระนาบควบคุมหรือคลัสเตอร์จะหยุดทำงาน คอนเทนเนอร์ที่ทำงานอยู่ในโนดใด ๆ จะสามารถทำงานอยู่ได้โดยไม่ขึ้นกับความพร้อมใช้ของคลัสเตอร์ การใช้เทคโนโลยีคิวเบอร์เนเทสร่วมกับเทคโนโลยีคอนเทนเนอร์จะทำให้บริการบนคอนเทนเนอร์มีระดับความพร้อมใช้ดียิ่งขึ้น

บทที่ 6
การออกแบบสถาปัตยกรรมระบบจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร

ในบทนี้สถาปัตยกรรมระบบที่นำเสนอจะให้ความสำคัญด้านความปลอดภัย (security) โดยพิจารณาทั้งด้านความปลอดภัยของกลุ่มผู้ใช้งาน และความปลอดภัยในการเข้าถึงทรัพยากร โดยการจำกัดการเข้าถึงทรัพยากร และการจัดลำดับชั้นทรัพยากรตามความเข้ากันได้ของคอนเทนเนอร์

จากกรณีตัวอย่างของระบบทดสอบไอโอทีแพลทฟอร์มในโครงการ IoTcloudServe@TEIN มีผู้ใช้งานจากหลายฝ่าย เช่น นักพัฒนาโปรแกรม (software developer) และผู้ดูแลระบบ (system admin) จึงมีประเด็นเรื่องการบริหารจัดการผู้ใช้งาน (user management) แยกตามฝ่ายต่าง ๆ ให้การใช้งานมีความปลอดภัย นอกจากนี้การจัดกลุ่มภาระและการจัดลำดับชั้นของทรัพยากร (hierarchical resource) เกี่ยวข้องกับการจัดการเพื่อไม่ให้มีการใช้ทรัพยากรมากเกินไปจนกระทบเสถียรภาพโดยรวมของระบบ และรักษาความเข้ากันได้ของคอนเทนเนอร์และโนดเซิร์ฟเวอร์

ในลำดับถัดไปจะอธิบายประเด็นการจัดกลุ่มภาระ การจัดกลุ่มผู้ใช้งาน โนดเซิร์ฟเวอร์ระดับเอดซ์ นำไปสู่การนำเสนอสถาปัตยกรรมที่ให้ความสำคัญด้านความปลอดภัยในการเข้าถึงทรัพยากร การจัดการผู้ใช้งาน และการจัดลำดับชั้นของทรัพยากร

6.1 การจัดกลุ่มภาระ

คิวเบอร์เนเทสมีการแบ่งกลุ่มของภาระเป็นส่วนประกอบต่าง ๆ ดังนี้

1. หน่วยย่อยที่สุดของภาระคือ คอนเทนเนอร์
2. การจัดกลุ่มคอนเทนเนอร์ในหน่วยเดียวกันเป็น พอด (เทียบเท่าวีเอ็ม 1 เครื่อง และ 1 ไอพีแอดเดรส)
3. การจัดกลุ่มของพอดแบบเดียวกันที่มีได้หลายสำเนาเป็นดีพลอยเมนต์ (deployment)
4. การจัดกลุ่มของพอดภายใต้เนมสเปซ (namespace) เป็นการแยกเดี่ยวทรัพยากร (resource isolation) และใช้จำกัดการเข้าถึงทรัพยากรของกลุ่ม
5. การจัดกลุ่มของเนมสเปซภายใต้โครงการ (project) ซึ่งเป็นการจัดการแบบเฉพาะของรานเชอร์
6. การจัดกลุ่มของโครงการภายใต้คลัสเตอร์ (cluster)

6.2 การจัดกลุ่มผู้ใช้งาน

การจัดกลุ่มผู้ใช้งานช่วยสร้างความปลอดภัย ด้วยการจำกัดการเข้าถึง (accessibility) ในโปรแกรมรานเชอร์มีการจัดกลุ่มผู้ใช้งานดังนี้

1. ผู้ใช้งานระดับโกลบอล (global) มีสิทธิในการ
    - เพิ่มลดคลัสเตอร์
    - จัดการผู้ใช้งานทั้งหมด
    - กำหนดบทบาท (role) ของผู้ใช้งานแต่ละลำดับ
2. ผู้ใช้งานระดับคลัสเตอร์ (cluster) มีสิทธิในการ
    - จัดการโครงการ (project) ภายใต้คลัสเตอร์
    - จัดการโนดในคลัสเตอร์ เช่น การกำหนดป้าย (label) การสั่งให้โนดหยุดทำงาน
    - จัดการการเข้าถึงหน่วยเก็บข้อมูล (storage) ของคลัสเตอร์
    - จัดการสิทธิการเข้าถึงของผู้ใช้งานระดับโครงการ
3. ผู้ใช้งานระดับโครงการ (project) มีสิทธิในการ
    - จัดการเนมสเปซภายในโครงการ
    - จัดการภาระต่าง ๆ ของคิวเบอร์เนเทส เช่น พอด อินเกรซ เซอร์วิซ เป็นต้น
    - จัดการซีไอซีดีไปป์ไลน์ (CI/CD pipeline) (CI ย่อมาจาก continuous integration คือกระบวนการอัตโนมัติในการรวมฟังก์ชันที่พัฒนาใหม่เข้ากับระบบเดิม และ CD ย่อมาจาก continuous deployment คือกระบวนการอัตโนมัติในการดีพลอยระบบ ที่รวมกระบวนการคอนเทนเนอร์ไรซ์เซชันและการรันคอนเทนเนอร์ในขั้นตอนนี้)

การจัดกลุ่มภาระและกลุ่มผู้ใช้งาน ช่วยในการออกแบบนโยบายการปรับใช้คิวเบอร์เนเทสในองค์กร ในการกระจายงานความรับผิดชอบการบริหารโครงการต่าง ๆ ได้

6.3 คิวเบอร์เนเทสของโนดเซิร์ฟเวอร์ระดับเอดช์ หรือ ราสเบอรีพาย

ราสเบอรีพาย (Raspberry pi) เป็นอุปกรณ์คอมพิวเตอร์ขนาดเล็ก ประกอบด้วยหน่วยประมวลผลกลาง และหน่วยความจำในตัว ราสเบอรีพายมีราคาถูกกว่าคอมพิวเตอร์ทั่วไป โดยมีราคาเริ่มต้น 10 เหรียญสหรัฐในโมเดลรุ่นซีโรดับเบิลยู (Zero W) [37] ในงานวิจัยนี้ได้นำราสเบอรีพายมาทดสอบการติดตั้งคิวเบอร์เนเทส เพื่อทดสอบการใช้งานบนสถาปัตยกรรม arm และสามารถรวมศูนย์ควบคุมได้จากระยะไกล ราสเบอรีพายมีสถาปัตยกรรมที่แตกต่างจากคอมพิวเตอร์ทั่วไปในชื่อ อาร์ม (ARM – Advanced RISC Machine) เป็นการออกแบบตามสถาปัตยกรรมริสค์ (RISC – Reduced Instruction Set Computing) ที่เน้นการประหยัดพลังงาน คิวเบอร์เนเทสสามารถติดตั้งบนราสเบอรีพายได้ สเปค คุณสมบัติ และราคาแต่ละรุ่นเป็นดังตารางที่ 17 ในการศึกษาเทคโนโลยีคอนเทนเนอร์และคิวเบอร์เนเทส จำเป็นต้องติดตั้งระบบปฏิบัติการ อัลไพน์ลินุกซ์ (Alpine linux) หรือ ราสเบียน (Raspbian) เพื่อรองรับการติดตั้งคิวเบอร์เนเทส ผลการศึกษาเป็นดังตารางที่ 17 เคสามเอสสามารถติดตั้งบนราสเบอรีพายได้ทั้ง 3 รุ่นที่ศึกษา [38] โดยสามารถกำหนดบทบาทให้เป็นได้ทั้งมาสเตอร์โนดหรือเวิร์คเกอร์โนดได้อย่างใดอย่างหนึ่ง

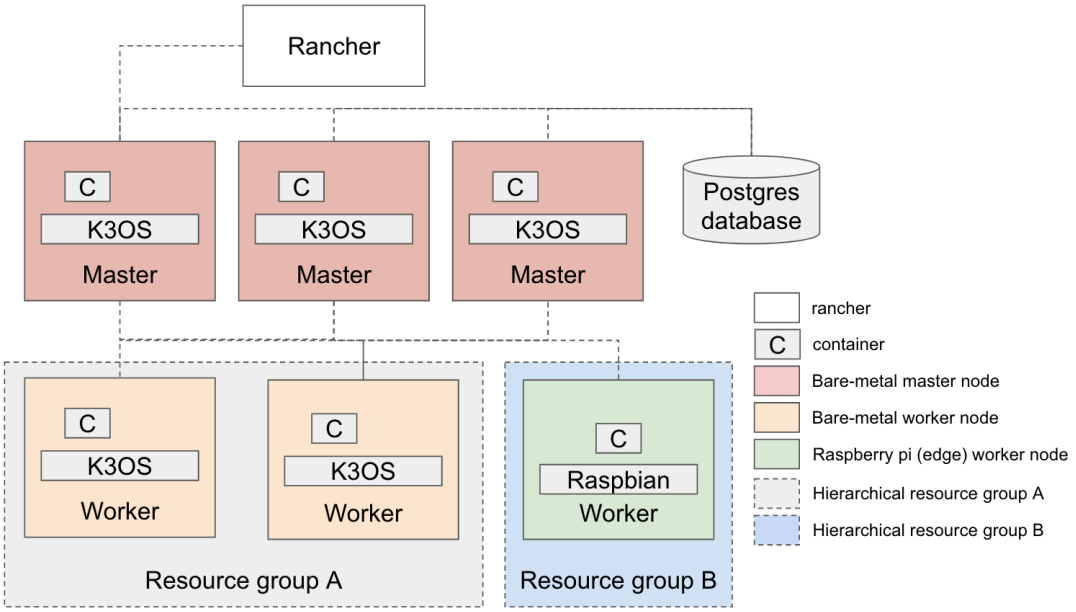
ตารางที่ 17 สเปค ราคา และคุณสมบัติของราสเบอรีพายแต่ละรุ่น

คุณสมบัติ	รุ่น Zero W	รุ่น Zero 2W [39]	รุ่น 4 [40]
หน่วยประมวลผลกลาง (CPU)	Single core 1GHz 32-bit	Quad core A53 1GHz 64-bit	Quad core A72 1.5GHz 64-bit
หน่วยความจำ (memory)	512MB	512MB	1-8 GB
ความจุหน่วยเก็บข้อมูลสูงสุด (disk)	256GB [41]	ไม่ระบุ	ไม่ระบุ
ราคาเริ่มต้น	10$	15$	35$

ตารางที่ 18 ผลการศึกษาการติดตั้งระบบปฏิบัติการต่าง ๆ บนราสเบอรีพาย

รุ่นของระบบปฏิบัติการ	รุ่น Zero W	รุ่น Zero 2W	รุ่น 4B
Alpine linux (armv7, armhf)	รองรับ	รองรับ	รองรับ
Alpine linux (aarch64)	ไม่รองรับ	รองรับ	รองรับ
Raspbian 32x	รองรับ	รองรับ	รองรับ
Raspbian 64x	ไม่รองรับ	รองรับ	รองรับ

6.4 สถาปัตยกรรมระบบที่นำเสนอ



รูปที่ 34 สถาปัตยกรรมระบบที่มีการจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร

รูปที่ 34 แสดงสถาปัตยกรรมระบบที่มีจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร โดยมีการติดตั้งคิวเบอร์เนเทสเพิ่มเติมลงบนอุปกรณ์ราสเบอรีพาย ซึ่งมีสถาปัตยกรรมเฉพาะคือ arm64 สถาปัตยกรรมระบบนี้ใช้โปรแกรมรานเชอร์ในการบริหารผู้ใช้งาน ส่วนในการจัดลำดับชั้นของทรัพยากร จะมีการกำหนดกลุ่มทรัพยากร (resource group) เป็นกลุ่ม A และ B รายการติดตั้งตามสถาปัตยกรรมของโนดเป็นดังตารางที่ 19 โดยมีวัตถุประสงค์เพื่อกำหนดกลุ่ม A เป็นกลุ่มโนดที่ใช้สถาปัตยกรรมทั่วไปคือ amd64 และกลุ่ม B เป็นกลุ่มโนดที่ใช้สถาปัตยกรรมเฉพาะคือ arm64

ตารางที่ 19 รายการติดตั้งคิวเบอร์เนเทสตามสถาปัตยกรรมระบบโครงการไอโอทีคลาวด์เซิร์ฟที่มีราสเบอรีพาย

Hostname	IP-Address	OS	Architecture	Role	Container engine	Resource group
Master1	202.28.193.102	K3OS	amd64	Master	containerd	-
Master2	202.28.193.100	K3OS	amd64	Master	containerd	-
Master3	202.28.193.101	K3OS	amd64	Master	containerd	-
k3os-2030	161.200.90.106	K3OS	amd64	Worker	containerd	A
k3os-4689	161.200.90.110	K3OS	amd64	Worker	containerd	A
rpb4-w1	192.168.0.159	Raspbian	arm64	Worker	docker	B

6.5 การทดสอบสถาปัตยกรรมระบบ

การทดสอบสถาปัตยกรรมระบบ จะขอยกตัวอย่างสถานการณ์ดังนี้ นายเบนซ์เป็นนักพัฒนาโปรแกรมโครงการ CU-BEMS นายแบงค์เป็นนักพัฒนาโปรแกรมสถานีไฟฟ้าอัจฉริยะ และนายบาสเป็นนักวิเคราะห์แผนธุรกิจ ที่ต้องการข้อมูลจากทั้ง 2 โครงการไปนำเสนอ ทั้ง 3 คนนี้มีสิทธิในการเข้าถึงโครงการไม่เหมือนกัน กำหนดดังตารางที่ 20 โดยนายบาสจะสามารถเข้าดูภาระได้ทั้ง 2 โครงการ แต่ไม่สามารถปรับแก้ภาระใด ๆ ได้ การทดสอบและผลการทดสอบการจัดการผู้ใช้งานบนรานเชอร์รวบรวมไว้ในภาคผนวก ผลการทดสอบสามารถสร้างสิทธิผู้ใช้งานตามที่กำหนดนี้ได้

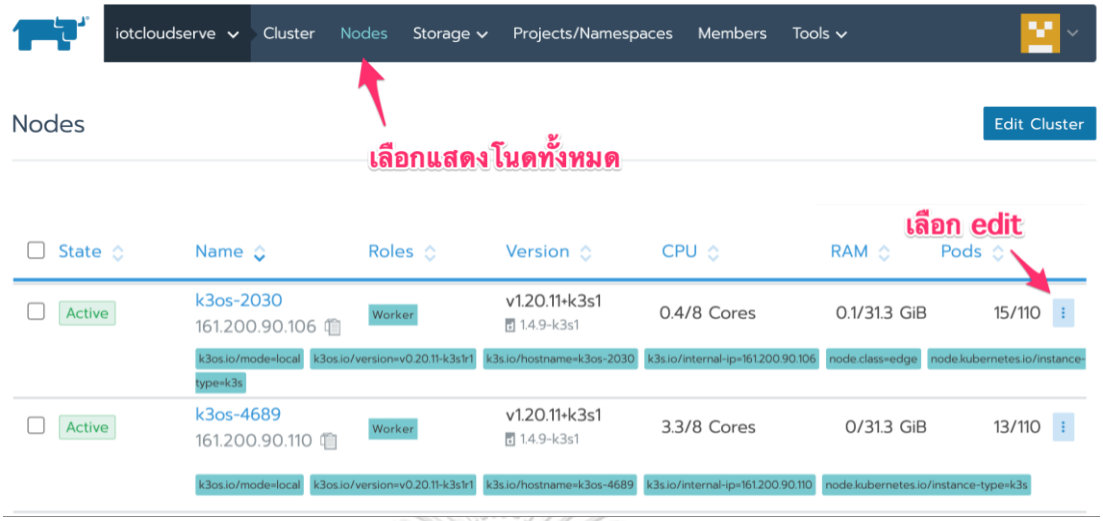
ตารางที่ 20 ตัวอย่างสิทธิผู้ใช้งานแต่ละโครงการ

โครงการ	สิทธิผู้ใช้งาน		
	นายเบนซ์	นายแบงค์	นายบาส
CU-BEMS	บำรุงรักษาได้ทั้งหมด	ไม่มีสิทธิ	ดู (view) ได้อย่างเดียว
สถานีไฟฟ้าอัจฉริยะ	ไม่มีสิทธิ	บำรุงรักษาได้ทั้งหมด	ดู (view) ได้อย่างเดียว

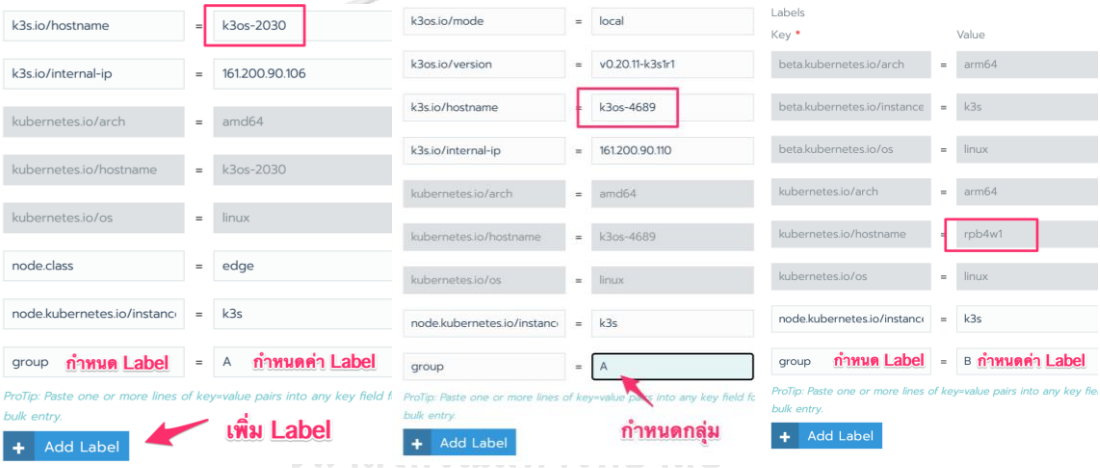
การสร้างคอนเทนเนอร์บนคลัสเตอร์มีหลายกรณีที่ต้องการทรัพยากรที่เจาะจงเช่น สถาปัตยกรรม amd64 และ arm64 โดยปกติสถาปัตยกรรม amd64 เป็นสถาปัตยกรรมของเซิร์ฟเวอร์ทั่วไป ส่วน arm64 เป็นสถาปัตยกรรมที่นิยมใช้บนอุปกรณ์เคลื่อนที่ทั่วไปเช่น โทรศัพท์มือถือ คิวเบอร์เนเทสรองรับการกำหนดกลุ่มของทรัพยากรที่เจาะจงในการคัดเลือกโนดที่รันเฉพาะในกลุ่มที่กำหนดได้ ในกรณีนี้จะกำหนดกลุ่มของทรัพยากร 2 กลุ่มดังนี้

1. กลุ่ม A หรือกลุ่มสถาปัตยกรรม amd64 จะกำหนดบนเวิร์กเกอร์โนด k3os-2020 และ k3os-4689
2. กลุ่ม B หรือกลุ่มสถาปัตยกรรม arm64 จะกำหนดบนเวิร์กเกอร์โนด rpb4-w1

การกำหนดกลุ่มของทรัพยากรทำได้โดยการติดป้าย (label) ให้โนดเพื่อให้คิวเบอร์เนเทสตรวจสอบการรันคอนเทนเนอร์เจาะจงบนกลุ่มโนดที่ต้องการ รูปที่ 35 แสดงการเข้าถึงการแก้ไขป้ายของแต่ละโนดบนรานเชอร์ ทำโดยเข้าสู่หน้าแสดงรายการโนดทั้งหมด และคลิกเลือกแก้ไขบนโนดที่ต้องการแก้ไขป้าย รูปที่ 36 แสดงการกำหนดป้ายบนโนด k3os-2030 k3os-4689 จัดเป็นกลุ่ม A และโนดราสเบอรีพาย rpb4w1 จัดเป็นกลุ่ม B การกำหนดป้ายบนโนดเพื่อการจัดกลุ่มของทรัพยากรช่วยในการจัดลำดับชั้นของทรัพยากรที่มีการเจาะจงการรันคอนเทนเนอร์ เช่น ตามสถาปัตยกรรม ตามระดับความสำคัญ ตามระดับความพร้อมใช้เป็นต้น การทดสอบการรันคอนเทนเนอร์บนโนดที่เข้ากันได้รวบรวมไว้ในภาคผนวก



รูปที่ 35 การเข้าถึงการแก้ไขป้าย (label) ของแต่ละโนด



รูปที่ 36 การกำหนดป้ายบนโนด k3os-2030 k3os-4689 และโนดราสเบอรีพาย rpb4w1

6.6 การประยุกต์ในโครงการ IoTcloudServe@TEIN และ OF@TEIN+++

โครงการ IoTcloudServe@TEIN ได้ประยุกต์การจัดการผู้ใช้งานและบริหารโครงการที่หลากหลาย เช่นโครงการ CU-BEMS ในการวิเคราะห์ข้อมูลพลังงานอัจฉริยะ โครงการ OF@TEIN++ ในการศึกษาการร่วมสมาพันธ์คลัสเตอร์ โครงการ Kubeflow ในการพัฒนา Machine Learning pipeline โครงการ SUMO ในการศึกษาแบบจำลองด้านวิศวกรรมจราจร และอื่น ๆ เป็นต้น โดยแต่ละโครงการได้กำหนดสิทธิเข้าถึงโดยกลุ่มผู้ใช้งานต่างกันเพื่อความปลอดภัย

โครงการ OF@TEIN+++ ได้ศึกษาและพัฒนาแนวทางการจัดลำดับชั้นของทรัพยากรเพื่อความปลอดภัย เป็นการประมวลผลระดับคลาวด์ และระดับเอดช์ภายในคลัสเตอร์เดียวกัน โดยในคลัสเตอร์ได้เชื่อมต่อกับราสเบอรีพายซึ่งเป็นโนดที่มีสถาปัตยกรรม arm64 ให้สามารถทำงานร่วมกันกับโนดอื่นในคลัสเตอร์ได้ คุณสมบัตินี้เป็นการจัดลำดับชั้นของทรัพยากรโดยการใช้คุณสมบัติการติดป้าย ในการกำหนดเงื่อนไขการรันคอนเทนเนอร์บนภาระ ทำให้สามารถการันตีความเข้ากันได้และกระจายคอนเทนเนอร์ไปแต่ละโนดได้อย่างถูกต้อง เป็นการจัดลำดับชั้นของทรัพยากรได้อย่างปลอดภัย

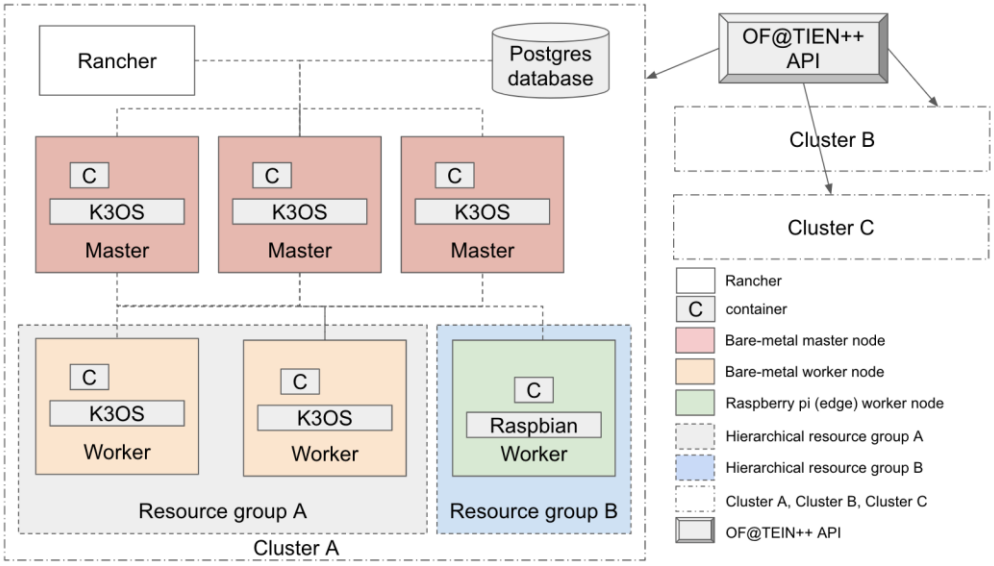
6.7 บทสรุป

สถาปัตยกรรมระบบในการจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร ได้ให้ความสำคัญด้านความมั่นคงปลอดภัยในการจำกัดสิทธิของกลุ่มผู้ใช้งานที่เข้าถึงโครงการต่าง ๆ สถาปัตยกรรมระบบนี้จะสามารถรองรับผู้ใช้งานที่หลากหลาย และความหลากหลายของโนดเซิร์ฟเวอร์ การออกแบบการติดป้ายให้กับโนด จะสามารถนำไปประยุกต์ใช้ได้หลากหลายเช่น การแบ่งกลุ่มโนดตามระดับความพร้อมใช้ ความหลากหลายของสถาปัตยกรรม การกำหนดให้รันในโนดที่มีชิปประมวลผลกราฟฟิก เป็นต้น

บทที่ 7
การออกแบบสถาปัตยกรรมระบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์

นอกเหนือจากในบริบทของการทำงานของคอนเทนเนอร์ในคลัสเตอร์เดียวกัน มีกรณีการสร้างการร่วมสมาพันธ์คลัสเตอร์ (cluster federation) คือการสร้างคอนเทนเนอร์ออร์เคสเตรชันแบบหลายคลัสเตอร์ ซึ่งมีคุณประโยชน์ดังที่ระบุในหัวข้อ 1.1.5 ได้แก่ กรณีความจุล้นที่มีความต้องการใช้ทรัพยากรชั่วคราวอย่างเร่งด่วน กรณีระบบงานที่อ่อนไหวที่ต้องการย้ายภาระข้ามคลัสเตอร์ตามข้อกำหนดที่เปลี่ยนแปลง กรณีการหลีกเลี่ยงการผูกขาดของผู้ให้บริการ และกรณีความพร้อมใช้งานระดับสูงในเชิงการกระจายทางภูมิศาสตร์ ในบทนี้สถาปัตยกรรมระบบที่นำเสนอจะมีการร่วมสมาพันธ์คลัสเตอร์ที่รองรับกรณีความจุล้นและกรณีความพร้อมใช้ระดับสูงในเชิงการกระจายทางภูมิศาสตร์ ซึ่งทำให้ระดับความพร้อมใช้ของบริการสูงยิ่งขึ้น

7.1 สถาปัตยกรรมระบบที่นำเสนอ



รูปที่ 37 สถาปัตยกรรมระบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์ออร์เคสเตรตคอนเทนเนอร์

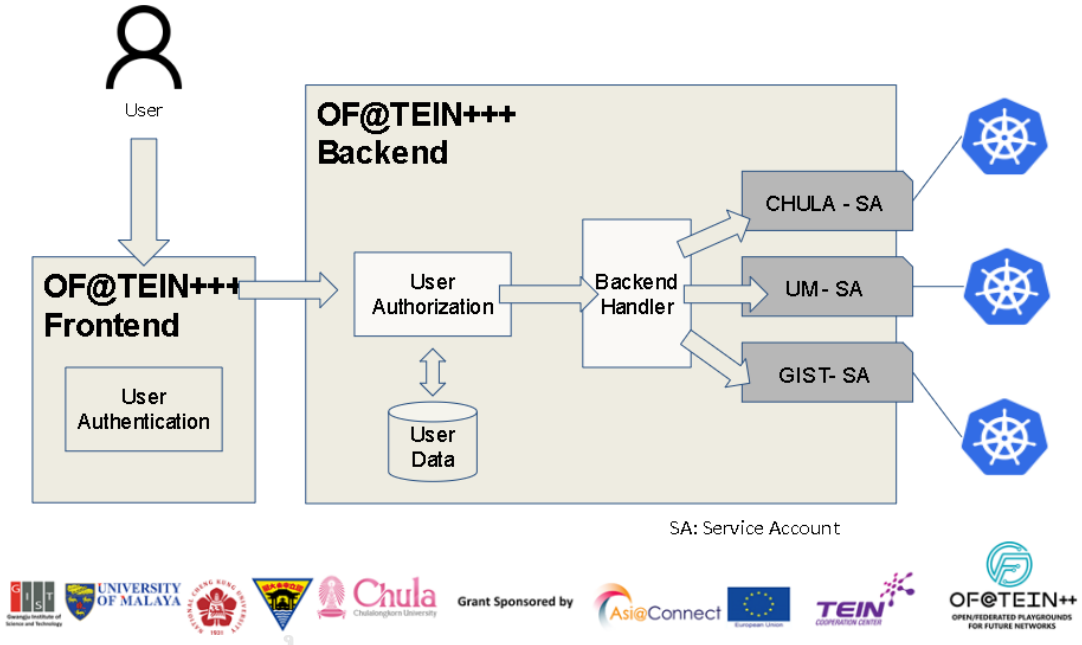
รูปที่ 37 แสดงสถาปัตยกรรมระบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์ออร์เคสเตรตคอนเทนเนอร์ โดยมีคลัสเตอร์ A เป็นคลัสเตอร์ที่ได้จากการต่อยอดในสถาปัตยกรรมระบบในแบบต่าง ๆ ที่นำเสนอมา โดยในโครงสร้างนี้จะมีคลัสเตอร์ B และคลัสเตอร์ C ที่ไม่ระบุโครงสร้างสถาปัตยกรรมระบบภายใน แต่ต้องการเน้นในจุดการร่วมสมาพันธ์คือการใช้ทรัพยากรระบบข้ามคลัสเตอร์กัน โดยพัฒนาต่อยอดจากเครื่องมือที่ประยุกต์ใช้จะจากโครงการ OF@TEIN++ ข้อมูลการติดตั้งแต่ละโนดในคลัสเตอร์ A ใช้การติดตั้งดังสถาปัตยกรรมระบบที่ผ่านมาและมีรายการติดตั้งตามตารางที่ 19 โดยสร้างคิวเบอร์เนเทสคลัสเตอร์ B ที่เพียงพอในการทดสอบการร่วมสมาพันธ์คลัสเตอร์

7.2 โครงการ OF@TEIN++

โครงการ OF@TEIN++ ในความร่วมมือจากผู้ดูแลระบบหลายกลุ่มจากหลายประเทศ ในการสร้างการร่วมสมาพันธ์ระหว่างคิวเบอร์เนเทสคลัสเตอร์ (Kubernetes cluster federation) ให้สามารถใช้ทรัพยากรคิวเบอร์เนเทสข้ามคลัสเตอร์กันได้ โครงการ OF@TEIN++ เป็นโครงการต่อยอดจากโครงการ IoTcloudServe@TEIN เพื่อควบรวมคิวเบอร์เนเทสคลัสเตอร์ของแต่ละหน่วยงานเข้าด้วยกัน ตามแนวทางการร่วมสมาพันธ์ (federation) โดยมีคลัสเตอร์ของมหาวิทยาลัย GIST จากประเทศเกาหลีใต้ คลัสเตอร์ของมหาวิทยาลัย UM จากประเทศมาเลเซีย และคลัสเตอร์ของจุฬาลงกรณ์มหาวิทยาลัย จากประเทศไทย โดยตัวแทนจุฬาฯ ได้ร่วมพัฒนากลไกการร่วมสมาพันธ์ดังกล่าว การเปิดคุณสมบัติการร่วมสมาพันธ์คลัสเตอร์โดยมากปรากฏในบริบทของผู้ดูแลระบบในหน่วยงานเดียวกัน การจัดการคิวเบอร์เนเทสหลายคลัสเตอร์โดยผู้ดูแลระบบจึงมีสิทธิสูงสุดครบถ้วนในทุกคลัสเตอร์ แต่ในบริบทของโครงการ OF@TEIN++ มีข้อจำกัดคือมีการมุ่งหวังให้มีการใช้ทรัพยากรข้ามกลุ่มกันโดยอนุญาตจำกัดเฉพาะทรัพยากรบางส่วนของคลัสเตอร์ที่กำหนดไว้ให้ โดยมีผู้ดูแลระบบต่างกลุ่มบริหารจัดการทรัพยากรของตนแยกส่วนกันไป จุดนี้จึงเป็นความท้าทายที่ต่างจากการเปิดคุณสมบัติการร่วมสมาพันธ์คิวเบอร์เนเทสคลัสเตอร์ที่แตกต่างโดยทั่วไป กลไกการออกแบบจึงมีลักษณะกระจายการบริหารจัดการทรัพยากรระหว่างคลัสเตอร์ให้สามารถจัดการโดยเจ้าของคลัสเตอร์แบบเอกภาพได้ โดยสร้างระบบศูนย์กลางในการกระจายภาระงานไปคลัสเตอร์ที่ร่วมสมาพันธ์ที่มีการบริหารจัดการร่วมกันโดยทุกกลุ่มผู้ดูแลระบบ และระบบศูนย์กลางนี้จะไม่กระทบกับนโยบายการปันทรัพยากรให้กับคลัสเตอร์ที่ร่วมสมาพันธ์ ข้อกำหนดนี้สามารถปรับใช้ในบริบทของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย ที่มีผู้ดูแลระบบหลายกลุ่มกระจายไปแต่ละภูมิภาค การขอใช้ทรัพยากรข้ามคลัสเตอร์ต่างกลุ่มผู้ให้บริการจะสามารถเรียกใช้ผ่านระบบกลางได้ นอกจากนี้สามารถประยุกต์ใช้ในกรณีตัวอย่างเช่นระบบจัดการพลังงานอัจฉริยะของแต่ละสถานีไฟฟ้าย่อย ที่สามารถรวมศูนย์การจัดการแอพลิเคชันพลังงานอัจฉริยะในการอัพเกรดหรือย้อนเวอร์ชันจากคอนเทนเนอร์อิมเมจกลางได้สะดวก และสามารถยืมทรัพยากรข้ามกลุ่มเพื่อการประมวลผลชั่วคราวได้เช่นกัน

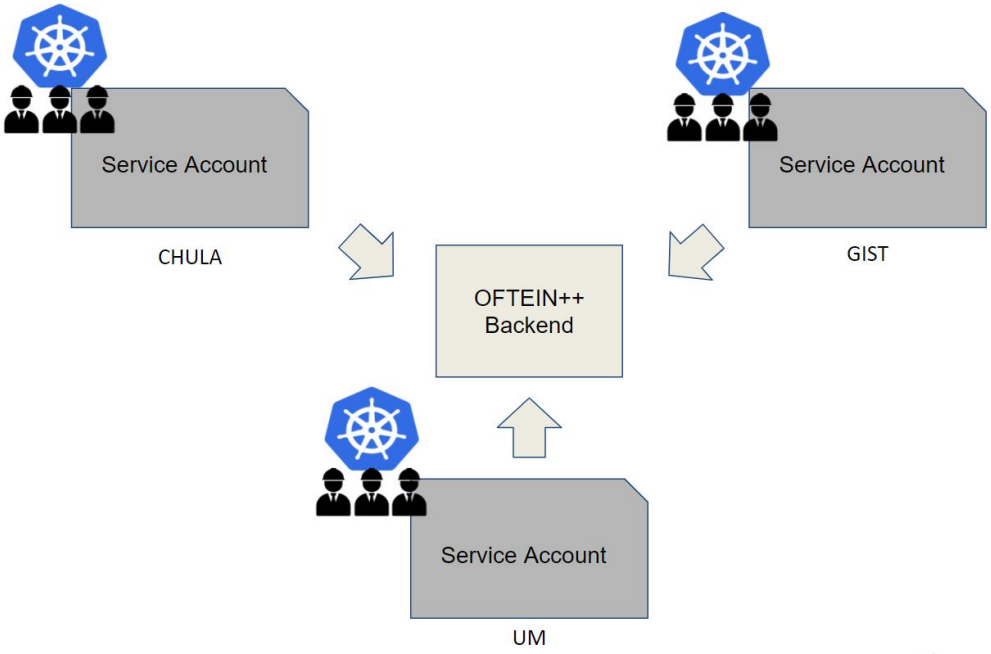
7.2.1 แนวคิดการออกแบบ

ระบบ OF@TEIN++ เป็นการเปิดคุณสมบัติการร่วมสมาพันธ์คิวเบอร์เนเทสคลัสเตอร์ โดยการออกแบบลักษณะรวมศูนย์ มีการใช้คุณสมบัติของเนมสเปซในการจำกัดโควตาการเข้าถึงทรัพยากรและกำหนดเซอร์วิสแอคเคาท์ (service account) เพื่อควบคุมสิทธิการเข้าถึงทรัพยากรคิวเบอร์เนเทส ระบบ OF@TEIN++ แบ่งระบบงานเป็น 2 ส่วนได้แก่ ส่วนระบบหน้าบ้าน (frontend) ดูแลการยืนยันตัวตนและส่วนเชื่อมต่อกับผู้ใช้ ส่วนระบบหลังบ้าน (backend) ดูแลการตรวจสอบสิทธิการใช้งานและการสั่งการคิวเบอร์เนเทสคลัสเตอร์ที่ร่วมสมาพันธ์ รูปที่ 38 แสดงแนวคิดการออกแบบระบบ OF@TEIN++



รูปที่ 38 แนวคิดการออกแบบระบบ OF@TEIN++

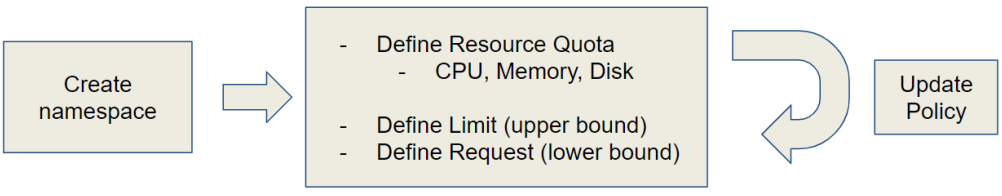
เซอร์วิสแอคเคาท์ คือสิ่งที่ใช้กำหนดขอบเขตการเข้าถึงทรัพยากรที่ผู้ดูแลระบบกำหนด โดยสามารถสร้างใหม่ หรือเพิกถอนเซอร์วิสแอคเคาท์ที่สร้างไปแล้วได้ เซอร์วิสแอคเคาท์ที่สร้างจากทุกคลัสเตอร์ที่ร่วมสมาพันธ์ จะรวมไว้ที่ระบบหลังบ้าน (รูปที่ 39) โดยระบบจะพิจารณาเลือกเซอร์วิสแอคเคาท์ที่รวมกันและนำไปใช้เมื่อมีผู้ใช้ของ OF@TEIN++ ร้องขอใช้งาน ดังนั้นผู้ใช้งาน OF@TEIN++ จะต้องผ่านการยืนยันตัวตน และได้รับอนุญาตให้สามารถใช้งานคลัสเตอร์ที่ร่วมสมาพันธ์ก่อนใช้งานได้



รูปที่ 39 การแชร์เซอร์วิสแอคเคาท์

จากรูปที่ 40 เซอร์วิสแอคเคาท์ที่แชร์มาศูนย์กลาง จะผูกโยงกับเนมสเปซของคิวเบอร์เนเทส ซึ่งสามารถจำกัดโควตาทรัพยากรต่าง ๆ ได้ โดยผู้ดูแลระบบสามารถควบคุมโควตาและแก้ไขนโยบายการร่วมสมาพันธ์ได้ตลอดโดยไม่ต้องเปลี่ยนเซอร์วิสแอคเคาท์ รูปที่ 40 แสดงกิจกรรมการควบคุมโควตาทรัพยากรด้วยเนมสเปซที่แต่ละคลัสเตอร์ในโครงการมีการจัดการทรัพยากรคลัสเตอร์แยกกัน

Cluster	Namespace
Chula	chula-ofteinplusplus-fedns
UM	um-ofteinplusplus-fedns
GIST	gist-ofteinplusplus-fedns

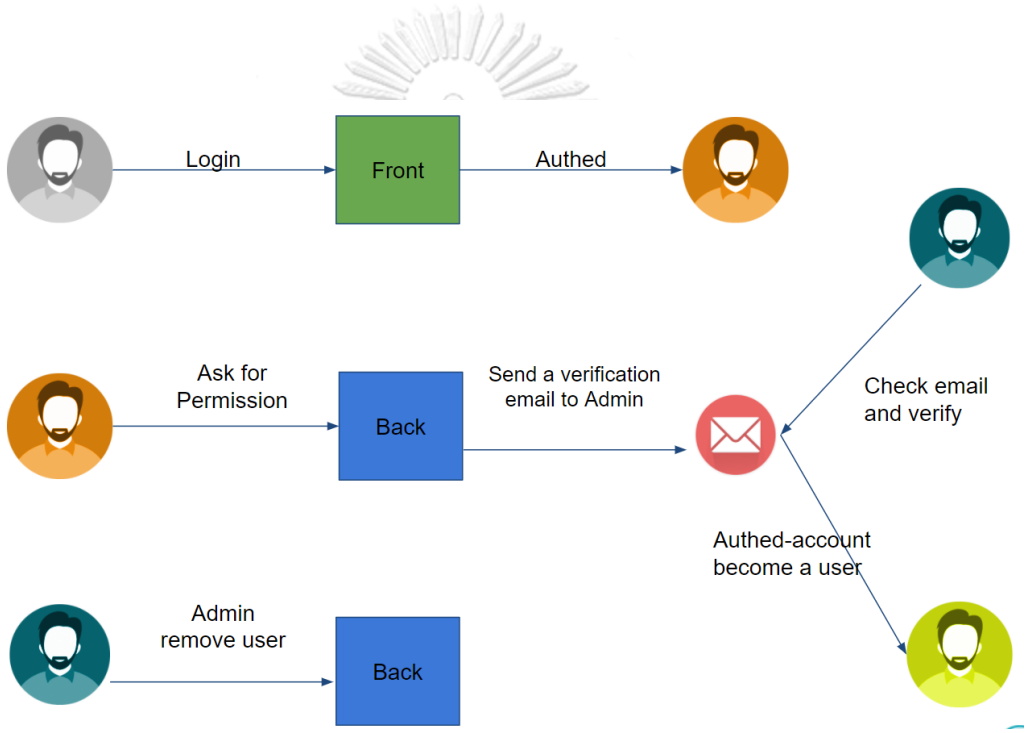


รูปที่ 40 การควบคุมโควตาทรัพยากรด้วยเนมสเปซ

7.2.2 ระบบจัดการผู้ใช้งาน

ผู้ใช้ระบบ OF@TEIN++ แบ่งออกเป็น 4 กลุ่มดังนี้ (รูปที่ 41)

1. กลุ่มผู้ที่ไม่ผ่านการยืนยันตัวตน (สีเทา) ไม่มีสิทธิในการใช้งานใด ๆ
2. กลุ่มผู้ที่ผ่านการยืนยันตัวตนแล้ว แต่ไม่ได้รับอนุญาตให้ใช้งาน (สีส้ม) ไม่มีสิทธิใช้งาน ต้องสร้างคำขอต่อผู้ดูแลระบบ และรอให้ผู้ดูแลระบบอนุญาตการใช้งาน
3. กลุ่มผู้ที่ผ่านการยืนยันตัวตนแล้ว และได้รับอนุญาตให้ใช้งาน (สีเขียว) สามารถใช้งานระบบได้
4. กลุ่มผู้ดูแลระบบ OF@TEIN++ (สีน้ำเงิน) สามารถใช้งานระบบได้และบริหารผู้ใช้งานได้



รูปที่ 41 การออกแบบระบบจัดการผู้ใช้งาน

7.2.3 การพัฒนาระบบหลังบ้าน

เริ่มจากการทดสอบเปิดคุณสมบัติการร่วมสมาพันธ์คิวเบอร์เนเทส (Kubernetes cluster federation enabler) โดยวิทยานิพนธ์นี้ได้พัฒนา API รุ่นที่ 1 ควบคุมทรัพยากร 2 ประเภทคือ พอดและดีพลอยเมนต์ มีผลการพัฒนาดังนี้

ตารางที่ 21 API ระบบหลังบ้าน OF@TEIN++ รุ่นที่ 1

Method	URL	Description	Payload	Params
GET	`host` /clusters	view all clusters	-	-
GET	`host` /clusters/ `cluster`	view cluster	-	`cluster`
GET	`host` /clusters/ `cluster` /pods	view all pods	-	`cluster`
POST	`host` /clusters/ `cluster` /pods	create a pod	`yaml`	`cluster`
GET	`host` /clusters/ `cluster` /pods/ `pod`	view pod	-	`cluster` `pod`
DELETE	`host` /clusters/ `cluster` /pods/ `pod`	delete pod	-	`cluster` `pod`
GET	`host` /clusters/ `cluster` /deployments	view all deployments	-	`cluster`
POST	`host` /clusters/ `cluster` /deployments	create a deployment	`yaml`	`cluster`
GET	`host` /clusters/ `cluster` /deployments/ `deployment`	view deployment	-	`cluster` `deployment`
DELETE	`host` /clusters/ `cluster` /deployments/ `deployment`	delete deployment	-	`cluster` `deployment`

API รุ่นที่ 1 ประสบความสำเร็จในการออร์เคสเตรตคอนเทนเนอร์ไปในหลายคลัสเตอร์ ตารางที่ 21 แสดงรายการ API OF@TEIN++ รุ่นที่ 1 ดังนั้นวิทยานิพนธ์นี้จึงต่อยอดการพัฒนาไปสู่ API รุ่นที่ 2 ในการเชื่อมโยงภาระที่นำขึ้นเข้ากับหมายเลขผู้ใช้งาน เพื่อให้สามารถจำแนกภาระต่าง ๆ เฉพาะตามหมายเลขผู้ใช้งานได้ ผลการพัฒนาเป็นดังตารางที่ 22

ตารางที่ 22 API ระบบหลังบ้าน OF@TEIN++ รุ่นที่ 2

Method	URL	Description	Payload	Params	Query
GET	`host` /v2/pods	view all pods	-	-	`userid` `cluster`
POST	`host` /v2/pods	create a pod	`yaml`	-	`cluster` `userid`
DELETE	`host` /v2/pods	delete specific pod	-	-	`cluster` `userid` `name`
GET	`host` /v2/deployments	view all deployments	-	-	`userid` `cluster`
POST	`host` /v2/deployments	create a deployment	`yaml`	-	`cluster` `userid`
DELETE	`host` /v2/deployments	delete specific deployment	-	-	`cluster` `userid` `name`

จากนั้นวิทยานิพนธ์นี้จึงพัฒนาสู่ API รุ่นที่ 3 ในการรองรับภาระที่หลากหลายขึ้นได้แก่ เซอร์วิสและอินเกรซ และเพิ่มคุณสมบัติการเลือกคลัสเตอร์เป้าหมายอัตโนมัติตามปริมาณทรัพยากรที่ขอและทรัพยากรที่เหลือแต่ละคลัสเตอร์ นอกจากนี้ได้พัฒนาส่วนการจัดการสิทธิผู้ใช้งาน (authorization management) โดยกลุ่มผู้ดูแลระบบ OF@TEIN++ เพื่อให้สามารถปรับปรุงสิทธิผู้ใช้งานได้ ผลการพัฒนารุ่นที่ 3 เป็นดังตารางที่ 23 แสดง API ระบบหลังบ้าน OF@TEIN++ รุ่นที่ 3 และตารางที่ 24 แสดง API ระบบหลังบ้าน OF@TEIN++ ในการปรับปรุงสิทธิผู้ใช้งาน

ตารางที่ 23 API ระบบหลังบ้าน OF@TEIN++ รุ่นที่ 3

Method	Auth-Level	URL	Description	Payload	Params	Query
GET	`admin` `user`	`host` /v3/pod	view all pods	-	-	`cluster` `token`
POST	`admin` `user`	`host` /v3/pod	create a pod	`yaml`	-	`cluster` `token`
DELETE	`admin` `user`	`host` /v3/pod	delete specific pod	-	-	`cluster` `token` `name`
GET	`admin` `user`	`host` /v3/deployment	view all deployments	-	-	`cluster` `token`
POST	`admin` `user`	`host` /v3/deployment	create a deployment	`yaml`	-	`cluster` `token`
DELETE	`admin` `user`	`host` /v3/deployment	delete specific deployment	-	-	`cluster` `token` `name`
GET	`admin` `user`	`host` /v3/service	view all services	-	-	`cluster` `token`
POST	`admin` `user`	`host` /v3/service	create a service	`yaml`	-	`cluster` `token`
DELETE	`admin` `user`	`host` /v3/service	delete specific service	-	-	`cluster` `token` `name`
GET	`admin` `user`	`host` /v3/ingress	view all ingress(es)	-	-	`cluster` `token`
POST	`admin` `user`	`host` /v3/ingress	create an ingress	`yaml`	-	`cluster` `token`
DELETE	`admin` `user`	`host` /v3/ingress	delete specific ingress	-	-	`cluster` `token` `name`

ตารางที่ 24 API ระบบหลังบ้าน OF@TEIN++ ในการปรับปรุงสิทธิผู้ใช้งาน

A new user asking for permission

Method	Auth-Level	URL	Description	Payload	Params	Query	Remark
GET	`authed` `admin` `user`	`host` /admins	• to list all admins • to let authed-user selecting admin	-	-	`page` `limit` `token`	`page` and `limit` are not required
GET	`authed`	`host` /sendemail	`authed` asks for permission	-	-	`admin` `token`	`admin` must be admin-email selected from `/admins`

- Verification link will be valid for 7 days.

Admin listing all users and promoting by himself

Method	Auth-Level	URL	Description	Payload	Params	Query	Remark
GET	`admin`	`host` /users	to list all users for maintenance	-	-	`page` `limit` `token`	`page` and `limit` are not required
GET	`admin`	`host` /promoteaccount	promote specific account	-	-	`email` `admin` `token`	`admin` must be boolean in order to set to be admin or not (default is false) and `email` must be user-email
GET	`admin`	`host` /demoteaccount	demote specific account	-	-	`email` `token`	`email` must be user-email

- `admin` could add accounts directly via `/promoteaccount`

7.2.4 การเปรียบเทียบคุณสมบัติระบบ OF@TEIN++ กับเครื่องมือในงานวิจัยอื่น

ระบบ OF@TEIN++ สามารถให้บริการระหว่างคิวเบอร์เนเทสคลัสเตอร์โดยผู้ดูแลระบบต่างกลุ่มได้ การเปรียบเทียบคุณสมบัติในการร่วมสมาพันธ์ของคิวเบเนเทสคลัสเตอร์แต่ละเครื่องมือ เป็นดังตารางที่ 25 ดังนี้

ตารางที่ 25 การเปรียบเทียบคุณสมบัติระบบที่พัฒนาในวิทยานิพนธ์ร่วมกับเครื่องมือในงานวิจัยอื่น

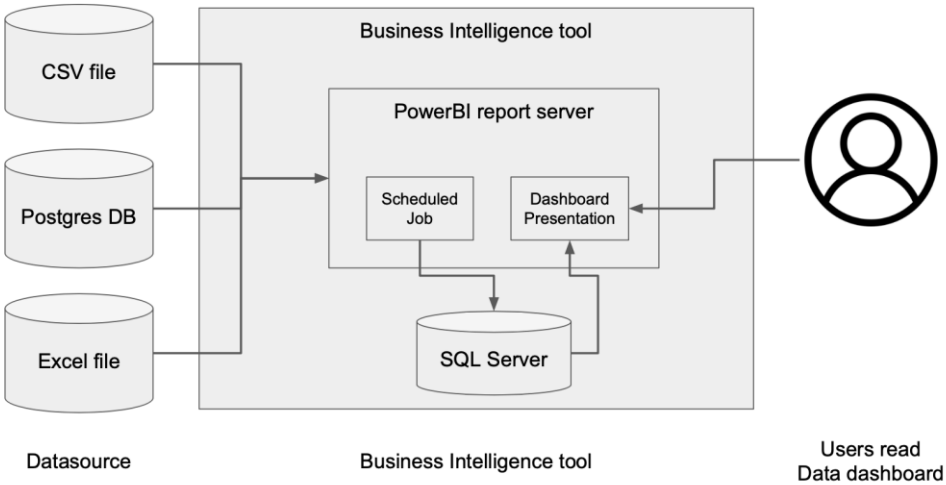
เครื่องมือ	การจัดการภาระ	การกระจายภาระข้ามคลัสเตอร์	การจำกัดทรัพยากรระหว่างคลัสเตอร์	ความเป็นเจ้าของคลัสเตอร์	การบริหารกลุ่มผู้ใช้งาน	รูปแบบของคลาวด์
Kubefed	รวมศูนย์	มี	มี	กลุ่มเดี่ยว	ไม่มี	Federation
Rancher	รวมศูนย์	ไม่มี	ไม่มี	กลุ่มเดี่ยว	มี	Multi-cluster
Paper [20]	รวมศูนย์	มี	ไม่มี	กลุ่มเดี่ยว	ไม่มี	Federation
Paper [22]	เชิงกระจาย	มี	ไม่มี	กลุ่มเดี่ยว	ไม่มี	Federation
Paper [24]	เชิงกระจาย	ไม่มี	ไม่มี	กลุ่มเดี่ยว	ไม่มี	Orchestrator
วิทยานิพนธ์นี้	**รวมศูนย์**	**มี**	**มี**	**หลายกลุ่ม**	**มี**	**Federation**

ตารางที่ 25 แสดงการเปรียบเทียบคุณสมบัติระบบที่พัฒนาในวิทยานิพนธ์ร่วมกับเครื่องมือในงานวิจัยอื่น สำหรับรูปแบบการจัดการภาระ ในงานวิจัย [22] และ [24] จะเป็นแบบเชิงกระจาย ส่วนของวิทยานิพนธ์ คิวเฟ็ด รานเชอร์และงานวิจัย [20] จะเป็นแบบรวมศูนย์ การกระจายภาระข้ามคลัสเตอร์ ไม่พบการกระจายภาระข้ามคลัสเตอร์ในรานเชอร์และในงานวิจัย [24] ส่วนการจำกัดทรัพยากรระหว่างคลัสเตอร์ มีเฉพาะแบบคิวเฟ็ดและแบบที่พัฒนาในวิทยานิพนธ์นี้

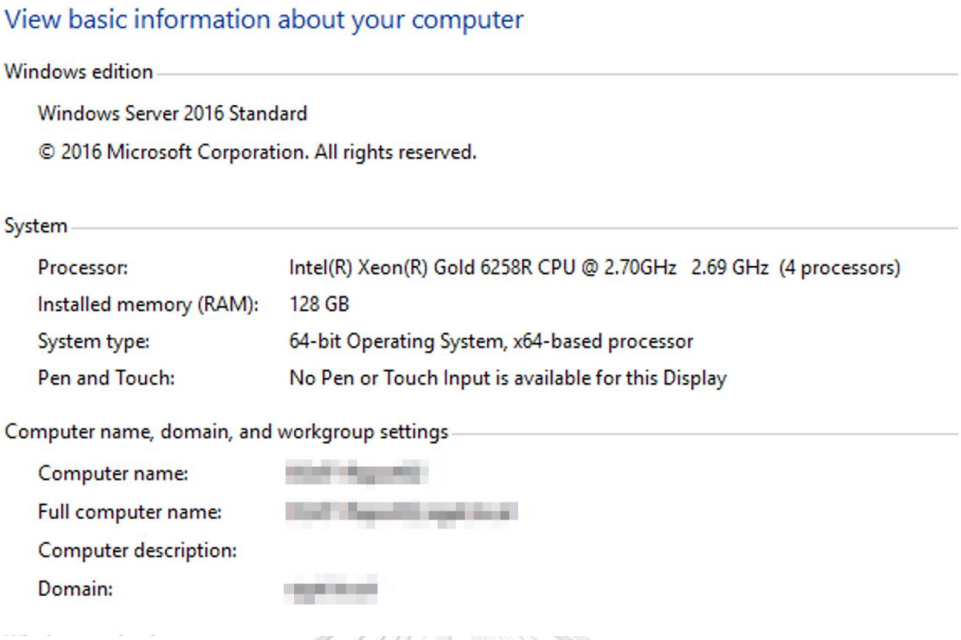
สำหรับรูปแบบความเป็นเจ้าของคลัสเตอร์ในการใช้ทรัพยากรข้ามคลัสเตอร์กัน เกือบทั้งหมดเป็นแบบกลุ่มเดี่ยว เฉพาะแบบที่นำเสนอในวิทยานิพนธ์เป็นแบบหลายกลุ่ม การบริหารกลุ่มผู้ใช้งานมีเฉพาะแบบรานเชอร์และแบบที่พัฒนาในวิทยานิพนธ์นี้เช่นกัน และมิติสุดท้ายรูปแบบของคลาวด์เฉพาะแบบคิวเฟ็ด งานวิจัย [20] งานวิจัย [22] และรูปแบบที่นำเสนอในวิทยานิพนธ์นี้ เป็นการจัดการแบบร่วมสมาพันธ์ (federation)

7.3 การทดสอบสถาปัตยกรรมระบบ

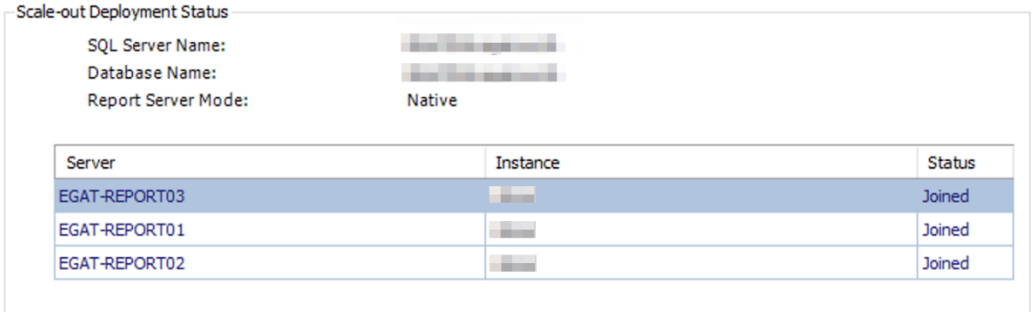
ในงานธุรกิจอัจฉริยะ (business intelligence) มีการนำเสนอข้อมูลดิบมาวิเคราะห์และนำเสนอในรูปแบบต่าง ๆ (visualization) แนวคิดคู่เสมือนดิจิทัล (digital twin) คือการจำลองการติดตามสถานะของระบบที่สนใจเป็นกระดานสรุปข้อมูล (Data dashboard) เช่นข้อมูลสรุปจากอุปกรณ์ไอโอที นำเสนอออกมาเป็นภาพรวมของระบบเป็นต้น ทำให้ติดตามสถานะภาพรวมของระบบได้สะดวก แนวคิดนี้เหมือนการให้บริการ PowerBI ที่ใช้ในปัจจุบันของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย รูปที่ 42 นำเสนอแผนภาพการทำงานของเครื่องมือวิเคราะห์ข้อมูลและนำเสนอด้วยเครื่องมือ PowerBI งานบีไอ (BI ย่อมาจาก Business intelligence) นี้มีขั้นตอนเริ่มจากการดึงข้อมูลจากที่เก็บข้อมูลต่าง ๆ ที่เป็นได้ทั้งฐานข้อมูล ไฟล์ CSV ไฟล์ excel มาประมวลผลและบันทึกผลลัพธ์กลับลงฐานข้อมูล จากนั้นผู้ใช้งานจะสามารถดูกระดานสรุปข้อมูล (dashboard) ที่ดึงข้อมูลผลลัพธ์ที่อัพเดทตามเวลาในการติดตามสถานะของระบบได้ เครื่องมือบีไออาศัยเซิร์ฟเวอร์ที่ต้องมีการดึงข้อมูลล่าสุดจากแหล่งกำเนิดข้อมูลเป็นระยะ โดยการติดตั้งและให้บริการ PowerBI ของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทยใช้ทรัพยากรสูง โดยการติดตั้ง PowerBI report server 3 เครื่องให้สามารถทำงานแทนกันได้ โดยแต่ละโนดใช้หน่วยประมวลผลกลาง 8 คอร์ และหน่วยความจำ 128 GB ซึ่งเป็นเทคโนโลยีเวอร์ชัวไลเซชันทั้ง 3 เครื่อง รูปที่ 43 แสดงสเปคของเครื่อง PowerBI report server และรูปที่ 44 แสดงการทำคลัสเตอร์ของ PowerBI report server จำนวน 3 โนดเข้าด้วยกัน



รูปที่ 42 แผนภาพการทำงานของเครื่องมือวิเคราะห์และนำเสนอข้อมูล (data visualization) ของ PowerBI

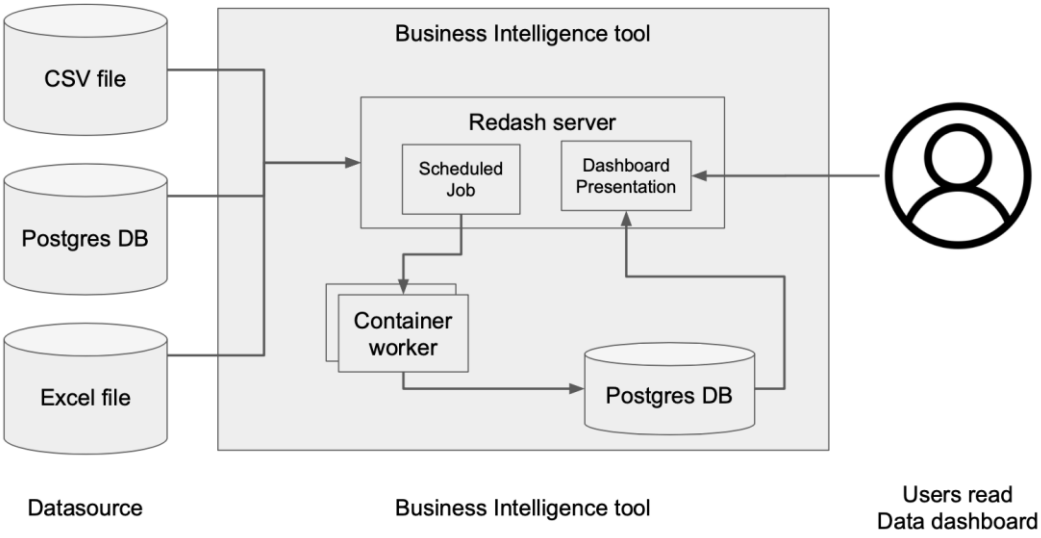


รูปที่ 43 สเปคการติดตั้งโนดประมวลผล PowerBI report server จำนวน 1 โนด



รูปที่ 44 การทำคลัสเตอร์ของ PowerBI ในการให้บริการจำนวน 3 โนด

เครื่องมือบีไออีกตัวหนึ่งซึ่งเป็นแบบโอเพนซอร์สคือ โปรแกรมรีแดช (Redash) เป็นเครื่องมือประเภทบีไอเช่นเดียวกับ PowerBI แต่ใช้เทคโนโลยีคอนเทนเนอร์ที่มีความยืดหยุ่นในการเพิ่มกำลังการประมวลผลและบำรุงรักษาได้สะดวกกว่า รูปที่ 45 แสดงโครงสร้างการทำงานของโปรแกรมรีแดชโดยใช้เทคโนโลยีคอนเทนเนอร์



รูปที่ 45 แผนภาพการทำงานของเครื่องมือวิเคราะห์และนำเสนอข้อมูลของ โปรแกรมรีแดช

เพื่อแสดงประโยชน์การร่วมสมาพันธ์คิวเบอร์เนเทสคลัสเตอร์และแสดงคุณสมบัติในการสร้างโนดประมวลผลที่มีคอนเทนเนอร์ข้ามคลัสเตอร์แบบโปรแกรมรีแดช วิทยานิพนธ์นี้จึงทดสอบคุณสมบัติดังนี้ โปรแกรมรีแดชอาศัยองค์ประกอบในการให้บริการ 2 ส่วน ได้แก่

1. รีแดชเซิร์ฟเวอร์ เป็นเว็บแอพลิเคชันสำหรับนำเสนอกระดานข้อมูล
2. รีแดชเวิร์คเกอร์ คือ คอนเทนเนอร์ที่คอยรับงานวิเคราะห์ข้อมูลจากรีแดชเซิร์ฟเวอร์

ในการทดสอบการสร้างภาระข้ามคลัสเตอร์ จะรันคอนเทนเนอร์บนคิวเบอร์เนเทสใน 2 คลัสเตอร์คือบนคลัสเตอร์ A และคลัสเตอร์ B ที่เป็นรีแดชเวิร์คเกอร์คอนเทนเนอร์ทั้งคู่ ทั้ง 2 คอนเทนเนอร์จะคอยรับงานจากรีแดชเซิร์ฟเวอร์ช่วยประมวลผลร่วมกัน และทดสอบคุณสมบัติเมื่อมีคอนเทนเนอร์ใดคอนเทนเนอร์หนึ่งตาย อีกคอนเทนเนอร์หนึ่งจะให้บริการแทนได้

ผลการทดสอบการรันรีแดชเวิร์คเกอร์เพื่อช่วยเพิ่มระดับความพร้อมใช้ของกระดานข้อมูลและช่วยในการประมวลผลการอัพเดทข้อมูล สามารถรันบนคลัสเตอร์อื่นที่ร่วมสมาพันธ์ได้ รายละเอียดการทดสอบและผลการทดสอบรวบรวมในภาคผนวก

7.4 บทสรุป

วิทยานิพนธ์นี้ได้พัฒนาสถาปัตยกรรมระบบการร่วมสมาพันธ์คลัสเตอร์ และทดสอบความสามารถในการแชร์ทรัพยากรข้ามคลัสเตอร์ ช่วยในการประมวลผลข้อมูล หรือเป็นคอนเทนเนอร์สำรองที่ทำให้ระบบหลักมีระดับความพร้อมใช้งานสูงขึ้น ระบบ OF@TEIN++ ช่วยรองรับการร่วมสมาพันธ์คลัสเตอร์โดยผู้ดูแลระบบหลายกลุ่ม แม้ผู้ใช้งานจะไม่มีสิทธิรันคอนเทนเนอร์บนคลัสเตอร์ของผู้ดูแลระบบกลุ่มอื่นโดยตรง แต่ระบบ OF@TEIN++ จะช่วยในการแชร์สิทธิการใช้งานข้ามคลัสเตอร์ที่มีการจำกัดสิทธิโดยผู้ดูแลระบบแต่ละคลัสเตอร์เรียบร้อยแล้ว ให้สามารถยืมทรัพยากรข้ามคลัสเตอร์ได้โดยสะดวก การทดสอบการรันระบบรีแดชที่สามารถนำไปใช้ในการนำเสนอข้อมูลในโครงการพลังงานอัจฉริยะ สามารถทดสอบประโยชน์การรันรีแดชเวิร์คเกอร์ในหลายคลัสเตอร์ที่สามารถช่วยกันทำงานหรือทำงานแทนกันได้ การใช้เทคโนโลยีคอนเทนเนอร์ของรีแดชสามารถบริหารจัดการเซิร์ฟเวอร์เพื่องานนำเสนอข้อมูลได้สะดวกและใช้ทรัพยากรน้อยกว่าแบบ PowerBI สถาปัตยกรรมระบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์ สามารถยกระดับความพร้อมใช้งานของระบบงานบนคอนเทนเนอร์ได้ดีขึ้น และสามารถประสานประโยชน์การทำงานระหว่างคิวเบอร์เนเทสคลัสเตอร์ได้

บทที่ 8
บริบทการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

8.1 การให้บริการคลาวด์ของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

แผนกบริหารระบบ กองบริหารระบบคอมพิวเตอร์ ฝ่ายปฏิบัติการเทคโนโลยีสารสนเทศ การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย ให้บริการเซิร์ฟเวอร์ 2 บริการหลัก คือ โคโลเคชัน และ การประมวลผลแบบคลาวด์ ปัจจุบันการพัฒนาระบบงานใหม่ๆ ให้ความสำคัญกับความสามารถในการหดหรือขยายขนาดของระบบ (scalability) และคุณสมบัติการเคลื่อนย้ายได้ (portability) โดยฝั่งผู้ให้บริการได้คำนึงถึงต้นทุนอุปกรณ์ คุณภาพการให้บริการ และงานบำรุงรักษาให้สามารถให้บริการได้อย่างต่อเนื่อง ปัจจุบันมีการให้บริการเฉพาะแบบวีเอ็ม กว่า 1,300 เครื่อง การพัฒนาบริการไปสู่บริการคอนเทนเนอร์แอสอะเซอร์วิส (container as a service) จะต้องมีการติดตั้งคอนเทนเนอร์ออร์เคสเตรเตอร์ และการคอนเทนเนอร์ไรซ์ระบบงานต่าง ๆ แนวทางการใช้คอนเทนเนอร์ในการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย มีความเป็นไปได้สูง โดยมีอุปสรรคในการเปลี่ยนแปลง คือ ระบบปฏิบัติการวีเอ็มส่วนใหญ่เป็นระบบปฏิบัติการวินโดว์ โดยเทคโนโลยีคอนเทนเนอร์มีการแยกการใช้งานคอนเทนเนอร์บนระบบปฏิบัติการวินโดว์และลินุกซ์ออกจากกัน ทำให้อาจต้องพิจารณาถึงการปรับปรุงจากระบบที่ทำงานบนวินโดว์มาสู่ลินุกซ์ในกรณีการใช้งานต่าง ๆ หรือหากใช้คอนเทนเนอร์บนวินโดว์ จะเป็นเทคโนโลยีที่อยู่ระหว่างการพัฒนา บางหน่วยงานมีการจ้างพัฒนาระบบที่ใช้งานบนวินโดว์รุ่นเก่าและหมดสัญญาการบำรุงรักษาแล้ว จึงจำเป็นต้องคงการทำงานแบบวีเอ็มเดิมไว้จนหมดอายุการใช้งาน สำหรับแอพลิเคชันใหม่ที่ต้องการคุณสมบัติการพร้อมใช้งานสูง และขยายขนาดได้ตามจำนวนผู้ใช้งาน เทคโนโลยีคอนเทนเนอร์มีความเหมาะสมอย่างยิ่ง และช่วยให้การบำรุงรักษามีประสิทธิภาพยิ่งขึ้น

ในการให้บริการวีเอ็มของแผนกบริหารระบบ ได้ใช้เครื่องมือ วีเอ็มแวร์ (VMware) ในการจัดการวีเอ็ม และเปิดคุณสมบัติ *การทนต่อความผิดพร่อง* (fault tolerance) และ *ความพร้อมใช้ระดับสูง* (high availability) ในการรองรับระบบที่มีความอ่อนไหว ข้อแตกต่างของคุณสมบัติทั้ง 2 คือ *ความพร้อมใช้ระดับสูง* ใช้การสร้างวีเอ็ม 2 เครื่อง โดยมีเครื่องหนึ่งทำงานจริง อีกเครื่องหนึ่งอยู่ในสภาพพร้อมใช้แบบรอทำงาน เมื่อเครื่องใดเครื่องหนึ่งหยุดทำงาน อีกเครื่องจะพร้อมให้บริการโดยเร็ว โดยมีช่วงเวลาหยุดให้บริการน้อยที่สุด ส่วนคุณสมบัติ *การทนต่อความผิดพร่อง* มีลักษณะคือมีการทำงานควบคู่กันทั้ง 2 เครื่อง เมื่อเกิดความผิดพลาดของเครื่องหนึ่ง อีกเครื่องจะรองรับการทำงานแทนโดยไม่มีการสูญเสียการให้บริการ (zero downtime)

รูปแบบ*การทนต่อความผิดพร่อง*จะใช้ทรัพยากร 2 เท่าในการให้บริการ ต่างจากแบบ*ความพร้อมใช้ระดับสูง*ที่ใช้ทรัพยากรเครื่องน้อยกว่า แต่แลกกับมีช่วงหยุดให้บริการในการย้ายระบบงานไปรันในโนดอื่น

8.2 การเปรียบเทียบข้อดีข้อเสียแต่ละแบบ

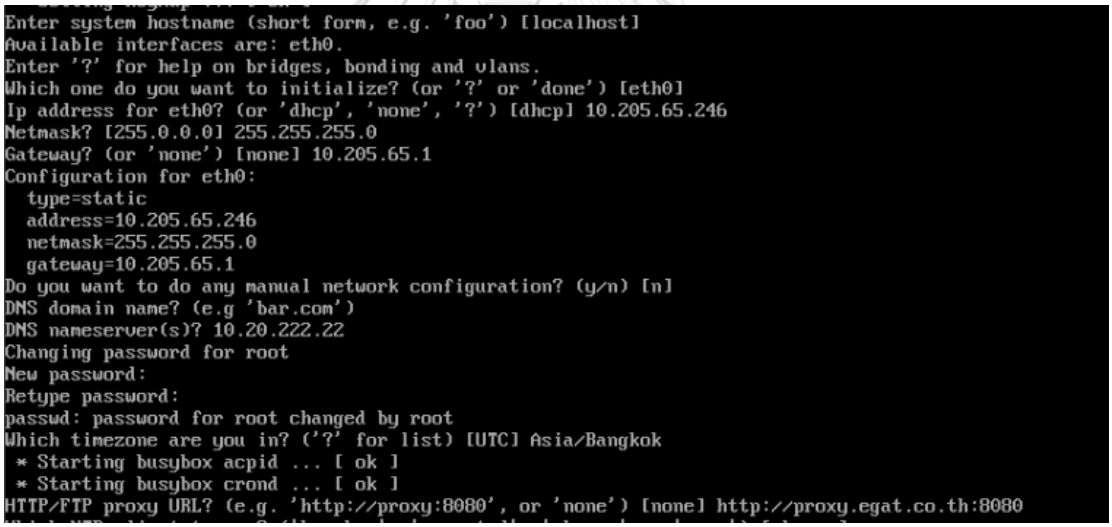
แผนกบริหารระบบปัจจุบันให้บริการและบำรุงรักษาระบบอยู่ 2 รูปแบบคือ แบบเซิร์ฟเวอร์มาตรฐาน และแบบเวอร์ชัวแมชชีน จึงเปรียบเทียบข้อดี ข้อเสียแต่ละเทคโนโลยีดังตารางที่ 26 ดังนี้

ตารางที่ 26 การเปรียบเทียบข้อดีข้อเสียแต่ละเทคโนโลยี

	ข้อดี	ข้อเสีย
เซิร์ฟเวอร์มาตรฐาน	● พัฒนาง่าย มีรูปแบบตรงไปตรงมา ● แอพลิเคชันยึดครองทรัพยากรทั้งหมด ● รองรับแอพลิเคชันเกือบทุกรูปแบบ	● ย้ายระบบงานยาก ● ไม่เอื้อต่อการขยายระบบ ● มีระบบปฏิบัติการเดียว ● มีต้นทุนสูง
เวอร์ชัวแมชชีน	● จำลองระบบปฏิบัติการได้หลายรูปแบบ ● รองรับแอพลิเคชันเกือบทุกรูปแบบ ● สำรองข้อมูลสถานะอย่างเร็วได้ (snapshot) ● เปิดคุณสมบัติความพร้อมใช้งานระดับสูงได้	● ใช้ทรัพยากรมากเนื่องจากจำลองเซิร์ฟเวอร์มาตรฐานทั้งเครื่อง
คอนเทนเนอร์	● ใช้ระบบปฏิบัติการร่วมกันกับโฮส ● เคลื่อนย้าย ติดตั้ง แอพลิเคชันได้ง่าย	● ระบบงานบางอย่างไม่สามารถคอนเทนเนอร์ไรซ์ได้ ● มีระบบปฏิบัติการเดียว
คิวเบอร์เนเทส	● ได้คุณสมบัติข้อดีของคอนเทนเนอร์ทั้งหมด ● เปิดคุณสมบัติความพร้อมใช้งานระดับสูงได้ ● ขยายขนาดระบบได้ง่าย ● บริหารโนดคอนเทนเนอร์แบบรวมศูนย์ ● บำรุงรักษาโครงสร้างพื้นฐานได้สะดวก ● มีได้หลายระบบปฏิบัติการ หลายสถาปัตยกรรม	● ระบบงานบางอย่างไม่สามารถคอนเทนเนอร์ไรซ์ได้ ● มาสเตอร์โนดมีความสำคัญสูงต้องได้รับการดูแล
คิวเบอร์เนเทสแบบร่วมสมาพันธ์	● ได้คุณสมบัติข้อดีของคิวเบอร์เนเทสทั้งหมด ● ขอใช้ทรัพยากรชั่วคราวได้ ● ยกระดับความพร้อมใช้งานยิ่งขึ้น	● ระบบที่ทำงานข้ามคลัสเตอร์ มีความซับซ้อนในการติดตั้ง

8.3 การทดสอบการใช้งานคอนเทนเนอร์ในองค์กร

ระบบเครือข่ายภายในของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย มีความปลอดภัยสูง ทุกการติดต่อเครือข่ายต้องกระทำผ่านระบบพร็อกซี่ การใช้งานคอนเทนเนอร์ในเครื่องคอมพิวเตอร์บนเครือข่ายภายในอาจมีความสับสนในการตั้งค่าพร็อกซี่บนเครื่องคอมพิวเตอร์และบนคอนเทนเนอร์เอนจิน โดยมีรายละเอียดการติดตั้งดอกเกอร์และการตั้งค่าพร็อกซี่ในภาคผนวก การทดสอบการใช้งานคอนเทนเนอร์ในองค์กรประสบความสำเร็จ สามารถแสดงการใช้งานคอนเทนเนอร์ได้ และภายในหน่วยงานสามารถให้บริการติดตั้งวีเอ็มที่รองรับการทำงานคอนเทนเนอร์ได้ ได้จัดทำเทมเพลตการสร้างวีเอ็มที่ติดตั้งคอนเทนเนอร์เอนจินพร้อมใช้งานบนระบบคลาวน์ขององค์กร และจัดทำขั้นตอนการตั้งค่าพร็อกซี่บนคอนเทนเนอร์เอนจินสำหรับกรณีอื่น ๆ ไว้ รูปที่ 46 แสดงการตั้งค่าพร็อกซี่บนระบบปฏิบัติการอัลไพน์ลินุกซ์ที่จำเป็นต่อการติดตั้งดอกเกอร์เอนจิน รูปที่ 47 แสดงการตั้งค่าพร็อกซี่บนดอกเกอร์เอนจินที่จำเป็นในการรันคอนเทนเนอร์ และรูปที่ 48 แสดงผลลัพธ์การตั้งค่าพร็อกซี่ซึ่งแสดงสถานะตัวอย่างการรันคอนเทนเนอร์ได้อย่างสมบูรณ์ในองค์กร



รูปที่ 46 การตั้งค่าพร็อกซี่บนระบบปฏิบัติการอัลไพน์ลินุกซ์

```
# /etc/conf.d/docker: config file for /etc/init.d/docker

# where the docker daemon output gets piped
# this contains both stdout and stderr. If  you need to separate them,
# see the settings below
#DOCKER_LOGFILE="/var/log/docker.log"

# where the docker daemon stdout gets piped
# if this is not set, DOCKER_LOGFILE is used
#DOCKER_OUTFILE="/var/log/docker-out.log"

# where the docker daemon stderr gets piped
# if this is not set, DOCKER_LOGFILE is used
#DOCKER_ERRFILE="/var/log/docker-err.log"

# Settings for process limits (ulimit)
#DOCKER_ULIMIT="-c unlimited -n 1048576 -u unlimited"

# seconds to wait for sending SIGTERM and SIGKILL signals when stopping docker
#DOCKER_RETRY="TERM/60/KILL/10"

# where the docker daemon itself is run from
#DOCKERD_BINARY="/usr/bin/dockerd"

# any other random options you want to pass to docker
DOCKER_OPTS=""

export HTTP_PROXY="http://proxy.egat.co.th:8080"
export HTTPS_PROXY="http://proxy.egat.co.th:8080"
~
~
~
```

รูปที่ 47 การตั้งค่าพร็อกซี่บนดอกเกอร์เอนจิน

```
devops1:~# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS
        PORTS                                      NAMES
f1ec13678908   filebrowser/filebrowser:v2   "/filebrowser"           5 weeks ago   Up 5 weeks (unhea
lthy)   0.0.0.0:443->80/tcp, :::443->80/tcp     filebrowser
05ea564b70f8   strapi/strapi:3.6.8-alpine   "docker-entrypoint.s…"   5 weeks ago   Up 4 weeks
        0.0.0.0:80->1337/tcp, :::80->1337/tcp   strapi
```

รูปที่ 48 สถานะการรันคอนเทนเนอร์ในตัวอย่างวีเอ็มที่ทดสอบในองค์กรบนเครือข่ายภายใน

บทที่ 9
บทสรุป

วิทยานิพนธ์นี้ได้นำเสนอสถาปัตยกรรมระบบพลังงานอัจริยะแบบคลาวด์ของคลัสเตอร์ที่ร่วมสมาพันธ์โดยผู้ดูแลระบบหลายกลุ่ม โดยมีกรณีศึกษาต่าง ๆ ที่เป็นพื้นฐานเพื่อการศึกษาและพัฒนาต่อยอดการประยุกต์ในบริบทของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทยได้ องค์ประกอบต่าง ๆ ที่นำเสนอสามารถประกอบเป็นสถาปัตยกรรมระบบตามความต้องการของระบบ และความพร้อมใช้ในระดับต่าง ๆ ทั้งความพร้อมใช้ในระดับคอนเทนเนอร์ ความพร้อมใช้ในระดับโนด และความพร้อมใช้ในระดับคลัสเตอร์

สถาปัตยกรรมระบบที่ 1 ในการเร่งความเร็วการประมวลผลข้อมูลแบบขนานเชิงกระจาย เป็นการนำเสนอแนวทางการรีดสมรรถนะ (performance) จากการควบรวมทรัพยากร ในการวิเคราะห์ข้อมูลขนาดใหญ่ เช่นการวิเคราะห์ไฟล์ประเภท CSV โดยใช้คุณสมบัติของคอนเทนเนอร์และความสามารถในการยืดหดทรัพยากร ที่สามารถเร่งความเร็วการประมวลผลมากขึ้นได้ตามจำนวนทรัพยากรที่มี สถาปัตยกรรมระบบนี้แสดงถึงสมรรถนะของเทคโนโลยีคอนเทนเนอร์ในการบริหารการใช้ทรัพยากรอย่างมีประสิทธิภาพ

สถาปัตยกรรมระบบที่ 2 เพื่อการจัดการคอนเทนเนอร์แบบรวมศูนย์และมีความยืดหยุ่นในการจัดการโนดเซิร์ฟเวอร์ระยะไกล สถาปัตยกรรมระบบนี้แสดงคุณสมบัติด้านการตั้งค่า (configuration) ที่เอื้อต่องานบำรุงรักษา และสามารถรวมศูนย์การจัดการคอนเทนเนอร์จากทุกโนดในคลัสเตอร์ได้

สถาปัตยกรรมระบบที่ 3 ออกแบบความพร้อมใช้ระดับสูง เป็นสถาปัตยกรรมระบบที่มีการทนต่อความผิดพร่อง (fault tolerance) รองรับกรณีการนำไปใช้งานจริงที่สามารถวิเคราะห์และประเมินระดับความพร้อมใช้ตามโครงสร้างคิวเบอร์เนเทสคลัสเตอร์ที่ออกแบบ ในการจัดการความล้มเหลวได้

สถาปัตยกรรมระบบที่ 4 รองรับความหลากหลายของผู้ใช้งาน ในการจัดการด้านความปลอดภัย (security) ในมิติด้านความสามารถในการเข้าถึงทรัพยากรระหว่างโครงการ และการจัดลำดับชั้นของทรัพยากรของการรันคอนเทนเนอร์ในโนดที่มีความเข้ากันได้ มีการนำเสนอการจัดการผู้ใช้งาน การจัดการกลุ่มภาระ และการจัดการกลุ่มของทรัพยากรที่มีความหลากหลายให้มีความเข้ากันได้ ให้เกิดความปลอดภัย

สถาปัตยกรรมสุดท้ายเป็นการออกแบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์ เป็นสถาปัตยกรรมระบบที่มีกลไกการร่วมสมาพันธ์ระหว่างคลัสเตอร์ในการแชร์ทรัพยากรร่วมกัน เป็นรูปแบบที่เกิดขึ้นระหว่างคลัสเตอร์ ทำให้บริการที่รันโดยคอนเทนเนอร์มีระดับความพร้อมใช้สูงขึ้น สถาปัตยกรรม

ระบบนี้มีความยืดหยุ่นในการยืมทรัพยากรข้ามหน่วยงานกัน ซึ่งอาจเป็นคลาวด์สาธารณะ หรือผู้ใช้งานต่างกลุ่มก็ได้ การใช้งานระหว่างคลัสเตอร์ข้ามหน่วยงานมีประเด็นในการเข้าถึงทรัพยากรที่ต้องทำงานผ่านระบบศูนย์กลาง สถาปัตยกรรมระบบที่นำเสนอจะสามารถจำกัดความสามารถในการเข้าถึงข้ามหน่วยงานได้อย่างปลอดภัย (security)

สถาปัตยกรรมระบบต่าง ๆ ที่นำเสนอ ได้แสดงการประยุกต์ในแต่ละกรณีสำหรับโครงการพลังงานอัจริยะต่าง ๆ ทั้งงานด้านแอพลิเคชัน และโครงสร้างพื้นฐาน ได้แก่ โครงการ CU-BEMS โครงการ IoTcloudServe โครงการ OF@TIEN++ และการประยุกต์ในบริบทการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

วิทยานิพนธ์นี้ได้นำเสนอแนวทางรองรับการใช้งานจริงในองค์กร ในการออกแบบโครงสร้างสถาปัตยกรรมระบบที่พิสูจน์การทำงานจริงบนพื้นฐานเทคโนโลยีคอนเทนเนอร์ออร์เคสเตรทชัน การเปลี่ยนผ่านจากเทคโนโลยีเวอร์ชัวไลเซชันไปสู่คอนเทนเนอร์ มีแนวคิด เครื่องมือ และวิธีปฏิบัติหลายอย่างที่ต้องค่อยปรับใช้ สถาปัตยกรรมระบบที่นำเสนอจะสามารถพัฒนาได้ทั้งในเชิงการศึกษา ตั้งแต่การใช้งานบนวีเอ็ม การติดตั้งบนคอมพิวเตอร์ขนาดเล็กบนราสเบอร์พาย การติดตั้งบนเซิร์ฟเวอร์จริง การทำงานร่วมกันระดับคลัสเตอร์ การให้บริการที่ต้องการความพร้อมใช้งานระดับสูง และการทำงานร่วมกันระหว่างหลายคลัสเตอร์ได้

สถาปัตยกรรมระบบที่ออกแบบคำนึงถึงการมีส่วนร่วมของผู้ดูแลระบบหลากหลาย ทั้งงานบำรุงรักษาของผู้ดูแลระบบ การติดตามสถานะของภาระงานโดยผู้พัฒนาแอพลิเคชัน การรักษาความต่อเนื่องทางธุรกิจ และกลไกการจัดสรรทรัพยากรข้ามกลุ่มของผู้ดูแลระบบหลายกลุ่ม ผู้ศึกษาสามารถนำไปปรับใช้ในการออกแบบสถาปัตยกรรมระบบตามความต้องการและติดตั้งใช้งานจริงได้

การต่อยอดงานในอนาคต มีกรณีศึกษาการใช้งานพื้นที่จัดเก็บข้อมูลที่ต้องการความพร้อมใช้ระดับสูงในการย้ายข้อมูลของภาระที่มีการอพยพข้ามคลัสเตอร์ การอพยพของภาระที่ใช้พื้นที่จัดเก็บข้อมูลต้องมีการวางแผนล่วงหน้า การย้ายข้อมูลระหว่างคลัสเตอร์เป็นกรณีที่ไม่สามารถดำเนินการได้อย่างทันที จึงอาจพัฒนาแนวทางการยืมทรัพยากรพื้นที่จัดเก็บข้อมูลระหว่างคลัสเตอร์ต่อไป นอกจากนี้ยังมีกรณีศึกษาที่น่าสนใจ เช่น การใช้เทคโนโลยีเวอร์ชัวไลเซชันร่วมกับคอนเทนเนอร์ในคลัสเตอร์เดียวกัน ที่มีความสามารถในการอพยพทั้งวีเอ็มและคอนเทนเนอร์ได้ เป็นต้น

ภาคผนวก

ภาคผนวก ก การออกแบบโครงสร้างและการติดตั้งคิวเบอร์เนเทส

องค์ประกอบสำคัญในการติดตั้งคิวเบอร์เนเทส

1. รุ่นของคิวเบอร์เนเทสเอนจิน (Kubernetes engine)

มาสเตอร์โนดและเวิร์คเกอร์โนดควรใช้คิวเบอร์เนเทสเอนจินรุ่นเดียวกัน ผู้พัฒนาระบบอาจใช้ตัวช่วยเคสามเอสในการติดตั้งคิวเบอร์เนเทสเอนจิน โดยรุ่นของคิวเบอร์เนเทสเอนจินและรุ่นของเคสามเอส (K3S) เป็นคนละอย่างกัน ควรตรวจสอบเคสามเอสที่เลือกใช้ก่อนการติดตั้ง [44] และตรวจสอบความเข้ากันได้หากใช้เคสามเอสหลายรุ่นในคลัสเตอร์เดียวกัน

2. คอนเทนเนอร์เอนจิน (container engine)

ปกติเคสามเอสจะติดตั้งคอนเทนเนอร์ดี (containerd) เป็นคอนเทนเนอร์เอนจินอัตโนมัติ [45] แต่คอนเทนเนอร์เอนจินแบบดอกเกอร์ได้รับความนิยมมากกว่า ในระยะก่อนติดตั้งสามารถกำหนดให้เคสามเอสควบคุมคอนเทนเนอร์เอนจินผ่านดอกเกอร์แทนได้ แต่บนเครื่องต้องติดตั้งดอกเกอร์เอนจินก่อนติดตั้งเคสามเอส และตั้งค่าเคสามเอสให้ใช้ดอกเกอร์เอนจิน

ในเครื่องคิวเบอร์เนเทสผู้ดูแลระบบสามารถควบคุมคอนเทนเนอร์ผ่านคำสั่งเฉพาะของคอนเทนเนอร์เอนจินได้โดยตรง หากผู้ดูแลระบบลบคอนเทนเนอร์บางส่วนที่สร้างผ่านคิวเบอร์เนเทส คิวเบอร์เนเทสเอนจินจะสั่งคอนเทนเนอร์เอนจินให้สร้างคอนเทนเนอร์ใหม่ทดแทน เพื่อรักษาเสถียรภาพของระบบ (desired state)

อาจมีคอนเทนเนอร์บางส่วน ที่ผู้ดูแลระบบรันคอนเทนเนอร์โดยไม่ผ่านคิวเบอร์เนเทสเอนจิน คอนเทนเนอร์เหล่านี้จะยังคงใช้งานได้ปกติ ดังนั้นในโนดที่ติดตั้งดอกเกอร์เอนจิน ผู้ดูแลระบบสามารถจัดการคอนเทนเนอร์ได้ผ่านทั้งคำสั่งคิวเบอร์เนเทสและคำสั่งดอกเกอร์ ส่วนโนดที่ใช้คอนเทนเนอร์ดีสามารถใช้คำสั่งคอนเทนเนอร์ดีแทนได้เช่นกัน [46]

ในคิวเบอร์เนเทสคลัสเตอร์เดียวกัน คอนเทนเนอร์เอนจินของแต่ละโนดไม่จำเป็นต้องใช้คอนเทนเนอร์เอนจินแบบเดียวกัน ในคลัสเตอร์เดียวกันอาจมีได้ทั้งคอนเทนเนอร์ดีและดอกเกอร์

3. โทเคน (token)

โทเคนมีความสำคัญที่ต้องใช้ในการเข้าร่วมคลัสเตอร์ของโนดใหม่ ทั้งกรณีเป็นเวิร์คเกอร์โนดและมาสเตอร์โนด ควรกำหนดโทเคนตั้งแต่โนดแรก หากไม่กำหนด การติดตั้งเคสามเอสจะสร้างโทเคนอัตโนมัติ และจัดเก็บในรูปของแฮชที่ถอดรหัสได้ยาก อาจทำให้ไม่สามารถขยายขนาดคลัสเตอร์ได้

4. ฐานข้อมูลจัดเก็บสถานะ

คิวเบอร์นิทิสคลัสเตอร์จะมีฐานข้อมูลเก็บสถานะของคลัสเตอร์ ถือเป็นส่วนที่สำคัญที่สุด หากฐานข้อมูลเสียหายจะทำให้คลัสเตอร์หยุดทำงานและอาจไม่สามารถกู้คืนได้ โดยปกติฐานข้อมูลจะฝังอยู่ในมาสเตอร์โนดโดยใช้อีทีซีดี (etcd) [33] หากในคลัสเตอร์เดียวกันมีมาสเตอร์โนดหลายโนด อีทีซีดีจะกระจายไปยังทุกมาสเตอร์โนด การทำงานของฐานข้อมูลจัดเก็บสถานะจะผลโดยตรงกับความพร้อมใช้งานของคลัสเตอร์ (availability) กล่าวคือการเปิดคุณสมบัติความพร้อมใช้งานระดับสูง (high availability) ที่ใช้ฐานข้อมูลจัดเก็บสถานะแบบอีทีซีดี จำเป็นต้องมีมาสเตอร์โนดเป็นเลขคี่ตั้งแต่ 3 ขึ้นไป [31] แต่อีทีซีดีอาจทำงานได้ช้าลงหากติดตั้งบางมาสเตอร์โนดบนราสเบอรีพาย [31]

การติดตั้งคิวเบอร์เนเทสอีกแนวทางหนึ่งคือใช้ฐานข้อมูลจัดเก็บสถานะภายนอก มีข้อดีคือการสำรองข้อมูลสถานะของคลัสเตอร์สามารถกระทำฐานข้อมูลโดยตรง การกู้คืนฐานข้อมูลจะเหมือนการย้อนเวลากลับไปในช่วงสถานะของคลัสเตอร์ยังปกติ การซ่อมแซมคิวเบอร์เนเทสคลัสเตอร์ลักษณะนี้อาจไม่กระทบกับมาสเตอร์โนดหรือเวิร์คเกอร์โนดใด ๆ วิธีนี้สามารถเปิดคุณสมบัติความพร้อมใช้งานระดับสูงได้ตั้งแต่มีจำนวนมาสเตอร์โนด 2 โนดขึ้นไป [32] ฐานข้อมูลแบบภายนอกที่รองรับได้แก่โปสเกรสคิวแอล (POSTGRESQL) มายเอสคิวแอลหรือมาเรียดีบี (MYSQL/MARIADB) และอีทีซีดีแบบภายนอก

5. พอร์ตสื่อสารระหว่างมาสเตอร์โนดและเวิร์คเกอร์โนด

ระหว่างเครื่องมาสเตอร์โนดและเวิร์คเกอร์โนดจะสื่อสารผ่านพอร์ตหมายเลข 6443 ของเครื่องมาสเตอร์โนด หมายเลขนี้สามารถเปลี่ยนแปลงได้ เพราะอาจถูกจำกัดจากนโยบายของผู้ให้บริการคลาวด์ กรณีใช้บริการดีเอนเอสพรอกซีที่ปิดบังไอพีแอดเดรสจริง อาจมีพอร์ตหมายเลขจำกัดที่ใช้งานได้ เช่นบริการคลาวด์แฟร์ การใช้พรอกซีจะไม่รองรับพอร์ตหมายเลข 6443 [47] หากออกแบบให้มีโนดทำงานในหลายผู้ให้บริการ เวิร์คเกอร์โนดอาจไม่สามารถติดต่อเครื่องมาสเตอร์โนดได้ จำเป็นต้องกำหนดหมายเลขพอร์ตอื่นแทน

การติดตั้งคิวเบอร์เนเทส

คิวเบอร์เนเทสมีวิธีติดตั้งหลายวิธีดังนี้

1. เคสามเอส (K3S) เป็นวิธีติดตั้งคิวเบอร์เนเทสบนระบบปฏิบัติการลินุกซ์อย่างง่าย และรองรับหลายสถาปัตยกรรม [44]
2. เคสามโอเอส (K3OS) เป็นวิธีติดตั้งคิวเบอร์เนเทสพร้อมระบบปฏิบัติการอัลไพน์ลินุกซ์ [48]
3. เคสามดี (K3D) เป็นวิธีติดตั้งเคสามเอสแบบคอนเทนเนอร์

การติดตั้งเคสามเอส

โดยปกติเคสามเอสเป็นวิธีพื้นฐานที่เลือกใช้ในการติดตั้งคิวเบอร์เนเทส โดยการรันคำสั่งดังนี้

```
curl -sfL https://get.k3s.io | sh -
```

การติดตั้งเคสามเอสจะใช้ค่าของตัวแปรกลาง (environment variable) หรือ จากตัวแปรที่กำหนดในระยะการรันคำสั่ง มีค่าตัวแปรที่สำคัญดังนี้

ตัวแปร	ความหมาย
K3S_URL	ที่อยู่ของมาสเตอร์โนด
INSTALL_K3S_VERSION	รุ่นของเคสามเอส
INSTALL_K3S_SKIP_START	การละให้เคสามเอสทำงานหลังการติดตั้ง
K3S_DATASTORE_ENDPOINT	ที่อยู่ของฐานข้อมูลจัดเก็บสถานะกรณีติดตั้งแบบภายนอก
K3S_TOKEN	โทเคน

นอกจากนี้ยังสามารถตั้งค่าเคสามเอสระหว่างการทำงาน ให้เลือกคอนเทนเนอร์เอนจินและพอร์ทที่ใช้งานระหว่างมาสเตอร์โนดและเวิร์คเกอร์โนด การตั้งค่าเคสามเอสระหว่างการทำงาน [49] มีดังนี้

ตัวแปร	ความหมาย
--docker	ให้ใช้ดอกเกอร์เอนจิน
--https-listen-port=xxxx	ใช้พอร์ทอื่นแทน 6443
server, -s	กำหนดให้ทำงานเป็นมาสเตอร์

การกำหนดบทบาทของคิวเบอร์เนเทสให้เป็นมาสเตอร์โนดหรือเวิร์คเกอร์โนด จะขึ้นอยู่กับการมีการกำหนดค่าของตัวแปร K3S_URL หรือไม่ หากกำหนดจะถือว่าให้เคสามเอสทำงานเป็นเวิร์คเกอร์โนด ตัวแปร K3S_DATASTORE_ENDPOINT จะมีผลเฉพาะกรณีที่กำหนดบทบาทของเคสามเอสเป็นมาสเตอร์ ในกรณีที่ใช้ฐานข้อมูลจัดเก็บสถานะแบบฝังและต้องการเพิ่มมาสเตอร์โนดให้กำหนดค่าของ K3S_URL และกำหนดตัวแปรกลุ่มเคสามเอสระหว่างการทำงานเป็นมาสเตอร์

ตัวอย่างที่ 1 คำสั่งติดตั้งมาสเตอร์โนด

curl -sfL https://get.k3s.io | sh -s - --docker	
บทบาท	มาสเตอร์
โทเคน	สร้างอัตโนมัติ
ที่อยู่เคสามเอสมาสเตอร์	-
รุ่นของเคสามเอส	ล่าสุด
ฐานข้อมูลจัดเก็บสถานะ	อีทีซีดี
คอนเทนเนอร์เอนจิน	ดอกเกอร์เอนจิน (ต้องติดตั้งมาก่อน)

ตัวอย่างที่ 2 คำสั่งติดตั้งมาสเตอร์โนดที่ใช้ฐานข้อมูลจัดเก็บสถานะภายนอก

curl -sfL https://get.k3s.io | \ INSTALL_K3S_VERSION=v1.21.1+k3s1 \ K3S_TOKEN=token \ K3S_DATASTORE_ENDPOINT=postgres://user:password@ip:port/kubernetes?sslmode=disable \ sh -s - --docker	
บทบาท	มาสเตอร์
โทเคน	token
ที่อยู่เคสามเอสมาสเตอร์	-
รุ่นของเคสามเอส	v1.21.1+k3s1
ฐานข้อมูลจัดเก็บสถานะ	โปสเกรสภายนอก
คอนเทนเนอร์เอนจิน	ดอกเกอร์เอนจิน (ต้องติดตั้งมาก่อน)

ตัวอย่างที่ 3 คำสั่งติดตั้งเวิร์คเกอร์โนด

curl -sfL https://get.k3s.io | K3S_URL=https://202.28.193.102:6443 K3S_TOKEN=token sh -	
บทบาท	เวิร์คเกอร์
โทเคน	token
ที่อยู่เคสามเอสมาสเตอร์	https://202.28.193.102:6443
รุ่นของเคสามเอส	ล่าสุด
ฐานข้อมูลจัดเก็บสถานะ	ตามมาสเตอร์โนด
คอนเทนเนอร์เอนจิน	คอนเทนเนอร์ดี

ตัวอย่างที่ 4 คำสั่งติดตั้งมาสเตอร์โนดที่ใช้พอร์ทอื่น

curl -sfL https://get.k3s.io | K3S_TOKEN=token \ INSTALL_K3S_VERSION=v1.21.1+k3s1 sh -s - --docker --https-listen-port=2083	
บทบาท	มาสเตอร์
โทเคน	token
รุ่นของเคสามเอส	v1.21.1+k3s1
พอร์ทที่ใช้	2083
ฐานข้อมูลจัดเก็บสถานะ	อีทีซีดี
คอนเทนเนอร์เอนจิน	ดอกเกอร์เอนจิน (ต้องติดตั้งมาก่อน)

ตัวอย่างที่ 5 คำสั่งติดตั้งมาสเตอร์โนดที่ 2 ขึ้นไป

curl -sfL https://get.k3s.io | K3S_TOKEN=token K3S_URL=https://202.28.193.102:6443 \ sh -s - --docker	
บทบาท	มาสเตอร์
โทเคน	token
ที่อยู่เคสามเอสมาสเตอร์	https://202.28.193.102:6443
รุ่นของเคสามเอส	ล่าสุด
ฐานข้อมูลจัดเก็บสถานะ	อีทีซีดี
คอนเทนเนอร์เอนจิน	ดอกเกอร์เอนจิน (ต้องติดตั้งมาก่อน)

ตัวอย่างที่ 6 คำสั่งติดตั้งเวิร์คเกอร์โนดที่ใช้ดอกเกอร์เอนจิน

curl -sfL https://get.k3s.io | K3S_URL=https://202.28.193.102:6443 \ K3S_TOKEN=token INSTALL_K3S_VERSION=v1.21.1+k3s1 sh - --docker	
บทบาท	เวิร์คเกอร์
โทเคน	token
ที่อยู่เคสามเอสมาสเตอร์	https://202.28.193.102:6443
รุ่นของเคสามเอส	v1.21.1+k3s1
คอนเทนเนอร์เอนจิน	ดอกเกอร์เอนจิน (ต้องติดตั้งมาก่อน)

กรณีติดตั้งบน Alpine linux ให้รันคำสั่งต่อไปนี้หลังการติดตั้งเพื่อให้เปิดโปรแกรม k3s อัตโนมัติดังนี้

- กรณีติดตั้งเป็นมาสเตอร์ หลังการติดตั้งให้รันคำสั่ง rc-update add k3s boot
- กรณีติดตั้งเป็นเวิร์คเกอร์ หลังการติดตั้งให้รันคำสั่ง rc-update add k3s-agent boot

ส่วนกรณีถอนการติดตั้ง ให้ล้างการเริ่มโปรแกรม k3s อัตโนมัติดังนี้

- กรณีติดตั้งเป็นมาสเตอร์
  - /usr/local/bin/k3s-uninstall.sh
  - หลังถอนการติดตั้งให้รัน rc-update del k3s boot
- กรณีติดตั้งเป็นเวิร์คเกอร์
  - /usr/local/bin/k3s-agent-uninstall.sh
  - หลังถอนการติดตั้งให้รัน rc-update del k3s-agent boot

การติดตั้งเคสามโอเอส

เคสามโอเอสเป็นระบบปฏิบัติการที่มีเคสามเอสฝังมาพร้อมใช้งาน เคสามโอเอสพัฒนาจากระบบปฏิบัติการอัลไพน์ลินุกซ์ [48] หากติดตั้งแบบไม่กำหนดการตั้งค่าพิเศษ จะกำหนดบทบาทของโนดเป็นมาสเตอร์อัตโนมัติ การตั้งค่าแบ่งออกเป็น 3 รูปแบบดังนี้

1. การตั้งค่าเป็นมาสเตอร์โนด
2. การตั้งค่าเป็นเวิร์คเกอร์โนด
3. การตั้งค่าเป็นมาสเตอร์โนดในคลัสเตอร์ที่มีมาสเตอร์โนดมากกว่า 1

การตั้งค่าในแบบที่ 1 และ 2 จะกำหนดผ่านตัวติดตั้งอัตโนมัติเมื่อเปิดเครื่องครั้งแรก โดยหากต้องการให้มาสเตอร์โนดทำงานเพียงโนดเดียว สามารถละการกำหนดโทเคนได้ แต่หากต้องการมีเครื่องเวิร์คเกอร์โนด หรือมีมาสเตอร์โนดมากกว่า 1 ต้องมีการกำหนดโทเคน การตั้งค่าในแบบที่ 3 ไม่สามารถทำได้ด้วยวิธีการติดตั้งแบบอัตโนมัติ จำเป็นจะต้องออกแบบโครงสร้างเฉพาะก่อนการติดตั้งเพื่อเตรียมไฟล์ตั้งค่าเคสามโอเอส การติดตั้งเคสามโอเอสมีขั้นตอนดังนี้

1. การกำหนดไฟล์โครงสร้าง ระหว่างออกแบบคลัสเตอร์
2. การติดตั้งเคสามโอเอส
3. การแก้ไขไฟล์โครงสร้างหลังการติดตั้งเคสามโอเอส

การตั้งค่าเคสามโอเอสสามารถกำหนดไฟล์โครงสร้างได้ 3 แหล่ง [48] ตามลำดับดังนี้

1. /k3os/system/config.yaml
2. /var/lib/rancher/k3os/config.yaml
3. /var/lib/rancher/k3os/config.d/*

ในระหว่างการเปิดเครื่องเคสามโอเอส ระบบจะเช็คไฟล์โครงสร้างนี้ตามลำดับ ไฟล์โครงสร้างของลำดับที่ 1 จะไม่สามารถแก้ไขได้ จะถูกกำหนดในระยะการติดตั้งเคสามโอเอสเท่านั้น หากมีไฟล์โครงสร้างในลำดับอื่น ๆ จะถือว่าให้ใช้ไฟล์โครงสร้างของลำดับที่มาทีหลังนั้นแทนที่ทั้งหมด ไม่มีการรวมไฟล์โครงสร้างกัน การตั้งค่าในไฟล์โครงสร้างมีองค์ประกอบสำคัญดังนี้ [48]

คีย์	การกำหนด	คีย์ที่อยู่ภายใต้
hostname	ชื่อเรียกของโนด	-
run_cmd	คำสั่งที่ใช้รันหลังเปิดเครื่อง	-
k3os	การตั้งค่าสำหรับเคสามโอเอส	-
k3s_args	การตั้งค่าเคสามเอส	k3os
token	โทเคน	k3os
password	รหัสผ่านของโนด	k3os
server_url	ที่อยู่ของมาสเตอร์โนด (https://$[master_node_ip]:$[port])	k3os

การตั้งค่าเป็นมาสเตอร์โนดในคลัสเตอร์ที่มีมาสเตอร์โนดมากกว่า 1 การสร้างเครื่องมาสเตอร์โนดเครื่องแรก จำเป็นต้องกำหนดตัวแปรพิเศษ “--cluster-init” ในตัวเลือกการตั้งค่าเคสามเอสของไฟล์โครงสร้าง ตัวอย่างไฟล์โครงสร้างเครื่องมาสเตอร์โนดตัวที่ 1

```
hostname: master1
k3os:
  k3s_args:
  - server
  - “--cluster-init”
  token: token
  password: password
```

ตัวอย่างไฟล์โครงสร้างเครื่องมาสเตอร์โนดตัวที่ 2 เป็นต้นไป (โทเคนต้องเหมือนของมาสเตอร์ตัวแรก) ไม่ต้องกำหนด “--cluster-init” ใน k3s_args แล้ว

```
hostname: master2
k3os:
  k3s_args:
  - server
  token: token
  password: password
  server_url: https://$[master_node_ip]:$[port]
```

การติดตั้งและใช้งานเคสามดี

เคสามดี ต้องติดตั้งดอกเกอร์เอนจินก่อน จากนั้นจึงติดตั้งเคสามดี และต้องติดตั้งคิวซีทีแอล เพื่อควบคุมคิวเบอร์เนเทสเพิ่ม คำสั่งที่ติดตั้งเคสามดีมีดังนี้

```
> apk add curl bash
> curl -s https://raw.githubusercontent.com/rancher/k3d/main/install.sh | bash
> curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
หรือ
> curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"
> install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
> rm kubectl
```

ตัวอย่างการสร้างคิวเบอร์เนเทสด้วยเคสามดี

k3d cluster create c1 -p "30001-30100:30001-30100@loadbalancer"	
ชื่อเรียกคลัสเตอร์	c1
การฟอร์เวิร์ดพอร์ท	เชื่อมโฮสพอร์ทตั้งแต่ 30001-30100 มายังคลัสเตอร์ในช่วงหมายเลขเดียวกัน

หมายเหตุ: การเชื่อมโฮสพอร์ทไม่สามารถทำได้หลังการสร้างคลัสเตอร์แล้ว

k3d cluster create c1 -i rancher/k3s:v1.21.1-k3s1 -p "10021-10030:10021-10030@loadbalancer"	
ชื่อเรียกคลัสเตอร์	c1
การฟอร์เวิร์ดพอร์ท	เชื่อมโฮสพอร์ทตั้งแต่ 10021-10030 มายังคลัสเตอร์ในช่วงหมายเลขเดียวกัน
รุ่นเคสามเอส	rancher/k3s:v1.21.1-k3s1 (ตามในที่มีในดอกเกอร์ฮับ)

การสลับการสั่งการคลัสเตอร์ปลายทาง

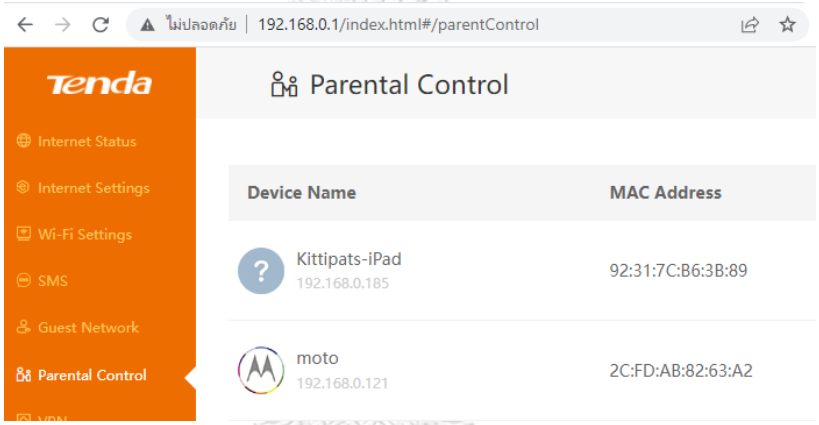
แสดงคลัสเตอร์ปลายทางทั้งหมด	kubectl config get-contexts
สลับไปสั่งการคลัสเตอร์อื่น	kubectl config use-context [ชื่อคอนเทกซ์]

การจัดการคลัสเตอร์

สร้างคลัสเตอร์ใหม่	k3d cluster create [ชื่อคลัสเตอร์]
แสดงคลัสเตอร์ทั้งหมด	k3d cluster list
ลบคลัสเตอร์	k3d cluster delete [ชื่อคลัสเตอร์]

การตั้งค่าเครือข่ายในการติดตั้งคิวเบอร์เนเทส

องค์ประกอบสำคัญในการเชื่อมต่อเครือข่ายได้แก่ ชื่อ Network interface, IP-Address, IP-Gateway,Subnet-mask, และ Name server ในบางกรณีเช่น เครือข่ายท้องถิ่น (LAN) เดียวกัน มีอุปกรณ์ DHCP server ที่ทำหน้าที่จ่าย IP-Address, IP-Gateway, และ subnet-mask ให้โดยอัตโนมัติ จึงอาจละให้ DHCP จดจำการจ่าย IP-Address ตาม Mac-address ของโนดนั้น เพื่อความสะดวกในการสงวนหมายเลย IP-Addess ซึ่งไม่ว่าโนดจะถูกติดตั้งใหม่หรือเปลี่ยนระบบปฏิบัติการ DHCP server จะจ่าย IP-Address หมายเลขเดิมตาม Mac-address ของเครื่อง ซึ่งหากผู้ดูแลระบบต้องการแก้ไข IP-Address ที่ผูกกับ Mac-address จะต้องแก้ไขบนรายการของ DHCP server รูปที่ 49 แสดงตัวอย่างรายการ IP-Address และ Mac Address โดย DHCP server บน Router Tenda



รูปที่ 49 ตัวอย่างรายการไอพีแอดเดรสและแมคแอดเดรสที่เคยเข้าร่วมเครือข่าย

สำหรับกรณีทั่วไป เช่นกรณีคลัสเตอร์ที่ติดตั้งอยู่ใน Uninet ตามโครงการ IoTCloudServe@TEIN จะไม่มี DHCP ซึ่งหน่วยงาน Uninet กำหนดช่วงของ IP-Address ที่สามารถใช้งานได้ดังนี้

- IP-Address range: 202.28.193.97 - 202.28.193.126
- Subnet-mask: 255.255.255.224 และกำหนด 202.28.193.98 สงวนให้ IP-Gateway

สิ่งสำคัญในการตั้งค่า Network คือ Network interface name ซึ่งเป็นชื่อจำเฉพาะของแต่ละโนดโดยไม่เปลี่ยนแปลง ผู้ดูแลระบบสามารถตรวจสอบรายการ Network interface ได้ผ่านคำสั่งที่ Bundle เข้ามาใน K3OS ในระหว่างก่อนการติดตั้งระบบปฏิบัติการ คำสั่งที่ใช้ตรวจสอบคือ

```
sudo connmanctl services
```

ตัวอย่างผลลัพธ์

```
*AO Wired                ethernet_ac1f6bbb9de8_cable
```

จากตัวอย่างผลลัพธ์นี้ ปรากฏชื่อ Network interface name จำนวน 1 รายการได้แก่ ethernet_ac1f6bbb9de8_cable ซึ่งจะใช้เป็นค่าในการตั้งค่าเครือข่าย การตั้งค่าเครือข่าย จะใช้คำสั่งดังนี้

```
sudo connmanctl config [network-interface-name] --ipv4 manual [ip-address]
[subnet-mask] [ip-gateway] --nameservers [ip-name-server]
```

ตัวอย่าง

```
sudo connmanctl config ethernet_ac1f6bbb9de8_cable --ipv4 manual
202.28.193.102 255.255.255.224 202.28.193.98 --nameservers 8.8.8.8
```

จากนั้น ให้ restart network service เพื่อ apply การตั้งค่า

```
sudo service connman restart
```

จากนั้นให้ทดสอบผลการตั้งค่าว่าสามารถออกอินเทอร์เน็ตได้หรือไม่ การตั้งค่า Network มีความสำคัญทั้งในระยะก่อนการติดตั้ง K3OS และหลังการติดตั้ง K3OS เนื่องจากในระยะก่อนการติดตั้ง K3OS สามารถตรวจสอบสถาปัตยกรรมของ Server เพื่อโหลดสถาปัตยกรรมที่ถูกต้องมาทดแทนตัวติดตั้งเดิมหากไม่สอดคล้องอัตโนมัติ ส่วนการตั้งค่า Network หลังการติดตั้งระบบปฏิบัติการ K3OS มีความสำคัญเนื่องจาก K3OS จะไม่จดจำการตั้งค่า Network เมื่อมีการ Restart ระบบ ดังนั้นจึงจำเป็นต้องกำหนดการตั้งค่า Network ลงในชุด Configuration ของ K3OS ภายใต้ชุดคำสั่ง run_cmd ใน config.yaml เพื่อให้ตั้งค่า Network ทุกครั้งที่ Restart ดังนี้

```
run_cmd:
- "sudo connmanctl config ethernet_ac1f6bbb9de8_cable --ipv4 manual
202.28.193.102 255.255.255.224 202.28.193.98 --nameservers 8.8.8.8"
- "sudo service connman restart"
```

การติดตั้งอัลไพน์ ลินุกซ์บนเครือข่ายการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย

การติดตั้งอัลไพน์ ลินุกซ์ มีขั้นตอนสำคัญในการตั้งค่าพร็อกซี่ของเครือข่ายภายใน ในระหว่างการติดตั้ง ต้องกำหนดสิ่งที่ต้องตั้งค่าดังนี้

สิ่งที่ต้องตั้งค่า	ตัวอย่าง
IP Address	10.205.65.231
Subnet mask	255.255.255.0
Gateway IP	10.205.65.1
Name server	10.20.222.22
Proxy	http://proxy.egat.co.th:8080

โดย name server และ proxy มักกำหนดเป็นค่าดังตัวอย่างนี้เหมือนกัน อย่างไรก็ตาม ในระหว่างการติดตั้งช่วงแรก การเรียกค้นโปรแกรมเสริมที่จำเป็น จะยังไม่สามารถดึงรายการโปรแกรมเสริมโดยเฉพาะจะต้องใช้ในการติดตั้งดอกเกอร์มาได้ ให้มีการกำหนดที่อยู่ของรายการโปรแกรมเสริมโดยตรงดังนี้

```
กรณีการดึงรายการโปรแกรมเสริมไม่ผ่าน ให้กำหนดเองโดยใส่ตัวเลือก “e” เพื่อแก้ไขที่การค้นรายการโปรแกรมเสริม จากนั้นให้ใส่ที่อยู่ต่อไปนี้

https://download.nus.edu.sg/mirror/alpine/edge/main
https://download.nus.edu.sg/mirror/alpine/edge/community
https://download.nus.edu.sg/mirror/alpine/edge/testing
https://download.nus.edu.sg/mirror/alpine/v3.14/main
https://download.nus.edu.sg/mirror/alpine/v3.14/community
```

จากนั้นให้ดำเนินการติดตั้งดอกเกอร์ด้วยคำสั่งดังนี้

```
apk add docker
apk update
rc-update add docker boot
service docker start
service docker status
```

จากนั้นจึงทดสอบการใช้งานดอกเกอร์ทั่วไป ในบางกรณีหากดอกเกอร์ไม่สามารถเรียกใช้งานผ่านพร็อกซี่ได้โดยตรง (ในดอกเกอร์เวอร์ชันเก่า) มีแนวทางการตั้งค่าพร็อกซี่ของดอกเกอร์ 2 แนวทาง คือ

1. แก้ไขที่ไฟล์ /etc/conf.d/docker โดยใช้คำสั่งดังนี้

```
vi /etc/conf.d/docker
## จากนั้น insert  บรรทัดสุดท้ายว่า
export HTTP_PROXY=”http://proxy.egat.co.th:8080”
export HTTPS_PROXY=”http://proxy.egat.co.th:8080”
## แล้วเซฟไฟล์โดยใช้คำสั่ง :wq จากนั้นให้ reboot เครื่อง
```

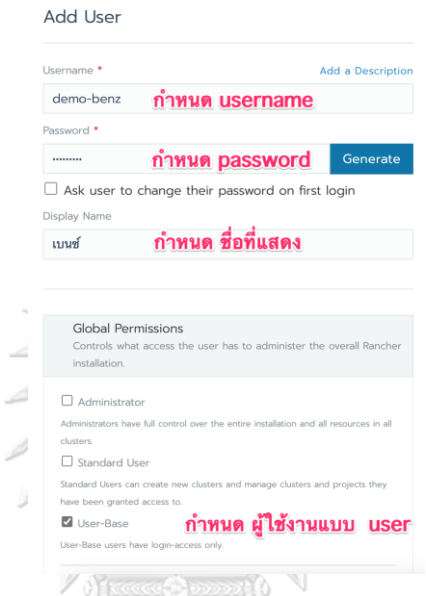
2. หากยังไม่สำเร็จ ให้แก้ไฟล์ ~/.docker/config.json โดยใช้คำสั่งดังนี้

```
~/.docker/config.json
## จากนั้นใส่ข้อมูลดังนี้
{
 "proxies":
 {
   "default":
   {
     "httpProxy": ”http://proxy.egat.co.th:8080”,
     "httpsProxy": ”http://proxy.egat.co.th:8080”
   }
 }
}
## แล้วเซฟไฟล์โดยใช้คำสั่ง :wq จากนั้นให้ reboot เครื่อง
```

ภาคผนวก ข การทดสอบสถาปัตยกรรมระบบจัดการผู้ใช้งานและการจัดลำดับชั้นของทรัพยากร

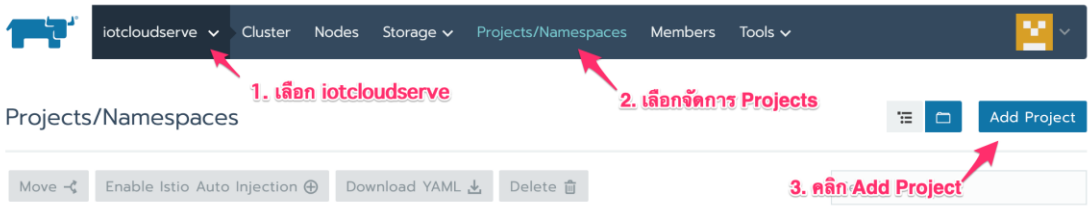
การทดสอบการจัดการผู้ใช้งานบนรานเชอร์

มีการสร้างผู้ใช้งานใหม่สำหรับ 3 คนได้แก่ demo-benz demo-bank และ demo-bas บนรานเชอร์โดยกำหนด username password ชื่อที่แสดง และกำหนดเป็นผู้ใช้งานแบบ User-base ดังรูปที่ 50 เป็นตัวอย่างการสร้างผู้ใช้ demo-benz

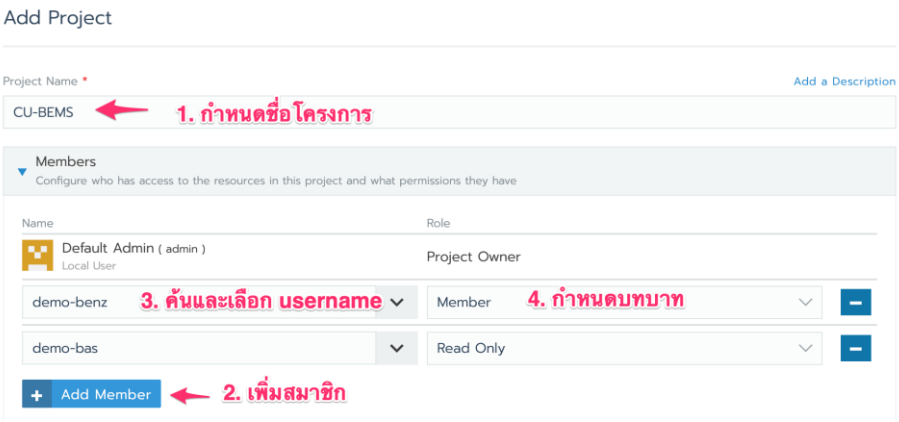


รูปที่ 50 การสร้างผู้ใช้งาน demo-benz

เมื่อสร้างผู้ใช้งานครบทั้ง 3 บัญชี จึงสร้างโครงการใหม่บน IoTcloudServe@TEIN รูปที่ 51 แสดงขั้นตอนการเข้าสู่หน้าการสร้างโครงการใหม่บนรานเชอร์ รูปที่ 52 แสดงการสร้างโครงการและสิทธิการเข้าถึงให้สมาชิกโครงการ CU-BEMS โดยการกำหนดชื่อโครงการ กำหนดสมาชิก และกำหนดบทบาทในโครงการ

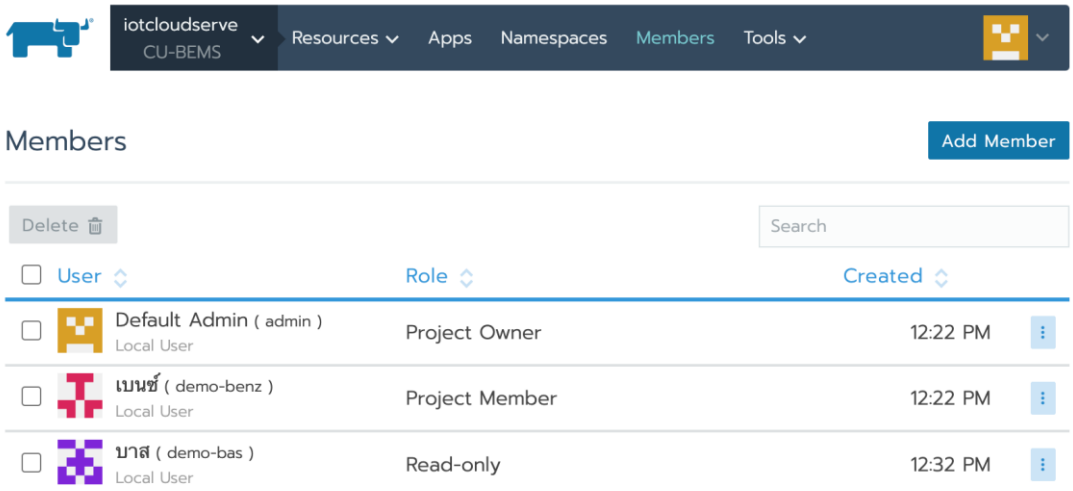


รูปที่ 51 ขั้นตอนเข้าหน้าสร้างโครงการใหม่บนรานเชอร์

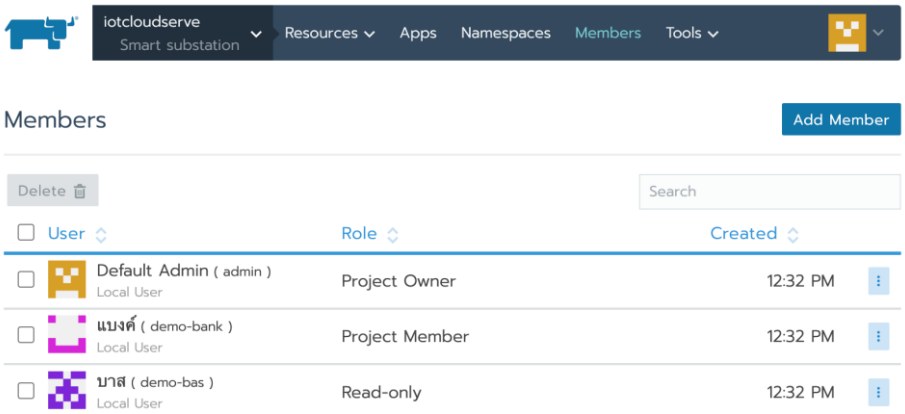


รูปที่ 52 การเพิ่มสิทธิให้สมาชิกเข้าถึงโครงการ

สำหรับกรณีบัญชี demo-bas มีการกำหนดให้สามารถดูภาระงานได้อย่างเดียว โดยไม่อาจบำรุงรักษาภาระอย่างอื่นเช่น สร้าง แก้ไข หรือลบได้ จากในรูปที่ 52 ให้เลือกบทบาทเป็นแบบ Read Only สำหรับกรณีของนายบาส ผลการสร้างและกำหนดสิทธิโครงการ CU-BEMS และ สถานีไฟฟ้าอัจฉริยะ เป็นดังรูปที่ 53 และ รูปที่ 54 ตามลำดับ ที่แสดงรายการบัญชีที่สามารถเข้าถึงโครงการได้และบทบาทในการใช้งาน



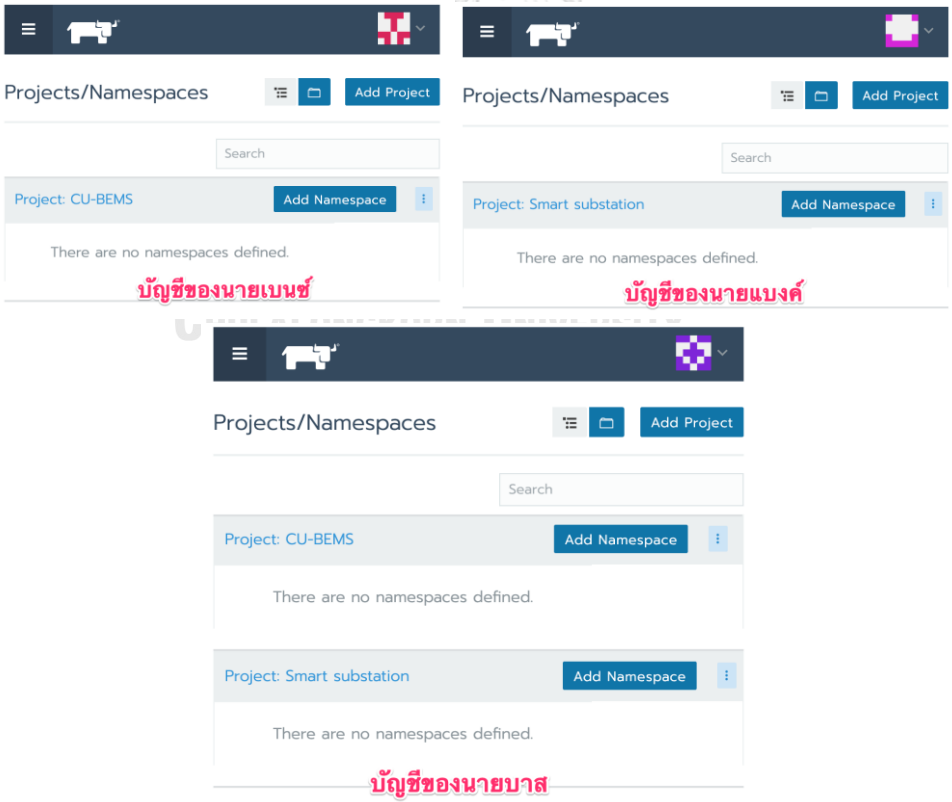
รูปที่ 53 ผลการสร้างและกำหนดสิทธิการใช้งานโครงการ CU-BEMS



รูปที่ 54 ผลการสร้างและกำหนดสิทธิโครงการ สถานีไฟฟ้าอัจฉริยะ

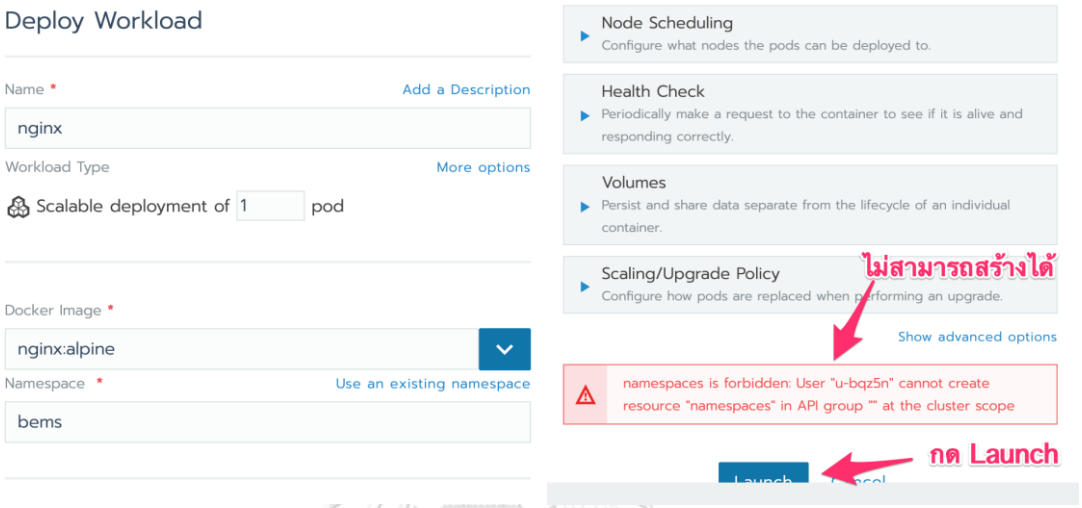
ผลการทดสอบการจัดการผู้ใช้งานบนรานเซอร์

เมื่อผู้ใช้งานแต่ละบัญชีเข้าสู่ระบบ จะสามารถเข้าถึงโครงการได้ต่างกัน โดยนายเบนซ์สามารถเข้าถึงได้เฉพาะโครงการ CU-BEMS นายแบงค์เข้าถึงได้เฉพาะโครงการ สถานีไฟฟ้าอัจฉริยะ และนายบาสเข้าถึงได้ทั้ง 2 โครงการ รูปที่ 55 แสดงผลการจัดการสิทธิที่ทำให้แต่ละบัญชีสามารถเข้าถึงโครงการต่าง ๆ ได้ไม่เหมือนกัน

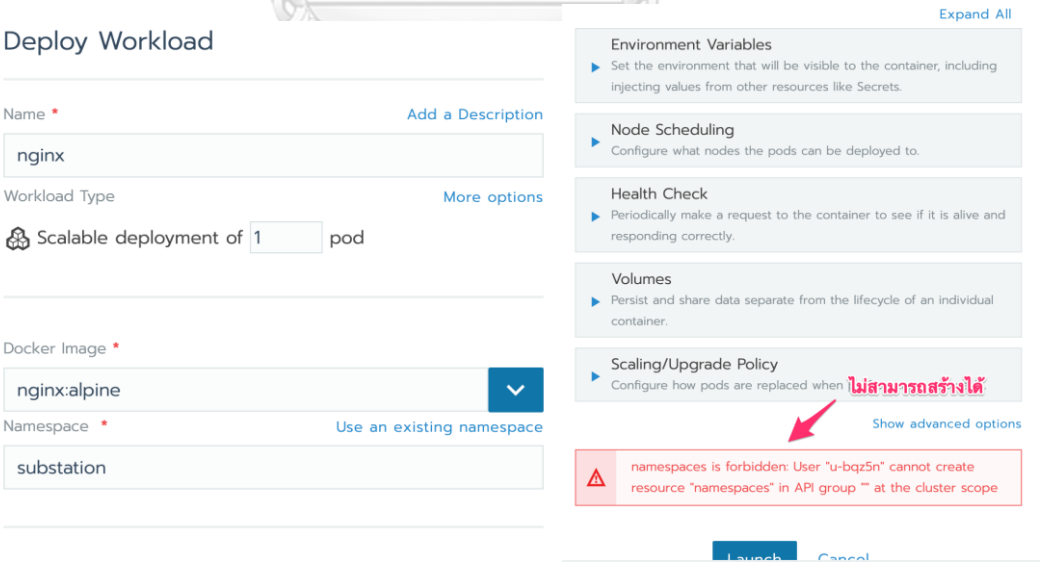


รูปที่ 55 ผลการจัดสิทธิการเข้าถึงการใช้งานบนรานเซอร์

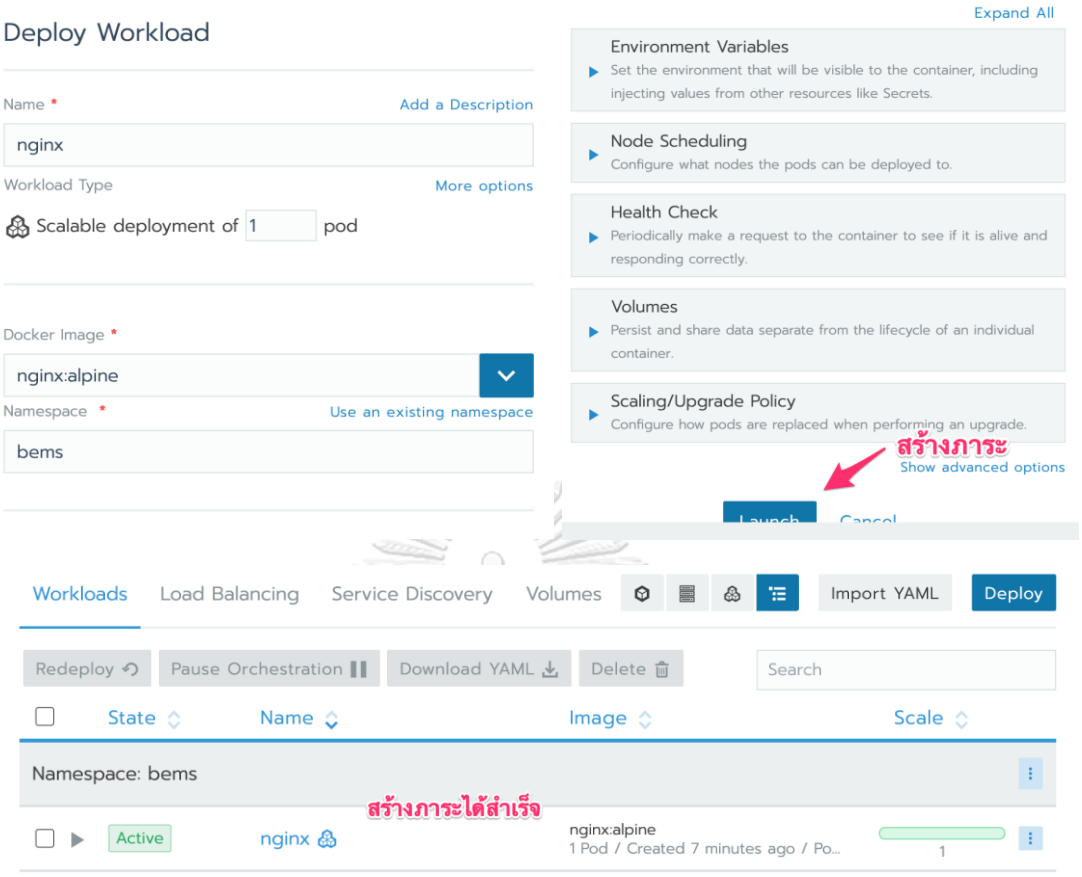
การทดสอบโดยใช้บัญชีนายบาส สร้างภาระ (workload) บนโครงการ CU-BEMS ปรากฏข้อความแจ้งเตือนไม่สามารถสร้างได้ ดังรูปที่ 56 เนื่องจากมีสิทธิเฉพาะการดูได้อย่างเดียว และบนโครงการ สถานีไฟฟ้าอัจฉริยะ ปรากฏข้อความแจ้งเตือนไม่สามารถสร้างภาระเพิ่มได้เช่นกัน ดังรูปที่ 57 ในขณะที่บัญชีนายเบนซ์ สามารถสร้างภาระบนโครงการ CU-BEMS ได้ปกติดังรูปที่ 58



รูปที่ 56 การทดสอบสร้างภาระโดยผู้ใช้งาน demo-bas บนโครงการ CU-BEMS

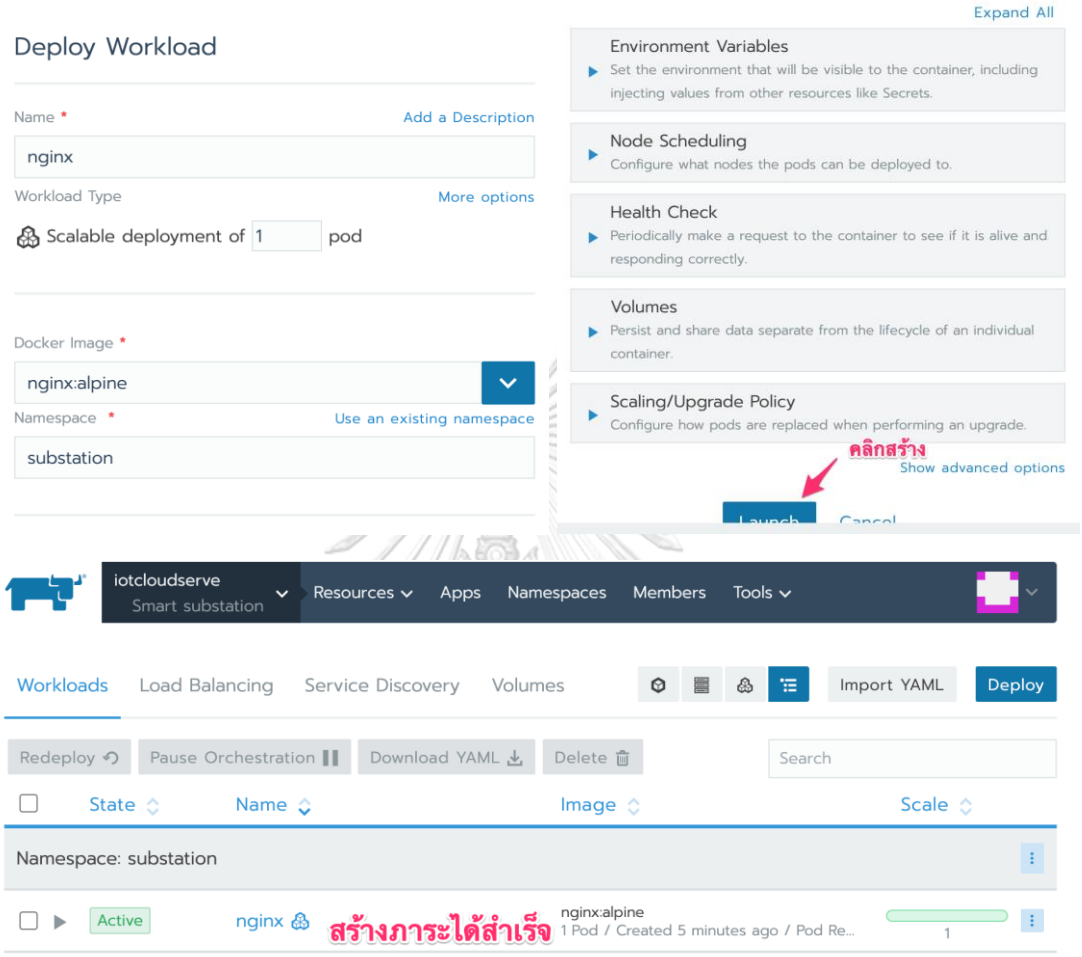


รูปที่ 57 การทดสอบสร้างภาระโดยผู้ใช้งาน demo-bas บนโครงการ สถานีไฟฟ้าอัจฉริยะ



รูปที่ 58 การทดสอบสร้างภาระโดยผู้ใช้งาน demo-benz บนโครงการ CU-BEMS

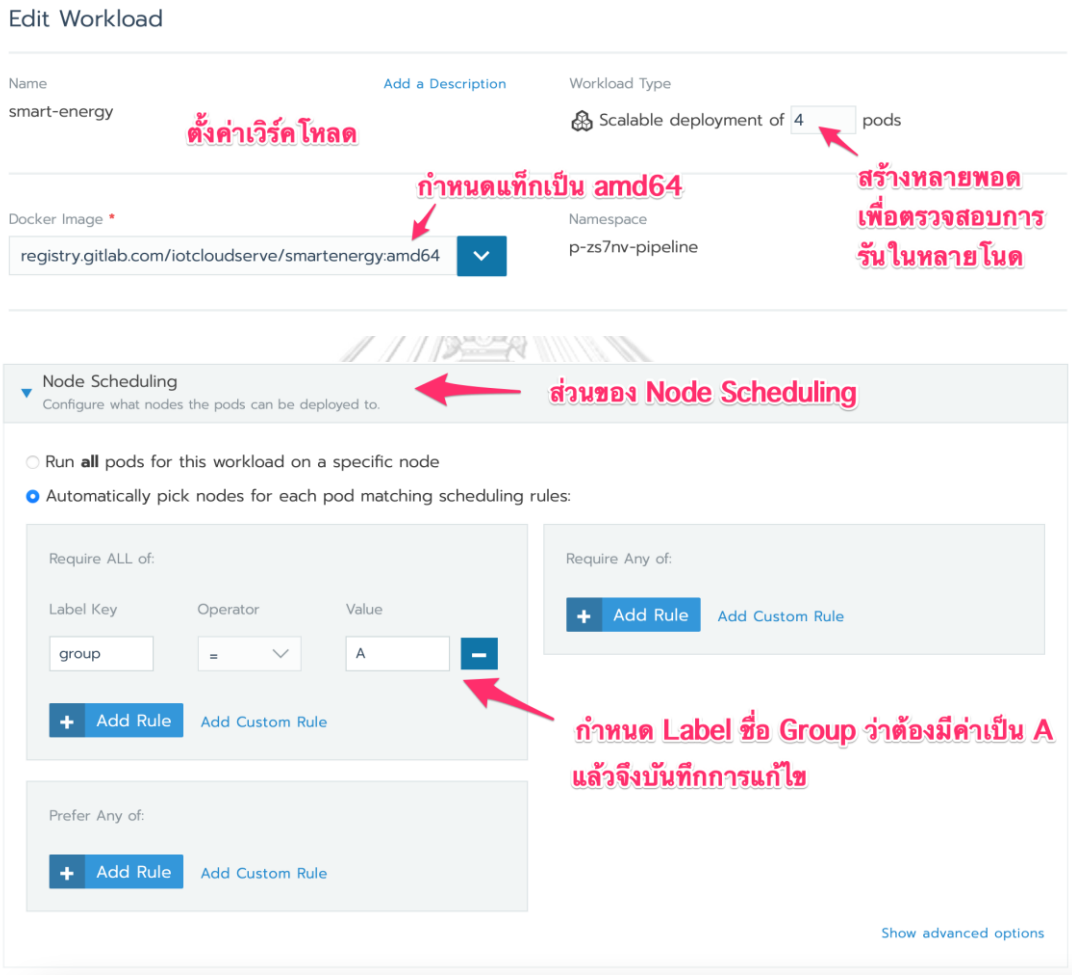
บัญชีนายแบงค์ สามารถสร้างภาระบนโครงการ สถานีไฟฟ้าอัจฉริยะ ได้ปกติดังรูปที่ 59



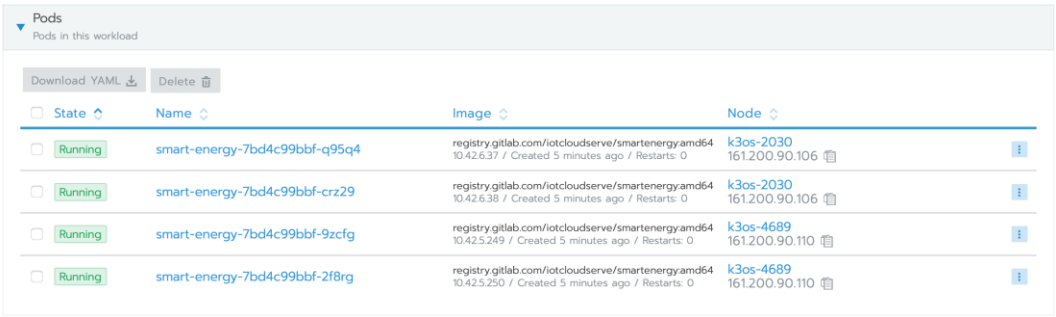
รูปที่ 59 การทดสอบสร้างภาระโดยผู้ใช้งาน demo-bank บนโครงการ CU-BEMS

การทดสอบการรันบนโนดที่เข้ากันได้

ในการทดสอบการรันคอนเทนเนอร์ registry.gitlab.com/iotcloudserve/smartenergy คอนเทนเนอร์นี้ได้มีการสร้างคอนเทนเนอร์ให้รองรับได้ทั้ง 2 สถาปัตยกรรมโดยแยกอิมเมจกันตามชื่อแท็ก แท็กแบบ amd64 จะรันได้เฉพาะบนโนดที่มีสถาปัตยกรรม amd64 เท่านั้น และได้จัดกลุ่มของโนดนี้เป็นกลุ่ม A การตั้งค่าภาระให้รันบนโนดกลุ่ม A สามารถทำผ่านรานเชอร์ดังรูปที่ 60

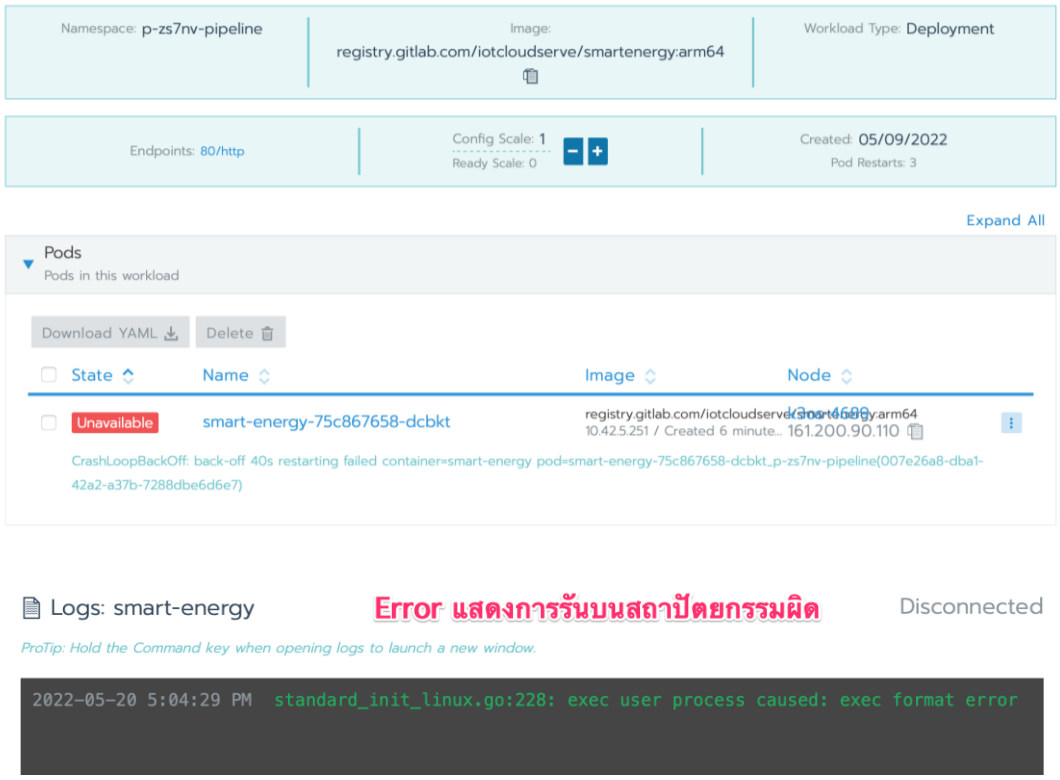


รูปที่ 60 การกำหนดเงื่อนไขการรันบนโนดตามป้าย ที่กำหนดของภาระ



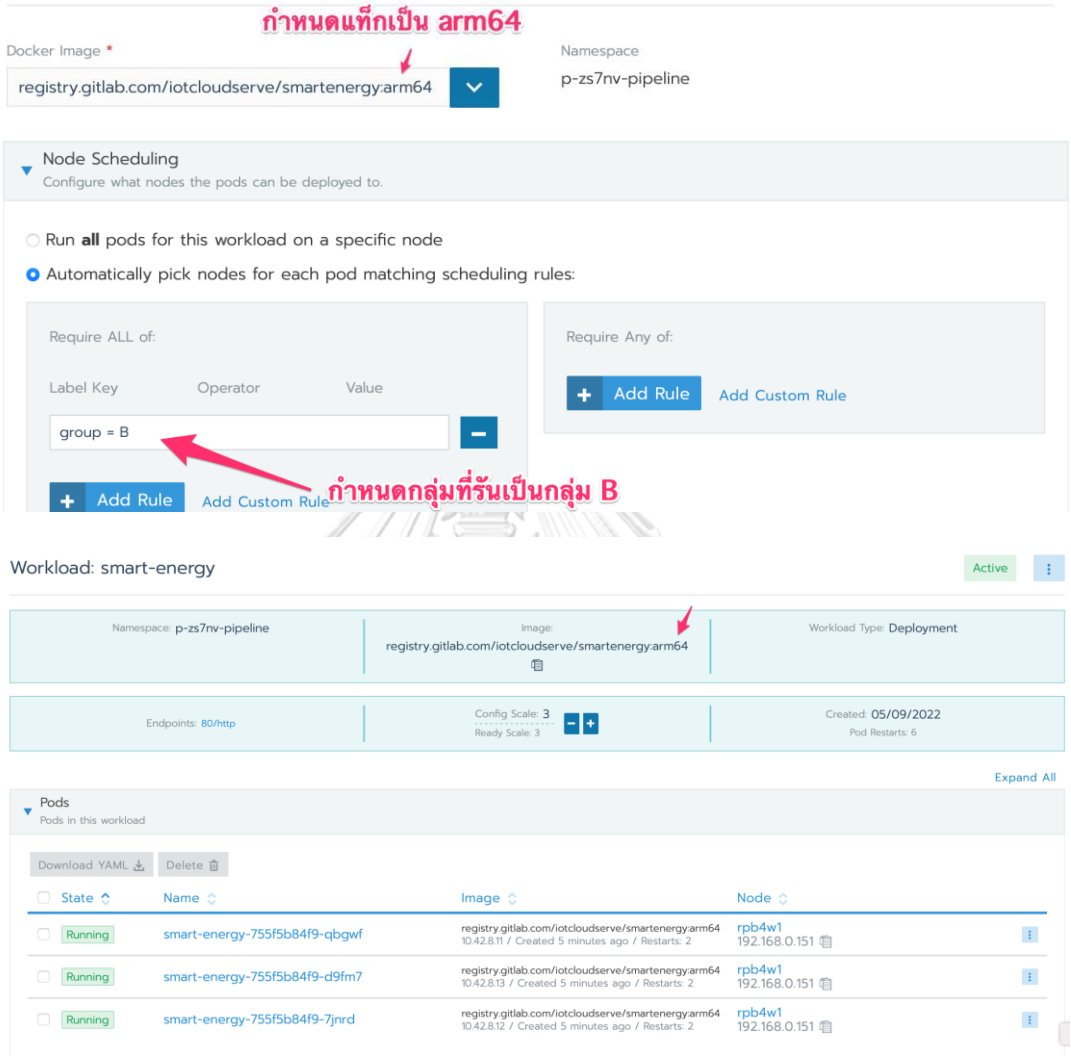
รูปที่ 61 ผลการทดสอบการรันคอนเทนเนอร์บนโนดกลุ่ม A

ผลการทดสอบคอนเทนเนอร์ที่รันจะกระจายอยู่เฉพาะโนดกลุ่ม A ซึ่งได้แก่เครื่อง k3os-2030 และ k3os-4689 เป็นดังรูปที่ 61 ซึ่งแสดงผลการทดสอบการรันคอนเทนเนอร์ที่รันอยู่เฉพาะโนด k3os-2020 และ k3os-4689 เท่านั้นได้ถูกต้อง กรณีที่หากกำหนดผิด เลือกใช้คอนเทนเนอร์สถาปัตยกรรม arm64 แต่รันในโนดกลุ่ม A จะเกิดปัญหาข้อผิดพลาดเมื่อรันคอนเทนเนอร์บนโนดผิดสถาปัตยกรรมดังรูปที่ 62 โดยมีการระบุเตือนว่าไม่สามารถใช้งานได้



รูปที่ 62 ข้อผิดพลาดเมื่อรันคอนเทนเนอร์บนสถาปัตยกรรมที่ไม่รองรับ

อีกกรณีหนึ่งในกลุ่ม B กำหนดคอนเทนเนอร์ที่รันเป็น arm64 และกำหนดกลุ่มที่รันเป็นกลุ่ม B จะทำให้คอนเทนเนอร์ทำงานได้ปกติดังรูปที่ 63



รูปที่ 63 การรันคอนเทนเนอร์สถาปัตยกรรม arm64 บนกลุ่ม B

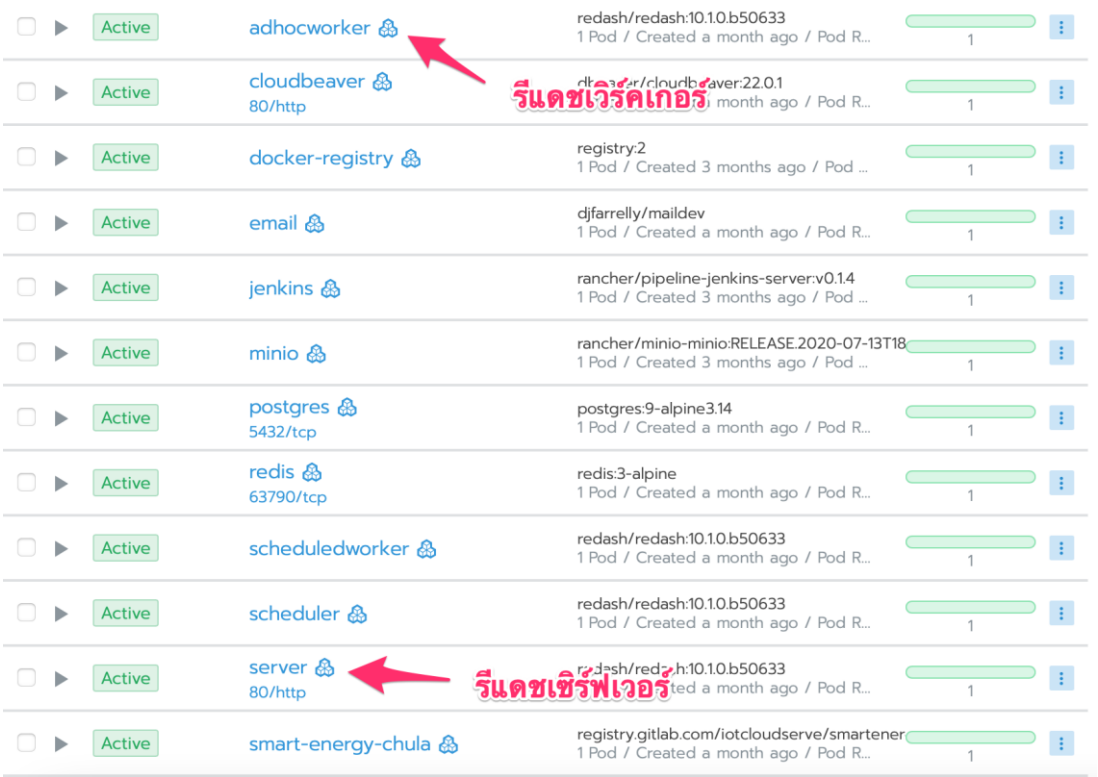
ภาคผนวก ค การทดสอบสถาปัตยกรรมระบบที่มีหลายคลัสเตอร์ร่วมสมาพันธ์

การทดสอบประโยชน์จากการร่วมสมาพันธ์คลัสเตอร์

บนไอโอทีคลาวด์เซิร์ฟ (คลัสเตอร์ A) มีการรันคอนเทนเนอร์รีแดชเซิร์ฟเวอร์และรีแดชเวิร์คเกอร์ ส่วนคลัสเตอร์ B มีการสร้างเฉพาะคอนเทนเนอร์รีแดชเวิร์คเกอร์ ดังตารางที่ 27 รูปที่ 64 แสดงรายการคอนเทนเนอร์ของรีแดชบนคลัสเตอร์ A ที่ประกอบด้วยรีแดชเซิร์ฟเวอร์และรีแดชเวิร์คเกอร์

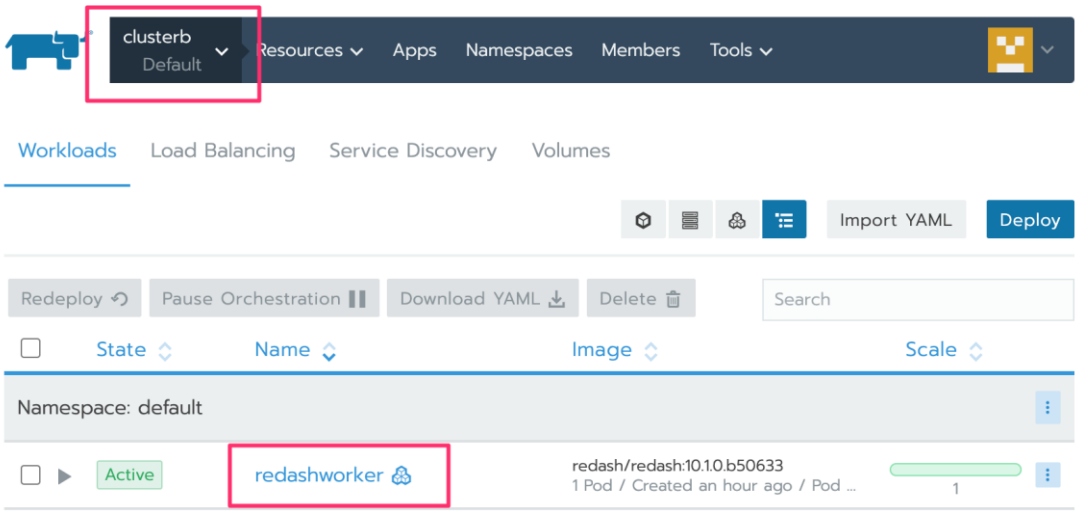
ตารางที่ 27 รายการคอนเทนเนอร์ในแต่ละคลัสเตอร์

คลัสเตอร์	รายการคอนเทนเนอร์
คลัสเตอร์ A (IoTcloudServe)	รีแดชเซิร์ฟเวอร์ และ รีแดชเวิร์คเกอร์
คลัสเตอร์ B	รีแดชเวิร์คเกอร์



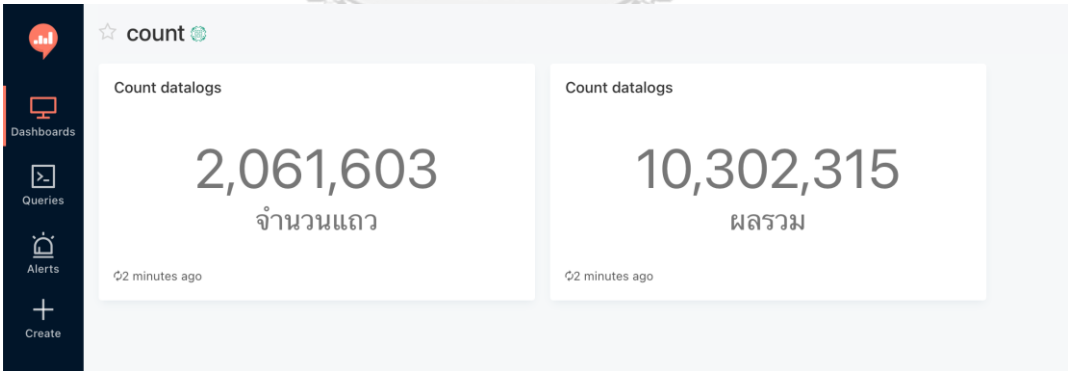
รูปที่ 64 รายการคอนเทนเนอร์ของรีแดชบน IoTcloudServe@TEIN (คลัสเตอร์ A)

ในคลัสเตอร์ B รันเฉพาะคอนเทนเนอร์รีแดชเวิร์คเกอร์ ดังรูปที่ 65



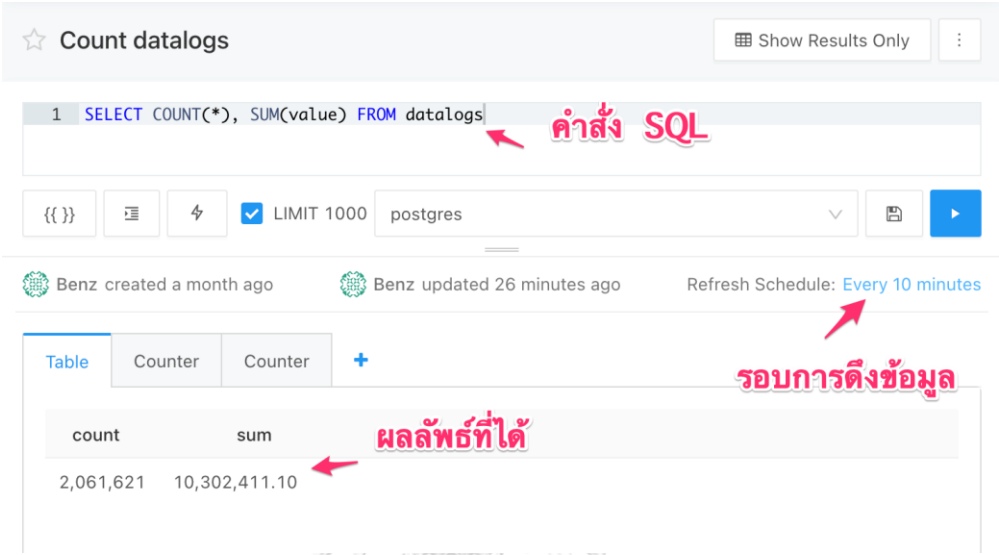
รูปที่ 65 การรันรีแดชคอนเทนเนอร์บนคลัสเตอร์ B

รูปที่ 66 แสดงตัวอย่างการดึงข้อมูลจากฐานข้อมูลภายนอกที่เก็บตัวอย่างข้อมูลพลังงานอัจฉริยะ และแสดงบนกระดานข้อมูลบนโปรแกรมรีแดช ซึ่งมีการจำลองการสร้างข้อมูลจากอุปกรณ์ไอโอทีเข้ามาอย่างต่อเนื่อง โดยมีคำสั่ง SQL ในการดึงข้อมูลจากฐานข้อมูลภายนอกเพื่ออัพเดทข้อมูลทุก 10 นาที รูปที่ 67 แสดงการบันทึกคำสั่ง SQL ในการดึงข้อมูลเพื่อมานำเสนอบนกระดานข้อมูลทุก 10 นาที



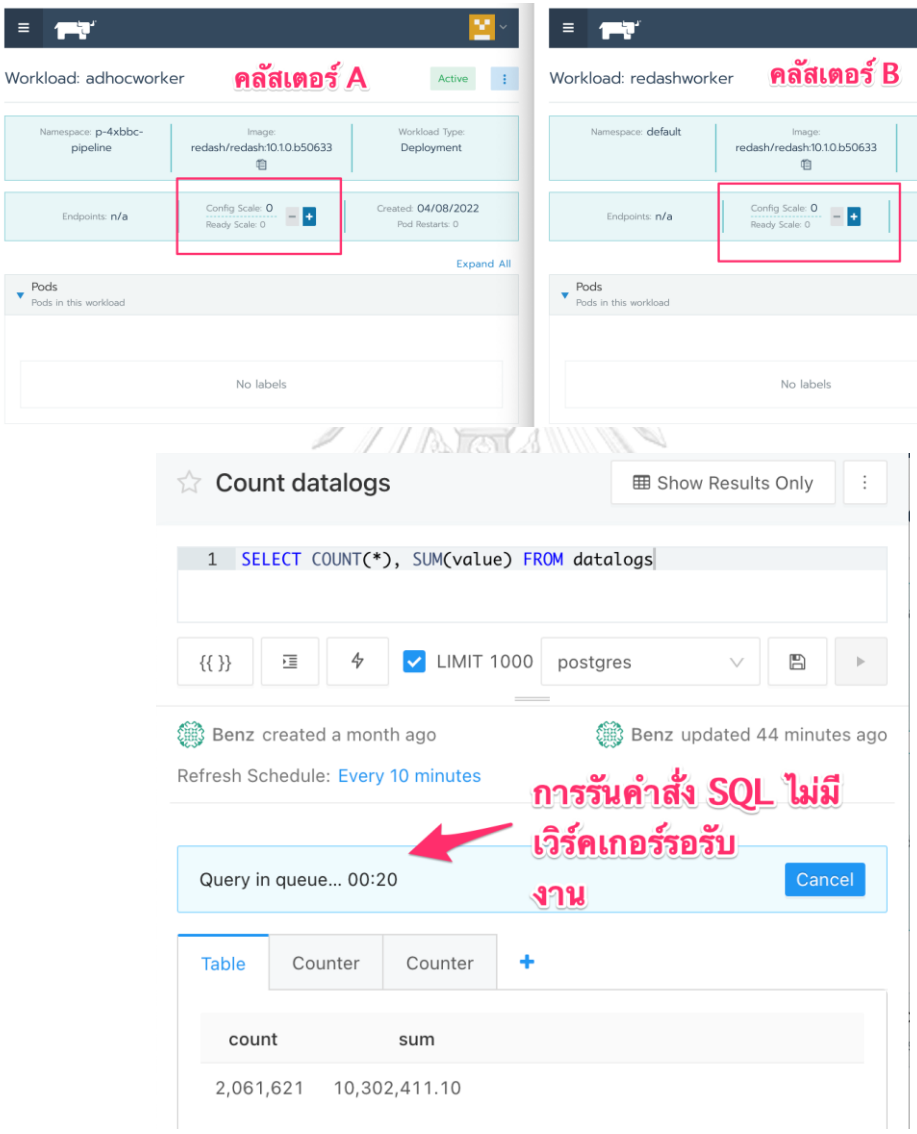
รูปที่ 66 การดึงข้อมูลเพื่อแสดงบนกระดานข้อมูลบนโปรแกรมรีแดช

ทุกครั้งที่มีการอัพเดทข้อมูลเพื่อนำเสนอบนกระดานข้อมูลใหม่ รีแดชเวิร์คเกอร์ที่กระจายอยู่แต่ละคลัสเตอร์จะรอรับงานจากรีแดชเซิร์ฟเวอร์ดังรูปที่ 67 เพื่อทดสอบคุณสมบัติที่รีแดชเวิร์คเกอร์สามารถช่วยกันทำงานได้ จึงได้ทดสอบการเปิดรีแดชเวิร์คเกอร์ในคลัสเตอร์ใดคลัสเตอร์หนึ่งเพียงคอนเทนเนอร์เดียว และทดสอบให้เวิร์คเกอร์ดึงข้อมูลเพื่อประมวลผลใหม่ การทดสอบเป็นดังนี้



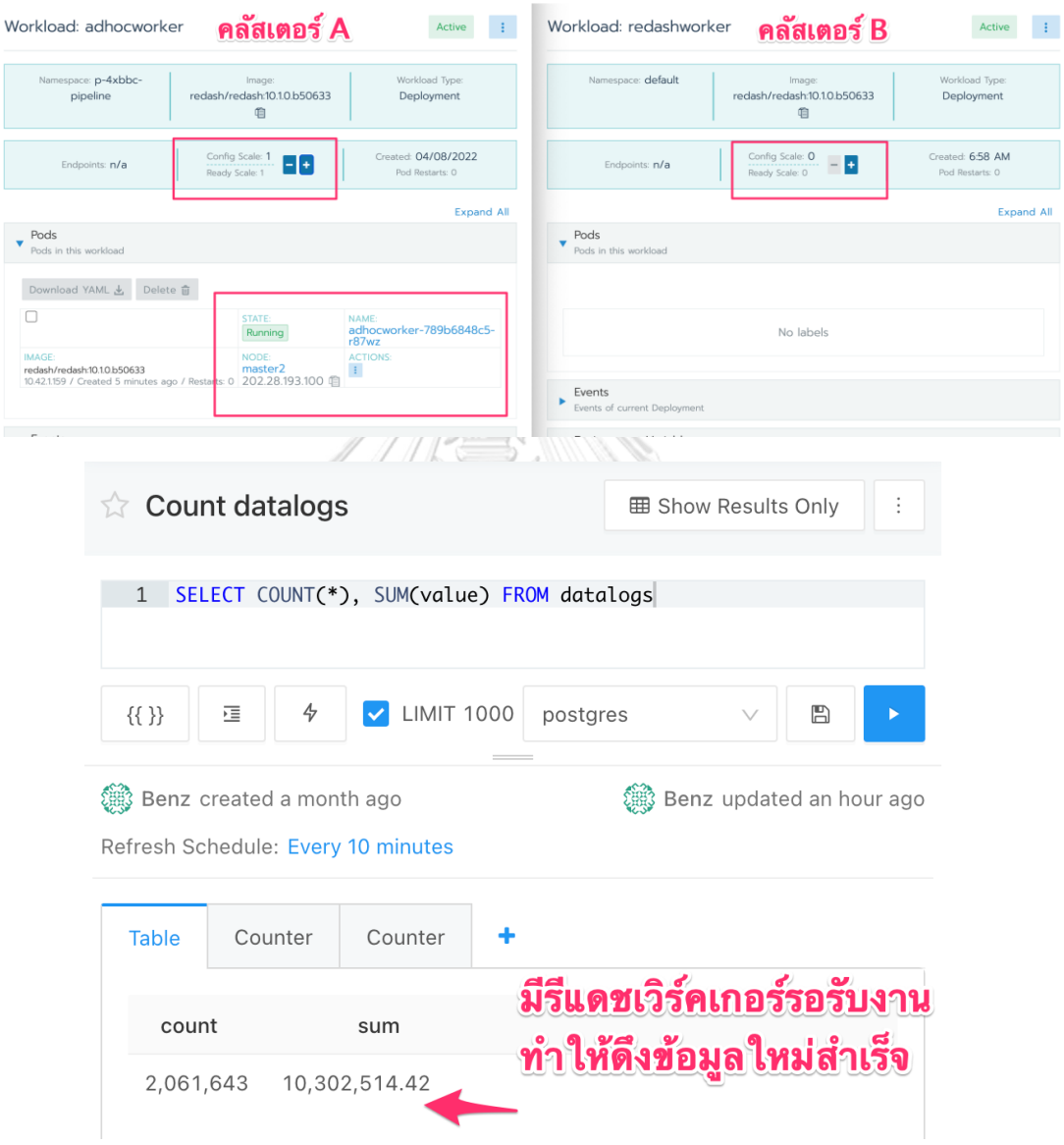
รูปที่ 67 การบันทึกคำสั่ง SQL ในการดึงข้อมูลจากฐานข้อมูลจำลองการเก็บข้อมูลจากอุปกรณ์ไอโอที

การทดสอบที่ 1 หยุดคอนเทนเนอร์ของรีแดชเวิร์คเกอร์ทุกคลัสเตอร์ ผลการทดสอบการอัพเดทข้อมูลบนกระดานข้อมูล ไม่สามารถดึงข้อมูลได้สำเร็จ เนื่องจากไม่มีรีแดชเวิร์คเกอร์รอรับงาน ดังรูปที่ 68 (ข้อมูลที่แสดงในรูปที่ 68 เป็นข้อมูลเก่าที่ไม่ถูกอัพเดทเมื่อมีการรันคำสั่งใหม่ ดังนั้นกรณีที่ไม่มีรีแดชเวิร์คเกอร์รอรับงาน ค่าที่แสดงบนกระดานข้อมูลจะไม่เป็นปัจจุบัน)



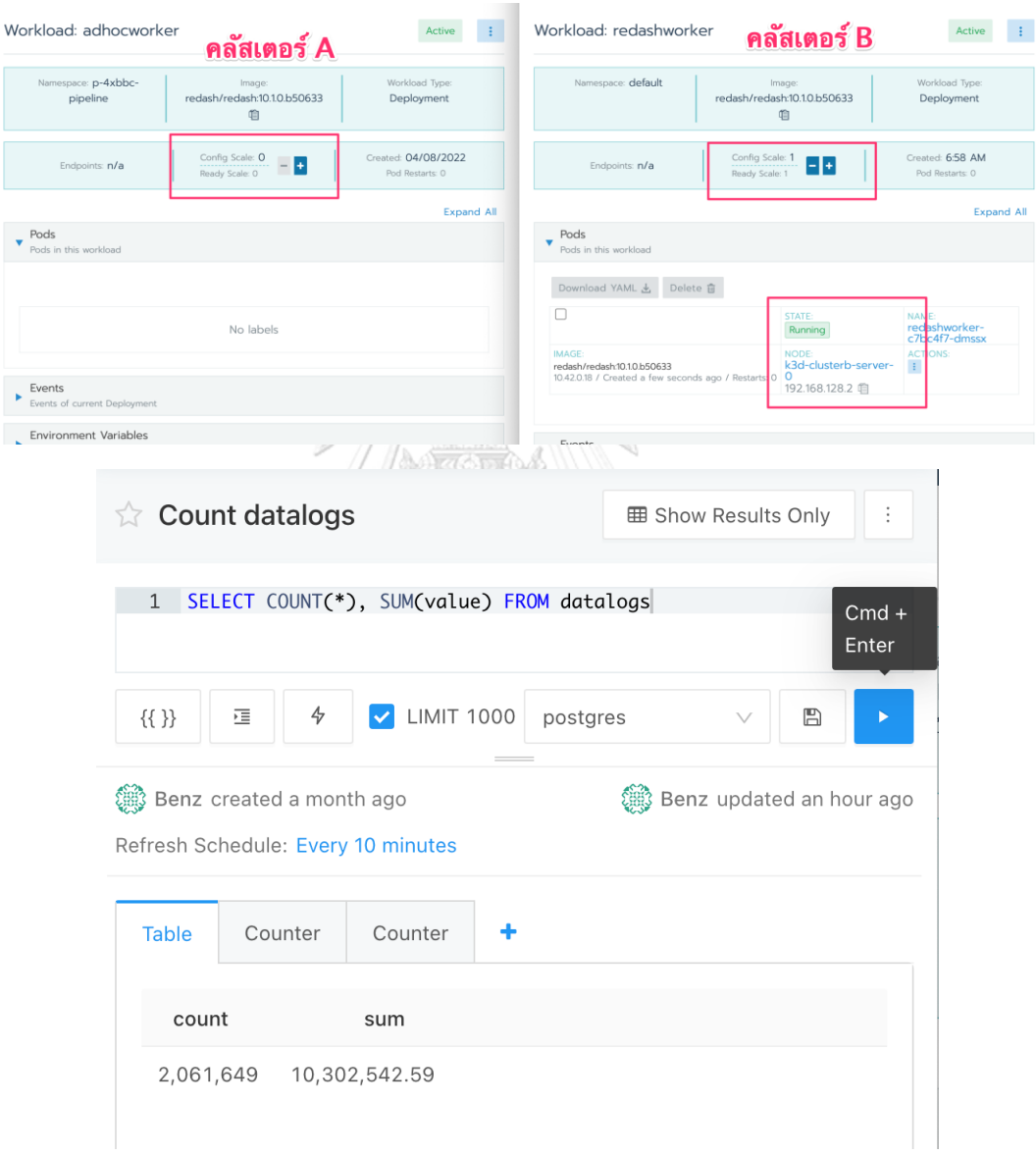
รูปที่ 68 ผลการรันงานอัพเดทข้อมูลกรณีที่ไม่มีรีแดชเวิร์คเกอร์รอรับงาน

การทดสอบที่ 2 การรันรีแดชเวิร์คเกอร์เฉพาะบนคลัสเตอร์ A เมื่อมีรีแดชเวิร์คเกอร์มารอรับงาน จะสามารถทำงานอัพเดทข้อมูลบนกระดานข้อมูลตามคำสั่ง SQL ได้สำเร็จ ผลการทดสอบดังรูปที่ 69



รูปที่ 69 ผลการรันงานอัพเดทข้อมูลกรณีมีรีแดชเวิร์คเกอร์รอรับงานบนคลัสเตอร์ A

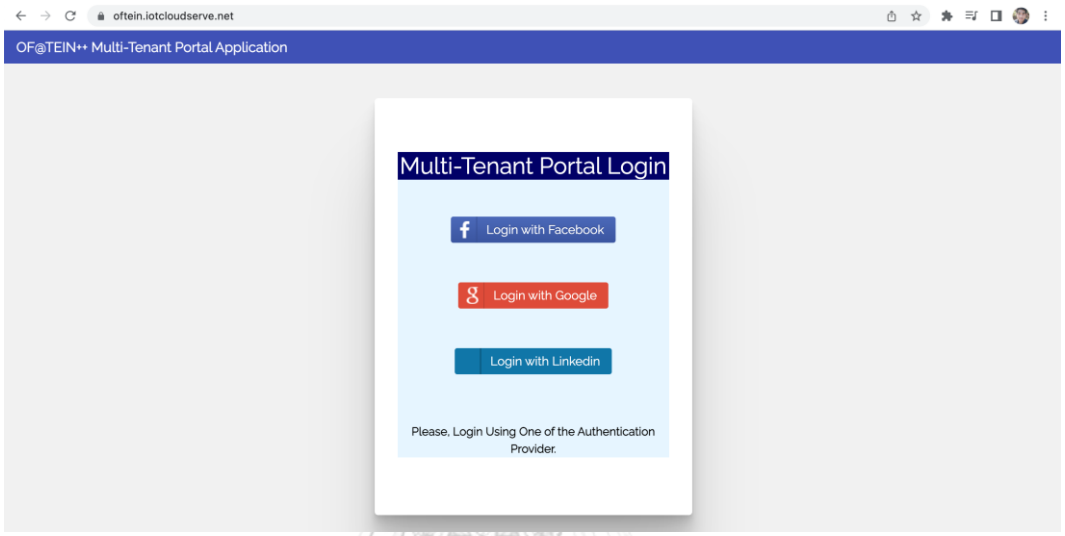
การทดสอบที่ 3 การรันรีแดชเวิร์คเกอร์เฉพาะบนคลัสเตอร์ B เมื่อตรวจสอบการดึงข้อมูลสามารถดึงข้อมูลได้สำเร็จ โดยเปรียบเทียบจากการทดสอบที่ 2 มีข้อมูลบันทึกเพิ่มขึ้นเนื่องจากดึงข้อมูลทีหลัง จากข้อมูล 2,061,643 แถว เพิ่มมาเป็น 2,061,649 แถว ผลการทดสอบเป็นดังรูปที่ 70 การทดสอบการรันรีแดชเวิร์คเกอร์เพื่อช่วยเพิ่มระดับความพร้อมใช้ของกระดานข้อมูลและช่วยในการประมวลผลการอัพเดทข้อมูล สามารถรันบนคลัสเตอร์อื่นที่ร่วมสมาพันธ์ได้



รูปที่ 70 ผลการรันงานอัพเดทข้อมูลกรณีมีรีแดชเวิร์คเกอร์รอรับงานบนคลัสเตอร์ B

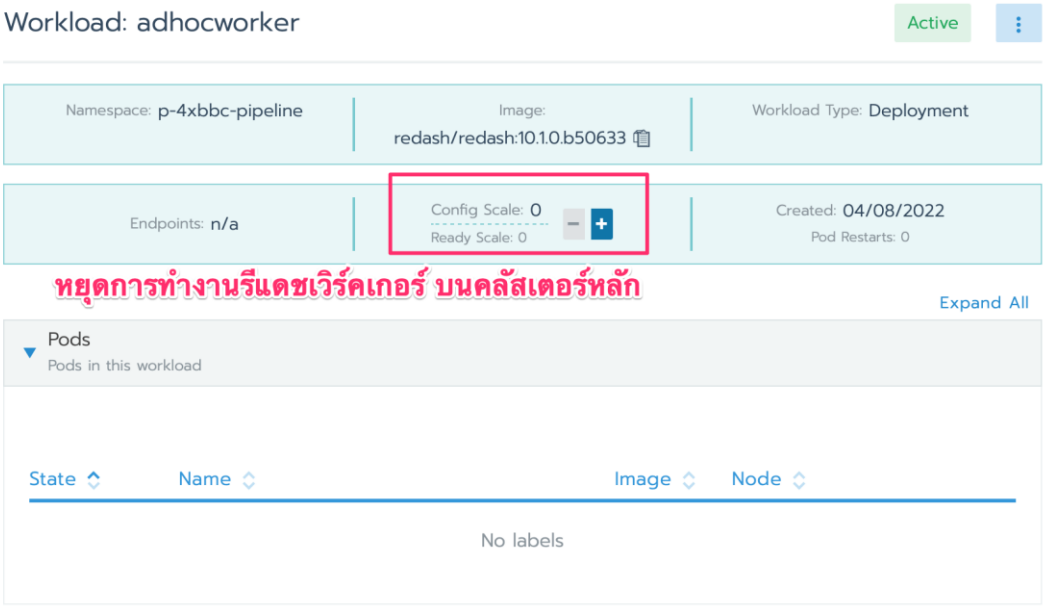
การทดสอบรันรีแดชคอนเทนเนอร์บนระบบ OF@TEIN++

ระบบ OF@TEIN++ ในการสร้างกลไกการร่วมสมาพันธ์คลัสเตอร์ ได้เปิดใช้งานบนเว็บไซต์ oftein.iotcloudserve.net จึงทดสอบการรันรีแดชเวิร์คเกอร์ผ่านกลไก OF@TEIN++ มีการทดสอบต่อไปนี้

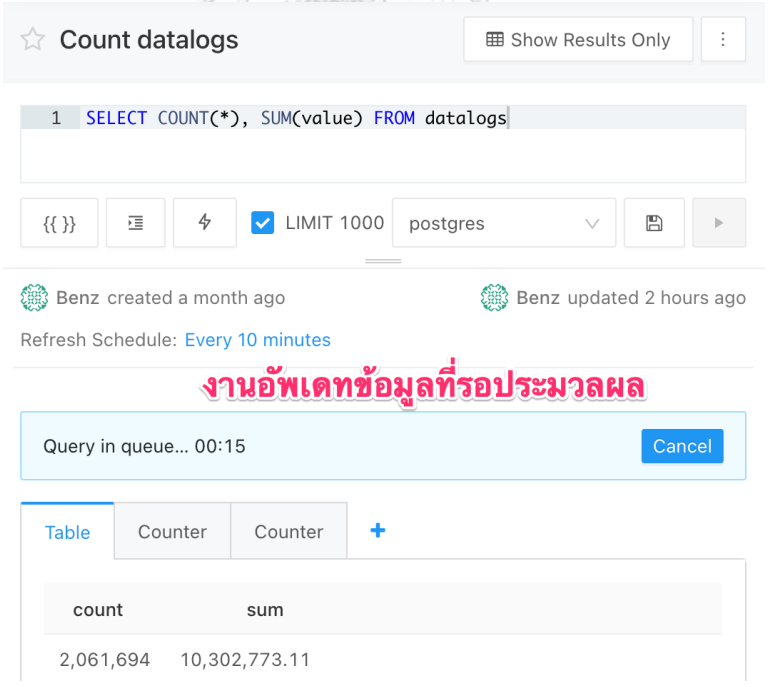


รูปที่ 71 ระบบ OF@TIEN++ บนเว็บไซต์ oftein.iotcloudserve.net

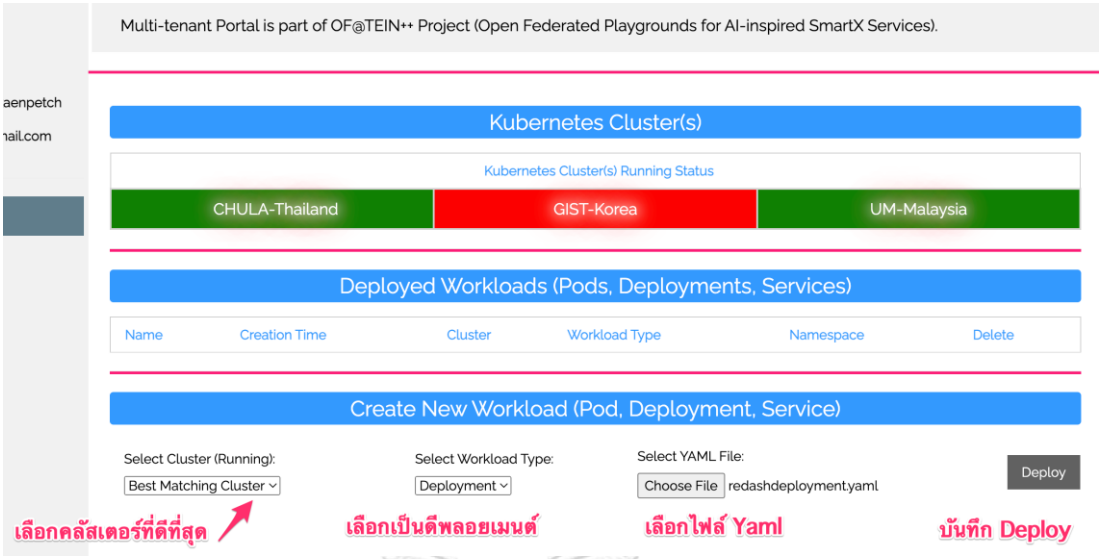
ระบบหลังบ้านที่ตัวแทนทีมจุฬาฯ ในวิทยานิพนธ์นี้ได้พัฒนา [42] สามารถทำงานประสานกับระบบหน้าบ้านที่ทางทีม GIST พัฒนาได้อย่างสมบูรณ์ [43] ดังรูปที่ 71 ในการร่วมสมาพันธ์ของคิวเบอร์เนเทสคลัสเตอร์มีการแชร์เซอร์วิสแอคเคาท์ร่วมกันในระบบหลังบ้านที่ถูกกำหนดสิทธิและจำกัดการเข้าถึงทรัพากรโดยผู้ดูแลระบบของแต่ละคลัสเตอร์เอง จาก API ที่รองรับในปัจจุบันมีบริการที่สามารถสร้างภาระงานพอดและดีพลอยเมนต์ได้ ผู้ใช้งานสามารถบริหารภาระที่ตนสร้างผ่านระบบศูนย์กลางนี้ได้ทุกภาระที่รันอยู่ในแต่ละคลัสเตอร์ที่ร่วมสมาพันธ์ การทดสอบการใช้งานคือ ให้ทดสอบอัพโหลดรีแดชเวิร์คเกอร์ YAML ไฟล์ผ่านระบบ OF@TEIN++ โดยมีการหยุดการทำงานของรีแดชเวิร์คเกอร์บนคลัสเตอร์หลักก่อนดังรูปที่ 72 และทดสอบการสั่งอัพเดทข้อมูลบนเว็บรีแดชซึ่งจะรอประมวลผลจนกว่าจะมีรีแดชเวิร์คเกอร์มารับงานดังรูปที่ 73 จากนั้นอัพโหลด YAML จากในภาคผนวก ไปบน OF@TEIN++ ดังรูปที่ 74



รูปที่ 72 การหยุดการทำงานของรีแดชเวิร์คเกอร์บนคลัสเตอร์หลัก

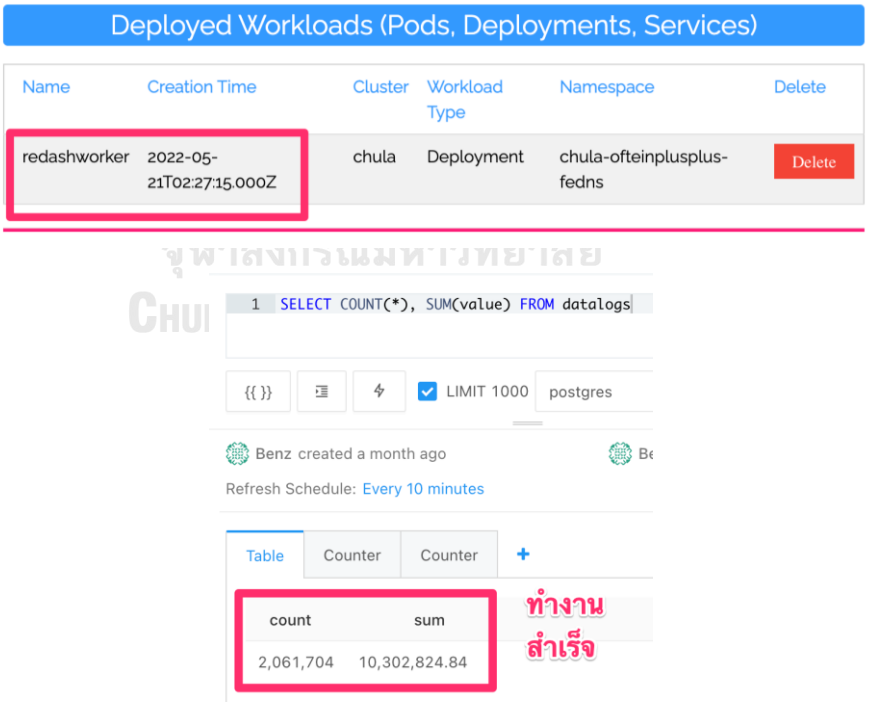


รูปที่ 73 คำสั่งการอัพเดทข้อมูลบนรีแดชที่รอการประมวลผลโดยรีแดชเซิร์ฟเวอร์



รูปที่ 74 ขั้นตอนการอัพโหลด YAML เพื่อรันคอนเทนเนอร์ข้ามคลัสเตอร์บนระบบ OF@TEIN++

ผลการสร้างคอนเทนเนอร์สามารถรันคอนเทนเนอร์ได้สำเร็จ และการดึงข้อมูลบนรีแดชสามารถทำได้สำเร็จด้วยรีแดชเวิร์คเกอร์ผ่านระบบ OF@TEIN++



รูปที่ 75 ผลการรันคอนเทนเนอร์ผ่านระบบ OF@TEIN++ และผลการดึงข้อมูลบนโปรแกรมรีแดช

ภาคผนวก ฆ คิวเบอร์เนเทสบนคลาวด์สาธารณะ

คิวเบอร์เนเทสบนคลาวด์สาธารณะสามารถเปิดคุณสมบัติคิวเบอร์เนเทสบนหลายผู้ให้บริการได้ (multi cloud) โดยติดตั้งบางโนดบนคลาวด์สาธารณะ คิวเบอร์เนเทสบนคลาวด์สาธารณะสามารถติดตั้งบนบริการวีเอ็ม มีการศึกษาผู้ให้บริการสามเจ้าคือ กูเกิล (Google), วาลเตอร์ (Vultr) และอัพคลาวด์ (Upcloud) ผลการศึกษาการติดตั้งบนคลาวด์สาธารณะมีข้อพิจารณาดังนี้

1. ค่าใช้จ่าย การใช้คลาวด์สาธารณะมีทั้งผู้ให้บริการรายใหญ่เช่น กูเกิล (Google), ไมโครซอฟท์, และแอมะซอล (Amazon) โดยจะมีค่าบริการโดยเฉลี่ยแพงกว่าผู้ให้บริการรายย่อย และคิดค่าใช้จ่ายแบนด์วิดนำออกข้อมูลแยกกัน กรณีตัวอย่างการใช้บริการวีเอ็มของกูเกิล ในบริการชื่อกูเกิลคอมพิวเอนจิน (Google compute engine) จะคิดค่าใช้จ่ายแยกกัน [50] เป็นหน่วยประมวลผลกลาง (CPU) คิดตามจำนวนคอร์และเวลาที่ใช้, หน่วยความจำ (Memory) คิดตามขนาดและเวลาที่ใช้, หน่วยเก็บข้อมูล (Disk) คิดตามขนาดตลอดระยะเวลาการนำข้อมูลคงไว้, และค่าใช้จ่ายปริมาณข้อมูลนำออก ในขณะที่ผู้ให้บริการรายย่อยเช่น วาลเตอร์จะมีค่าใช้จ่ายโดยรวมถูกกว่า ทั้ง 2 ผู้ให้บริการ มีการคิดค่าบริการในอัตราคิดเต็มรายชั่วโมง โดยเฉลี่ยจากอัตรารายเดือน เช่นการใช้บริการจำนวน 1 ชั่วโมง 30 นาที จะคิดเป็นราคา 2 ชั่วโมง เปรียบเทียบค่าใช้จ่ายระหว่างกูเกิลคลาวด์และวาลเตอร์ โดยเลือกบริการแบบหน่วยประมวลผลกลางแบบแยกเฉพาะ (dedicated CPU) มีผลสรุปดังนี้

บริการ	Google	ค่าใช้จ่าย [51]	Vultr	ค่าใช้จ่าย [52]
หน่วยประมวลผลกลาง	2 cores	**49.82$**	2 cores	**40$**
หน่วยความจำ	4 GB		4 GB	
หน่วยเก็บข้อมูล	50 GB	**9.35$**	50 GB	
แบนด์วิดข้อมูลนำออก	คิดตั้งแต่หน่วยแรก	0.12$/GB [53]	ฟรี 5TB แรก	0.01$/GB
รวม	ต่อเดือน	**>59.17$**	ต่อเดือน	**40$**

มีข้อสังเกตคือ ค่าบริการต่อเดือนวาลเตอร์มีค่าบริการถูกกว่าประมาณ 30% โดยทั้ง 2 ผู้ให้บริการไม่คิดราคาปริมาณข้อมูลนำเข้า คิดเฉพาะกรณีนำออก แต่วาลเตอร์จะรวมค่าบริการข้อมูลนำออกให้ฟรี 5TB แรก และอัตราค่าบริการข้อมูลนำออกส่วนเกินของวาลเตอร์ มีอัตราถูกกว่าของกูเกิลเกิน 10 เท่า

2. คุณภาพการให้บริการ เป็นปัจจัยสำคัญในการคัดเลือกผู้ให้บริการคลาวด์ โดยบางบริการที่เน้นประหยัดค่าใช้จ่าย อาจเลือกใช้หน่วยประมวลผลกลางแบบแชร์ร่วมกัน (shared CPU) ทำให้มีประสิทธิภาพน้อยกว่าแบบแยกเฉพาะ (dedicated CPU) ขึ้นอยู่กับลักษณะงาน รวมทั้งหน่วยเก็บข้อมูลที่มีทั้งแบบเร็วและแบบธรรมดา และมีประสิทธิภาพการเขียนอ่านเร็วต่างกัน

3. ความหลากหลาย แอมาซอลมีบริการหลากหลายทั้งงานการประมวลผล, ไอโอที, เกม, การเรียนรู้ของเครื่อง, การประมวลผลควอนตัม เป็นต้น ส่วนอัพคลาวด์จะมีบริการเฉพาะงานการประมวลผล และฐานข้อมูล ส่วนวาลเตอร์ จะมีบริการเฉพาะงานการประมวลผลเท่านั้น

4. การรับประกัน โดยมากการรับประกันของผู้ให้บริการคลาวด์ จะไม่รับประกันหรือรับผิดชอบตามมูลค่าความเสียหายของข้อมูลใด ๆ หรืออย่างมากอาจชดใช้ตามจำนวนเครดิตที่ผู้ใช้บริการเสียไป โดยผู้ใช้งานต้องเสนอคำร้องด้วยตัวเอง หรือบางกรณีเช่น วาลเตอร์ ไม่มีการรับประกันใด ๆ ให้บริการแบบตามสภาพ (as-is) [54]

5. พื้นที่ให้บริการ กรณีพื้นที่ให้บริการอยู่ห่างไกลจากผู้ใช้งาน จะส่งผลต่อระยะเวลาตอบสนอง หรืออาจทำให้บางแอพลิเคชันทำงานผิดพลาดได้ เช่นงานประเภทธุรกรรม (transaction) ที่เกี่ยวข้องกับหลายระบบและต้องดำเนินการให้เร็วที่สุด หากมีระบบใดใช้ระยะเวลาตอบสนองนานจะกระทบทั้งระบบ

6. การให้คำปรึกษา ผู้ให้บริการคลาวด์เช่น กูเกิล และ วาลเตอร์ จะไม่มีพนักงานให้คำปรึกษาโดยตรงจำเป็นต้องเขียนคำร้องตามแต่กรณี ในขณะบริการอัพคลาวด์จะมีพนักงานให้คำปรึกษาตลอด 24 ชั่วโมง

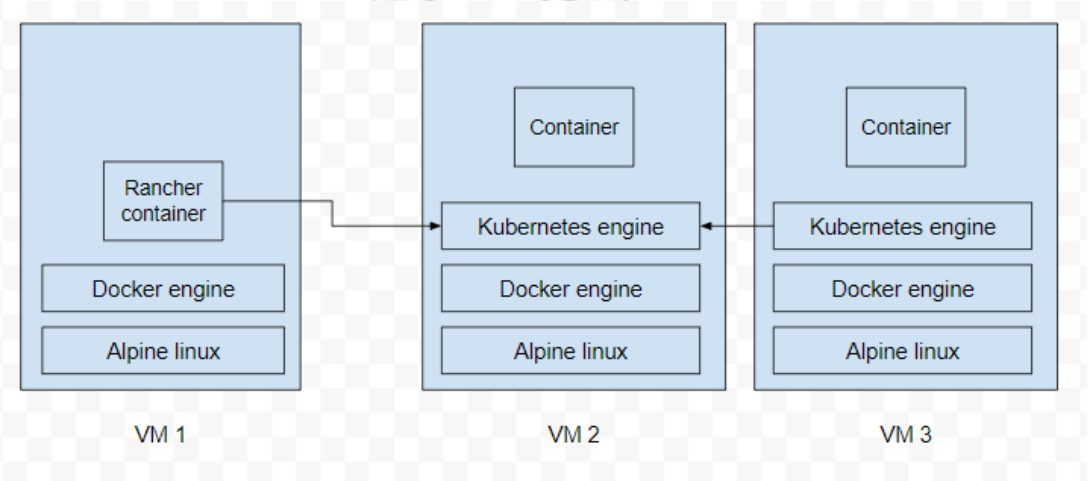
7. บริการเสริม บางผู้ให้บริการมีบริการสำรองข้อมูลฟรี เช่น วาลเตอร์มีบริการสำรองข้อมูลฟรี 20 GB หรืออัพคลาวด์ไม่มีค่าใช้จ่ายการนำออกข้อมูลระหว่างวีเอ็มของเครือข่ายภายในแม้อยู่ต่างทวีป เป็นต้น

ภาคผนวก ง การสร้างระบบทดสอบคิวเบอร์เนเทส

การสร้างระบบทดสอบ (sandbox) เป็นการจำลองสถานการณ์เพื่อประโยชน์ในการสร้างความเข้าใจการทำงานของสถาปัตยกรรมระบบ ตลอดจนการทดสอบแอพลิเคชันในการเปิดคุณสมบัติต่าง ๆ เช่นความพร้อมใช้งานระดับสูง, การขยายขนาด, การยืมทรัพยากร เป็นต้น สำหรับคิวเบอร์เนเทส 1 คลัสเตอร์ ประกอบด้วย เครื่องมาสเตอร์โนด, เครื่องเวิร์คเกอร์โนด, และซอฟ์ตแวร์รานเชอร์ มีรูปแบบดังนี้

รูปแบบอย่างง่าย

กำหนดให้สร้างเครื่องมาสเตอร์โนด, เครื่องเวิร์คเกอร์โนด, และซอฟ์ตแวร์รานเชอร์ บนเวอร์ชัวแมชชีน**แยกกัน** ดังรูปที่ 76



รูปที่ 76 โครงสร้างระบบทดสอบคิวเบอร์เนเทสอย่างง่าย

รูปที่ 76 แสดงโครงสร้างระบบทดสอบคิวเบอร์เนเทสอย่างง่าย โครงสร้างนี้สร้างเวอร์ชัวแมชชีนหรือวีเอ็ม 3 เครื่องแยกกัน ทั้ง 3 เครื่องติดตั้งดอกเกอร์เอนจิน และติดตั้งเคสามเอสเฉพาะเครื่องที่ 2 กับ 3 วีเอ็มเครื่องแรกใช้รันโปรแกรมรันเชอร์ วีเอ็มเครื่องที่ 2 และ 3 ทำงานเป็นมาสเตอร์โนดและเวิร์คเกอร์โนดตามลำดับ โครงสร้างนี้สามารถทดสอบในคอมพิวเตอร์เครื่องเดียวได้ โดยใช้โปรแกรมเวอร์ชัวบอก การตั้งค่าเครือข่ายของโปรแกรมเวอร์ชัวบอกมีรูปแบบที่แตกต่างกันดังนี้

รูปแบบ	วีเอ็ม -> โฮส*	วีเอ็ม <- โฮส	วีเอ็ม <-> วีเอ็ม	วีเอ็ม -> เน็ต	วีเอ็ม <-เน็ต
โฮสเท่านั้น	ได้	ได้	ได้	ไม่ได้	ไม่ได้
ภายใน	ไม่ได้	ไม่ได้	ได้	ไม่ได้	ไม่ได้
บริดจ์ (Bridged)	ได้	ได้	ได้	ได้	ได้
แนท (NAT)	ได้	พอร์ทฟอร์เวิร์ด	ไม่ได้**	ได้	พอร์ทฟอร์เวิร์ด

* หัวลูกศรหมายถึงเครื่องปลายทาง, ** เชื่อมต่อได้เฉพาะกรณีทั้ง 2 วีเอ็มอยู่ในแนทกลุ่มเดียวกัน

เพื่อความง่าย จะเลือกใช้รูปแบบบริดจ์ ที่มีการเชื่อมต่อวีเอ็มเข้ากับเครือข่ายเหมือนการเข้าร่วมเครือข่ายจริงของแต่ละวีเอ็ม โดยปกติทำให้ดีเอชซีพี (DHCP server) จ่ายไอพีแอดเดรซให้วีเอ็มเครื่องใหม่เหล่านี้โดยอัตโนมัติ เครือข่ายแบบบริดจ์จะรองรับทั้งการเข้าถึงอินเตอร์เน็ต และการสื่อสารระหว่างวีเอ็มและวีเอ็มด้วยกัน ทำให้สามารถทดสอบการทำงานระหว่างมาสเตอร์โนดและเวิร์คเกอร์โนดได้ อย่างไรก็ตามหากคอมพิวเตอร์ที่ใช้ทดสอบเปลี่ยนเครือข่าย จะทำให้ไอพีแอดเดรสของวีเอ็มเปลี่ยนบนเครือข่ายใหม่ ทำให้วีเอ็มมาสเตอร์โนดและเวิร์คเกอร์โนดไม่สามารถเชื่อมต่อไอพีแอดเดรสเดิมได้ การแก้ไขให้เลือกใช้เครือข่ายแบบแนท (NAT) เครือข่ายแบบแนทจะสร้างเครือข่ายภายในเครื่องคอมพิวเตอร์ที่ใช้ทดสอบ และมีดีเอชซีพีภายในจ่ายไอพีให้แต่ละวีเอ็มอัตโนมัติ ทำให้เมื่อคอมพิวเตอร์เปลี่ยนเครือข่าย แนทที่จำลองจะใช้งานได้ไม่ทำให้ไอพีเปลี่ยนแปลง รูปแบบแนทมีข้อสังเกตคือ หากต้องการเข้าถึงบริการที่เปิดบนวีเอ็ม ต้องทำพอร์ทฟอร์เวิร์ดบนเครื่องโฮส การติดตั้งคิวเบอร์เนเทสบนแต่ละวีเอ็มสามารถทำได้ทั้งแบบเคสามเอสและเคสามโอเอส

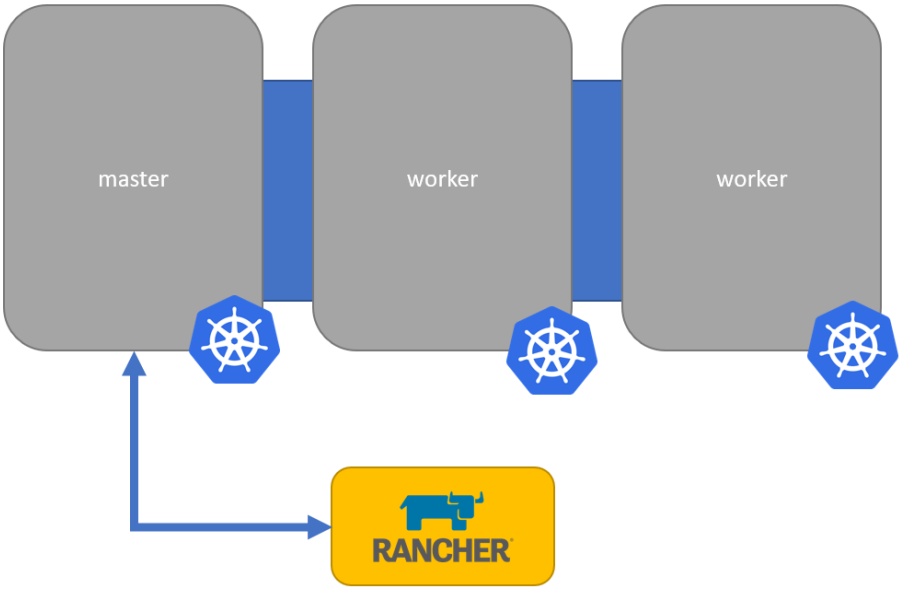
รูปแบบเคสามดี

เคสามดี เป็นการห่อคิวเบอร์เนเทสด้วยคอนเทนเนอร์ ทำให้การรันคอนเทนเนอร์บนคิวเบอร์เนเทสเหมือนการรันคอนเทนเนอร์ในคอนเทนเนอร์ที่สามารถรัน**คิวเบอร์เนเทสหลายคลัสเตอร์**พร้อมกันได้ในเครื่องเดียว เป็นประโยชน์ในการศึกษาการร่วมสมาพันธ์ระหว่างคิวเบอร์เนเทสคลัสเตอร์ แต่ไม่เหมาะกับการนำไปใช้งานจริง (production) เนื่องจากการสงวนทรัพยากรในการรันคอนเทนเนอร์ไม่สามารถควบคุมได้ชัดเจน และการเปิดบริการ (expose service) ต้องเชื่อมโยงพอร์ตระหว่างโฮสไปยังคอนเทนเนอร์ที่เป็นคิวเบอร์เนเทสโนดแต่ละโนดเอง (port forwarding) และต้องกำหนดในระยะก่อนการสร้างคลัสเตอร์ การใช้งานรานเชอร์สามารถใช้งานบนวีเอ็มที่รันเคสามดีได้

รูปแบบรวมศูนย์

รูปแบบรวมศูนย์ถูกพัฒนาขึ้นภายใต้ทรัพยากรที่จำกัด เพื่อศึกษาการใช้งานคิวเบอร์เนเทสและรานเชอร์ในวีเอ็มเครื่องเดียว มีรูปแบบที่เข้าใจง่าย สามารถทำได้โดยติดตั้งดอกเกอร์เอนจิน, ติดตั้งเคสามเอสแบบใช้ดอกเกอร์เอนจิน จากนั้นรันรานเชอร์ผ่านดอกเกอร์เอนจินโดยตรงและต้องกำหนดให้เข้าถึงพอร์ทของรานเชอร์ที่ไม่ทับซ้อนกับคิวเบอร์เนเทส (หมายเลข 443) การติดตั้งเคสามเอส ให้ทำงานเป็นมาสเตอร์โนด ทำให้ในวีเอ็ม 1 เครื่อง ผู้ศึกษาสามารถทดสอบทั้งการใช้งานคอนเทนเนอร์ คิวเบอร์เนเทส และรานเชอร์ได้ในเครื่องเดียว สะดวกต่อการสำรองข้อมูลสถานะอย่างรวดเร็ว (snapshot) ในระหว่างการทดสอบได้

การติดตั้งคิวเบอร์เนเทสคลัสเตอร์ ในโครงการ IoTcloudServe@TEIN

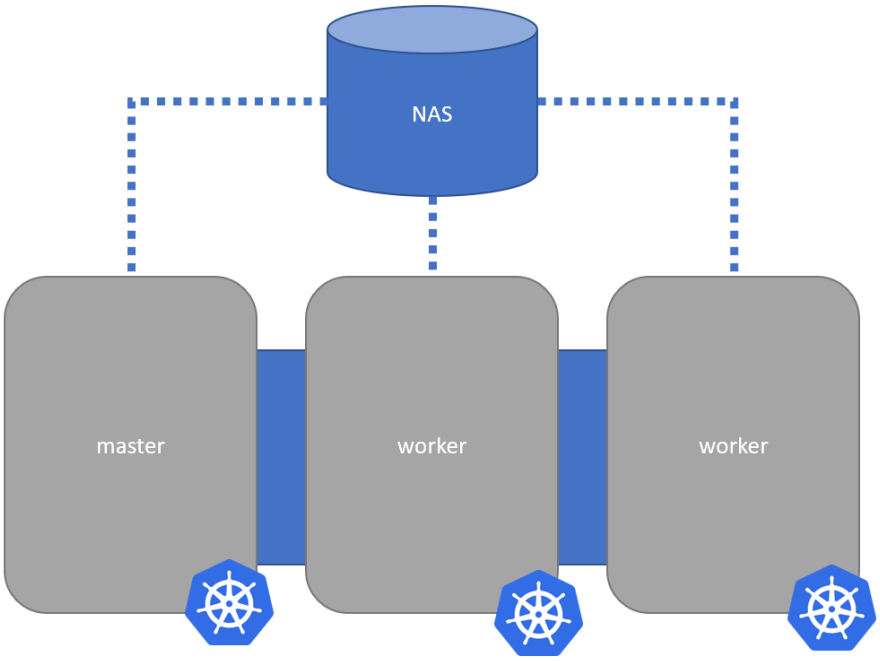


รูปที่ 77 โครงสร้างคิวเบอร์เนเทสคลัสเตอร์ในโครงการ IoTcloudServe@TEIN

ระยะแรก เริ่มจากการติดตั้งคิวเบอร์เนเทสคลัสเตอร์ใน 3 เซิร์ฟเวอร์ให้ทำงานร่วมกัน โดยกำหนดเครื่องหนึ่งเป็นมาสเตอร์โนด ที่เหลือเป็นเวิร์คเกอร์โนด การติดตั้งคิวเบอร์เนเทสมีหลายวิธีที่มีการบำรุงรักษาต่างกัน ได้เลือกใช้วิธีติดตั้งระบบปฏิบัติการ K3OS [48] เนื่องจากมีการติดตั้งคิวเบอร์เนเทสเอนจิน (Kubernetes Engine) อัตโนมัติ และมีเครื่องมือจัดการหน่วยเก็บข้อมูลที่มีประสิทธิภาพ [55] การติดตั้งเริ่มจากนำเข้าไฟล์อิมเมจ K3OS ที่เครื่องมาสเตอร์โนดก่อน มีสิ่งที่ต้องเตรียมการดังนี้

1. โฮสเนม (hostname) ต้องไม่ซ้ำกับโนดใด ๆ ในคลัสเตอร์
2. โทเค็น (token) ที่ใช้ในการสื่อสารระหว่างโนด
3. การเตรียมหน่วยเก็บข้อมูลให้พร้อมใช้
4. การตั้งค่าเครือข่าย

เมื่อติดตั้งเสร็จสมบูรณ์ จึงติดตั้งเครื่องเวิร์คเกอร์อีก 2 เครื่องที่เหลือ โดยกำหนดให้ติดต่อกับเครื่องมาสเตอร์ผ่านไอพีแอดเดรสที่ตั้งค่า ร่วมกับโทเค็นที่กำหนดไว้ล่วงหน้า นอกจากนี้ในคลัสเตอร์ยังมีอุปกรณ์แนส (NAS) ที่เป็นหน่วยเก็บข้อมูลขนาดใหญ่และอยู่ในดาต้าเซ็นเตอร์เดียวกัน จึงตั้งค่าให้ทุกโนดสามารถเข้าถึงแนสได้



รูปที่ 78 โครงสร้างการเชื่อมต่อระหว่างโนดกับแนส

ซอฟต์แวร์รานเชอร์ (rancher) เป็นเครื่องมือสำหรับมอนิเตอร์และสั่งการคิวเบอร์เนเทสคลัสเตอร์อีกที มีการติดตั้งภายในเครื่องมาสเตอร์ เพื่อให้สามารถมอนิเตอร์คลัสเตอร์ได้สะดวก จากนั้นได้ใช้ฟิเจอร์ของรันเชอร์ในการกำหนดผู้ใช้งานในคลัสเตอร์ ให้สามารถสร้างคอนเทนเนอร์และทรัพยากรคิวเบอร์เนเทสอื่น ๆ ได้

การขยายผลสามารถทำได้ด้วยขั้นตอนเดียวกัน จึงประสานงานการติดตั้งไปยังหน่วยงานต่าง ๆ ให้ใช้โทเค็นและกำหนดมาสเตอร์โนดมายังศูนย์กลางเดียวกัน

การจัดการหน่วยเก็บข้อมูลของคิวเบอร์เนเทสในโครงการ IoTcloudServe@TEIN มี 2 แบบ คือ แบบเส้นทางท้องถิ่น (local path) และแบบลองฮอร์น (longhorn) ลองฮอร์นเป็นซอฟต์แวร์พิเศษในการจัดการหน่วยเก็บข้อมูลของทุกโนดแบบรวมศูนย์ โดยกระจายหน่วยควบคุมการเข้าถึงหน่วยเก็บข้อมูลไปแต่ละโนด ลองฮอร์นมีคุณสมบัติพิเศษในการสำรองข้อมูลในหน่วยเก็บข้อมูลได้ แต่มีข้อจำกัดคือ การเข้าใช้ข้อมูลในหน่วยเก็บข้อมูลผ่านลองฮอร์น จะต้องเป็นคอนเทนเนอร์ที่ทำงานอยู่ใน**โนดเดียวกัน**กับโนดที่หน่วยเก็บข้อมูลของลองฮอร์นเชื่อมต่ออยู่ ดังนั้นหากมีหลายคอนเทนเนอร์เข้าใช้งานหน่วยเก็บข้อมูลของลองฮอร์นพร้อมกัน การใช้งานสูงสุดจึงขึ้นอยู่กับทรัพยากรของโนดนั้น ๆ ที่หน่วยเก็บข้อมูลเชื่อมต่ออยู่

การใช้หน่วยเก็บข้อมูลแต่ละประเภท จึงต้องพิจารณาลักษณะการเข้าถึงข้อมูลของระบบดังนี้

1. ระบบงานทั่วไป ไม่จำเป็นต้องใช้การประมวลผลเชิงกระจาย ให้เลือกใช้หน่วยเก็บข้อมูลแบบใดก็ได้
2. ระบบงานทั่วไป ที่ต้องการสำรองข้อมูลแบบสแนปชอต (snapshot) ได้ ให้ใช้ลองฮอร์น
3. ระบบงานที่ต้องการเข้าถึงข้อมูลจากหลายคอนเทนเนอร์ที่กระจายไปหลายโนด ให้เลือกวิธีอื่นที่ไม่ใช่ลองฮอร์น

การตั้งค่าแบบเส้นทางท้องถิ่นที่ใช้ของโนดในโครงการ IoTcloudServe@TEIN มีความพิเศษคือ มีการเชื่อมต่อกับแนสทั้ง 3 โนด ทำให้คอนเทนเนอร์ที่ทำงานอยู่สามารถกระจายไปได้ทั่วทั้ง 3 โนดพร้อมกัน เมื่อต้องการพลังการประมวลผลระดับสูง ซึ่งสูงสุดได้ตามจำนวนทรัพยากรของทั้ง 3 โนดรวมกัน

เมื่อขยายโครงการไปยังเซิร์ฟเวอร์ของมหาวิทยาลัยต่าง ๆ ลองฮอร์นจะขยายขอบเขตและรวบรวมหน่วยเก็บข้อมูลที่ได้เพิ่มขึ้นมารวมกัน อย่างไรก็ตาม การนำหน่วยเก็บข้อมูลไปใช้ จะขึ้นอยู่กับระยะทางระหว่างหน่วยประมวลผลและหน่วยเก็บข้อมูลด้วย หากมีช่องว่างทางเวลามากเกินไป (latency) คอนเทนเนอร์จะไม่สามารถเข้าถึงหน่วยเก็บข้อมูลนั้นได้ ดังนั้นระยะห่างเชิงกายภาพระหว่างโนดมีผลต่อการให้บริการของระบบ แนวทางหนึ่งคือการสร้างคอนเทนเนอร์เข้าถึงข้อมูลแยกกัน หากมีการเรียกใช้ข้อมูลจากท้องถิ่นอื่น ให้เรียกผ่านคอนเทนเนอร์นั้น

การเชื่อมต่อแนสกับเซิร์ฟเวอร์ มีการเชื่อมต่อกันด้วยโปรโตคอลเอ็นเอสเอฟ (NFS - network file system) ไปยังโนดต่าง ๆ แต่การเชื่อมต่อจะหายไปทุกครั้งเมื่อเปิดปิดเครื่องใหม่ ดังนั้นจึงกำหนดสคริปต์ให้แต่ละโนดเมื่อรีบูทเครื่องต้องเชื่อมต่อหน่วยเก็บข้อมูลนี้ใหม่ทุกครั้ง โครงสร้างการรันสคริปต์พิเศษเมื่อรีบูธเครื่องของ K3OS ดังรูปที่ 79

```
- "sudo mount [ip-address]:/volume1/storage1 /nfs/storage1"
- "sudo mount [ip-address]:/volume2/storage2 /nfs/storage2"
- "sudo mount [ip-address]:/volume3/storage3 /nfs/storage3"
```

รูปที่ 79 สคริปต์เชื่อมต่อกับแนส

ความท้าทายในการติดตั้งบนเครือข่ายจำเพาะคือ ต้องมีการตั้งค่าเครือข่ายแบบทำมือทุกครั้งที่มีการรีบูธเครื่อง จึงต้องมีสคริปต์ที่ใช้ตั้งค่าเครือข่าย ดังรูปที่ 80

```
run_cmd:
- "sudo connmanctl config [ชื่อ network interface] --ipv4
  manual [ip-address] [subnetmask] [gateway] --nameservers
  [name-server]"
- "sudo service connman restart"

```

รูปที่ 80 สคริปต์การตั้งค่าเครือข่าย

คำสั่งแรกเป็นการกำหนดไอพีไม่ผันแปร (static ip-address) และคำสั่งที่ 2 เป็นคำสั่งปรับใช้เครือข่ายทันทีกับเครื่อง สามารถสรุปเป็นคำสั่งรวมทั้งหมดได้ดังนี้

```
run_cmd:
- "sudo connmanctl config [ชื่อ network interface] --ipv4
  manual [ip-address] [subnetmask] [gateway] --nameservers
  [name-server]"
- "sudo service connman restart"

- "sudo mount [ip-address]:/volume1/storage1 /nfs/storage1"
- "sudo mount [ip-address]:/volume2/storage2 /nfs/storage2"
- "sudo mount [ip-address]:/volume3/storage3 /nfs/storage3"
```

รูปที่ 81 คำสั่งรวมการตั้งค่าเครือข่ายกับการเชื่อมต่อแนส

จากการทดสอบ ไม่สามารถเชื่อมต่อกับแนสได้ทันทีหลังการตั้งค่าเครือข่ายสำเร็จ จึงกำหนดคำสั่งหน่วงเวลาเล็กน้อย ก่อนรันสคริปต์เชื่อมต่อกับแนส

```
run_cmd:
- "sudo connmanctl config [ชื่อ network interface] --ipv4 manual [ip-address] [subnetmask] [gateway] --nameservers [name-server]"
- "sudo service connman restart"
- "sleep 10"
- "sudo mount [ip-address]:/volume1/storage1 /nfs/storage1"
- "sudo mount [ip-address]:/volume2/storage2 /nfs/storage2"
- "sudo mount [ip-address]:/volume3/storage3 /nfs/storage3"
```

รูปที่ 82 คำสั่งรวมการตั้งค่าเครือข่ายกับการเชื่อมต่อแนสหลังการเพิ่มการหน่วงเวลา

สคริปต์ที่ได้ให้บรรจุรวมกับการตั้งค่าของ K3OS โดย K3OS มีขั้นตอนการดึงข้อมูลการตั้งค่าพิเศษดังนี้

1. /k3os/system/config.yaml ไม่สามารถแก้ไขได้ ถูกสร้างครั้งแรกหลังการติดตั้ง
2. /var/lib/rancher/k3os/config.yaml
3. /var/lib/rancher/k3os/config.d/*

ไฟล์สคริปต์การตั้งค่าในข้อ 2 และ 3 จะถูกเรียกใช้ตามลำดับ ในระยะการเปิดเครื่อง (Booting time)

ภาคผนวก จ คำสั่งที่ใช้ในการทดสอบการดุลภาระ

ติดตั้งคิวเบอร์เนเทสคลัสเตอร์ในวีเอ็มเครื่องเดียวกันดังนี้

```
k3d cluster create c1 -i rancher/k3s:v1.21.1-k3s1 -p "10001-10011:10001-10011@loadbalancer"
k3d cluster create c2 -i rancher/k3s:v1.21.1-k3s1 -p "10011-10021:10011-10021@loadbalancer"
k3d cluster create c3 -i rancher/k3s:v1.21.1-k3s1 -p "10021-10031:10021-10031@loadbalancer"
```

ติดตั้งเว็บเซอร์วิซไปยังทั้งสามคลัสเตอร์ ผ่าน API ของ OF@TEIN++ ดังนี้

IP Address ของ วีเอ็มตัวดุลภาระ	192.168.0.168
โหลดบาลลานซ์ ดอกเกอร์	docker run -d -p 81:80 --name nginx2 nginx:alpine
คำสั่งควบคุม nginx2	docker exec -it nginx2 ash
Service 1	202.28.193.107:10001
Service 2	202.28.193.107:10011
Service 3	202.28.193.107:10021
ไฟล์กำหนดการดุลภาระ	vi /etc/nginx/conf.d/default.conf

ไฟล์เดิม	ไฟล์ใหม่

server {
...
location / {
root
/usr/share/nginx/html;
index index.html
index.htm;
}
...
}
	upstream smartenergy {
	server 202.28.193.107:10001;
	server 202.28.193.107:10011;
	server 202.28.193.107:10021;
	}
	server { ...
	location / {
	#root /usr/share/nginx/html;
	#index index.html index.htm;
	proxy_pass http://smartenergy;
	} ...
	}

Apply ใช้งาน	nginx -s reload

ภาคผนวก ฉ คำสั่งที่ใช้รันคอนเทนเนอร์เพื่อทดสอบความพร้อมใช้ระดับสูงของคลัสเตอร์

kubectl config use-context [ชื่อคลัสเตอร์ปลายทาง]
kubectl create deployment nginx --image=nginx:alpine kubectl create service clusterip nginx --tcp=80:80 kubectl apply -f nginxservice.yaml // กรณียังไม่มีไฟล์ nginxservice ให้สร้างดังนี้ vi nginxservice.yaml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
  annotations:
  ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
  paths:
  - path: /
  pathType: Prefix
  backend:
  service:
  name: nginx
  port:
  number: 80

kubectl apply -f nginxservice.yaml

ภาคผนวก ช โค้ดโปรแกรมวิเคราะห์ข้อมูล

โค้ดอ่านไฟล์ CSV PIR

```
parse = require('csv-parse')
parser = parse({delimiter: ',', columns: true})                    // สร้าง Parser อ่านไฟล์ CSV ด้วยการคั่นด้วย ','
const streamPIR = fs.createReadStream(' /usr/src/app/csv/' + request.params.pir)
                                                                   // เปิด Stream ในการอ่านไฟล์ CSV
let pirslot = {}                          // สร้างตัวแปร pirslot กำหนดช่วงเวลา PIR ตรวจจับความเคลื่อนไหว
await new Promise((resolve, reject) => {                           // สร้าง Promise เพื่อทำ Async
    parser.on('readable', () => {                                  // ทุกระหว่างที่ Parser สามารถจับข้อมูลได้
        while (record = parser.read()) {                           // ใส่ข้อมูลการอ่านลงในตัวแปร record
            let time = new Date(record.TIME)                       // parse ข้อมูลเวลา
            time.setSeconds(0)                                     // ปรับให้เป็นวินาที 0 เพื่อให้เทียบกับ PIR ได้
            time = time.getTime()                                  // แปลงเวลาเป็นตัวเลขหน่วย ms
            switch (record.VALUE) {                                // อ่านข้อมูล PIR
                case '1.0':                                        // กรณีข้อมูลเป็น 1 คือตรวจพบความเคลื่อนไหว
                    pirslot[time] = true;
                    break;
                case 'ON':                                         // กรณีข้อมูลเป็น ON คือตรวจพบความเคลื่อนไหว
                    pirslot[time] = true;
                    break;
                default:                                           // กรณีข้อมูลเป็นอย่างอื่น คือไม่ตรวจพบความเคลื่อนไหว
                    pirslot[time] = false;
                    break;
            } } })
    parser.on('error', (error) => reject(error.message))           // เมื่อเกิด error ให้คืนข้อความบ่งบอก error
    parser.on('finish', () => {       console.log("read pir done")
        resolve()   })
    streamPIR.pipe(parser)                                         // Stream ข้อมูล PIR ลงใน parser
})
```

โค้ดอ่านไฟล์ CSV จากเครื่องใช้ไฟฟ้า

```
const streamEnergy = fs.createReadStream(__dirname + '/../csv/' +
request.params.energy)

let energystat = {
    allenergy: 0,
    wastedenergy: 0,
    usefulenergy: 0,
    insight: {
        wastedenergy: {},
        usefulenergy: {}
    }
}

parser = parse({
    delimiter: ',',
    columns: true
})

await new Promise((resolve, reject) => {
    parser.on('readable', () => {
        while (record = parser.read()) {
            let time = new Date(record.TIME)
            let insightTime = new Date(record.TIME)

            time.setSeconds(0);
            time = time.getTime()

            insightTime = insightTime.getFullYear() + "-" + (insightTime.getMonth() + 1);
```

```
if (!(insightTime in energystat.insight.wastedenergy)) {
    energystat.insight.wastedenergy[insightTime] = 0;
    energystat.insight.usefulenergy[insightTime] = 0;
}

if (time in pirslot) {

    let energyusage = parseFloat(record.VALUE)
    if (energyusage < 0) energyusage = 0;
    energyusage = energyusage / 60;
    let HASPERSON = pirslot[time];

    if (!HASPERSON) {
        if (pirslot[time - 1000 * 60 * 1]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 2]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 3]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 4]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 5]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 6]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 7]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 8]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 9]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 10]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 11]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 12]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 13]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 14]) HASPERSON = true;
        else if (pirslot[time - 1000 * 60 * 15]) HASPERSON = true;
    }

    energystat.allenergy += energyusage
```

```
                if (HASPERSON) {
                    energystat.usefulenergy += energyusage;
                    energystat.insight.usefulenergy[insightTime] += energyusage;
                } else {
                    if (energyusage > 15) {
                        energystat.wastedenergy += energyusage;
                        energystat.insight.wastedenergy[insightTime] += energyusage;
                    }
                }
            }
        }
    })
    parser.on('error', (error) => {
        console.log(error)
        reject(error.message)
    })
    parser.on('finish', () => {
        console.log("read energy done")
        resolve()
    })
    streamEnergy.pipe(parser)
})

return energystat;
```

โค้ดดอกเกอร์ไฟล์ในการทำคอนเทนเนอร์ไรเซชัน

```
FROM node:12-slim
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
EXPOSE 8080
COPY . .
CMD ["node", "index.js"]
```

โค้ดสรุปงานระหว่าง PIR และเครื่องใช้ไฟฟ้า และการสั่งงานผ่านอินเกรช

```
let const listfile = [
["0", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "aircon_3ph1", "monitor", "energy_r"],
    ["1", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "aircon_3ph1", "monitor", "energy_s"],
    ["2", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "aircon_3ph1", "monitor", "energy_t"],
    ["3", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "light1", "monitor", "energy"],
    ["4", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "outlet1", "monitor", "energy"],
    ["5", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "sensor1", "monitor", "humidity"],
    ["6", "eng4", "fl13", "north", "lab_tsrl_dsprl_emrl", "z1", "sensor1", "monitor", "illuminance"],
    ...
]
let zonepirlist = []
let energylist = []
let zoneenergy = {}

for (const each of listfile) {
    if (each[each.length - 1] == "pir") {
        const zone = each[1]+"-"+each[2]+"-"+each[3]+"-"+each[4]+"-"+each[5];
        zonepirlist.push(zone)
        zoneenergy[zone] = {
            pir: each[0],
```

```
            energy: []
        }
    }
}

for (const each of listfile) {
    if (each[each.length - 1] == "energy") {
        const zone = each[1]+"-"+each[2]+"-"+each[3]+"-"+each[4]+"-"+each[5];
        if (zonepirlist.includes(zone)) {
            energylist.push(each)
            if (each[each.length - 1] == "energy")
            // if (each[each.length - 1] == "energy" && each[6].indexOf("aircon") != -1)
                zoneenergy[zone]["energy"].push(each[0])
        }
    }
}

let jobs = []

for (const eachzone in zoneenergy) {
    for (const pointid of zoneenergy[eachzone]["energy"]) {
        const job = {
            pirpointid: zoneenergy[eachzone]["pir"],
            energypointid: pointid
        }
        jobs.push(job)
    }
}

const Wreck = require('@hapi/wreck');
```

```
let path = 'http://newwebserver.parallelcomputingdemo.161.200.90.110.xip.io';
// let path = 'http://localhost:8080'

let main = async () => {

    let laps = []
    const parallel = 25
    const maxlap = 4
    console.log(jobs.length);
    while (jobs.length > 0) laps.push(jobs.splice(0, parallel))
    laps = laps.splice(0, maxlap)

    let result = []

    const executejob = (pir, energy) => {
        return Wreck
            .get(path + `/csv/${pir}.csv/${energy}.csv`, { json: true })
            .then((res) => res.payload)
            .then((res) => {
                return {
                    "allenergy": parseFloat(res.allenergy),
                    "wastedenergy": parseFloat(res.wastedenergy),
                    "usefulenergy": parseFloat(res.usefulenergy)
                }
            })
            .catch((e) => console.log("Error: " + e.message))
    }

    n = 0;

    while (laps.length > 0) {
```

```
        n++
        lap = laps.shift()
        lapresult = await Promise.all(lap.map((job) => executejob(job.pirpointid,
job.energypointid)))
        result = result.concat(lapresult);
        console.log("Lap "+n+ ": done");
    }

    let summarize = {
        allenergy: 0,
        wastedenergy: 0,
        usefulenergy: 0
    }

    result.map((res) => {
        summarize.allenergy += res.allenergy
        summarize.wastedenergy += res.wastedenergy
        summarize.usefulenergy += res.usefulenergy
    })

    console.log(summarize);
}

main();
```

ไฟล์ดีพลอยเมนต์รีแดชเวิร์คเกอร์

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redashworker
  labels:
    app: redashworker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redashworker
  template:
    metadata:
      labels:
        app: redashworker
    spec:
      containers:
      - name: redashworker
        image: redash/redash:10.1.0.b50633
        args: ["worker"]
        env:
        - name: GOOGLE_CLIENT_ID
          value: "..."
        - name: PYTHONUNBUFFERED
          value: "0"
        - name: QUEUES
          value: "queries"
        - name: REDASH_COOKIE_SECRET
          value: "secret"
```

```
        - name: REDASH_DATABASE_URL
          value:
"postgresql://postgres:iotcloudserve@tein@postgres@202.28.193.100:5432/redash"
        - name: REDASH_ENFORCE_CSRF
          value: "true"
        - name: REDASH_ENFORCE_PRIVATE_IP_BLOCK
          value: "false"
        - name: REDASH_LOG_LEVEL
          value: "INFO"
        - name: REDASH_MAIL_DEFAULT_SENDER
          value: "redash@example.com"
        - name: REDASH_MAIL_SERVER
          value: "email"
        - name: REDASH_RATELIMIT_ENABLED
          value: "false"
        - name: REDASH_REDIS_URL
          value: "redis://202.28.193.100:63790/0"
        - name: REDASH_SECRET_KEY
          value: "secret"
        - name: WORKERS_COUNT
          value: "2"
```

บรรณานุกรม

1. Marinakis, V., Doukas, H., Tsapelas, J., Mouzakitis, S., Sicilia, Á., Madrazo, L., and Sgouridis, S., *From big data to smart energy services: An application for intelligent energy management.* Future Generation Computer Systems, 2020. **110**: p. 572-586.
2. Tiainen, P., *New opportunities in electrical engineering as a result of the emergence of the Internet of Things*. 2016, Aalto University.
3. *Microsoft 365 and Office Resources*. [cited 2022 May 30]; Available from: https://www.microsoft.com/en-us/microsoft-365/microsoft-365-and-office-resources.
4. *Requirements Overview*. [cited 2022 May 30]; Available from: https://www.enterprisedb.com/edb-docs/d/postgresql/installation-getting-started/installation-guide-installers/12/requirements_overview.html.
5. Boisvert, M., Bigelow, S. J., and Chai, W. *What is iaas? infrastructure as a service definition*. 2020 [cited 2022 May 30]; Available from: https://www.techtarget.com/searchcloudcomputing/definition/Infrastructure-as-a-Service-IaaS.
6. *The Top 5 Enterprise Type 1 Hypervisors You Must Know*. [cited 2022 May 30]; Available from: https://www.actualtechmedia.com/io/top-5-enterprise-type-1-hypervisors/.
7. Potdar, A., D G, N., Kengond, S., and Mulla, M., *Performance Evaluation of Docker Container and Virtual Machine.* Procedia Computer Science, 2020. **171**.
8. *"What is container orchestration?," Red Hat - We make open source technologies for the enterprise*. 2021 [cited 2022 May 30]; Available from: https://www.redhat.com/en/topics/containers/what-is-container-orchestration.
9. Pan, Y., Chen, I., Brasileiro, F., Jayaputera, G., and Sinnott, R., *A Performance Comparison of Cloud-Based Container Orchestration Tools*, in *2019 IEEE International Conference On Big Knowledge (ICBK)*. 2019.

10. Platform9. *Kubernetes Federation: What it is and how to set it up*. 2021 [cited 2022 May 30]; Available from: https://platform9.com/blog/kubernetes-federation-what-it-is-and-how-to-set-it-up/.
11. Sayfan, G., *Mastering Kubernetes: level up your container orchestration skills with Kubernetes to build, run, secure, and observe large-scale distributed apps*. 2020: Packt Publishing.
12. Davis, A., *Data wrangling with JavaScript*. 2019: Manning Publications.
13. *From divide and conquer algorithm to MapReduce - Programmer Sought. Programmersought.com*. 2021 [cited 2022 May 30]; Available from: https://www.programmersought.com/article/9153365366/.
14. TU, C., HE, X., SHUAI Z. and JIANG, F., *Big data issues in smart grid – A review.* Renewable and Sustainable Energy Reviews, 2017. **79**: p. 1558.
15. SYED, D., ZAINAB, A., GHRAYEB, A., S. REFAAT, S. ABU-RUB, H. and BOUHALI, O., *Smart Grid Big Data Analytics: Survey of Technologies, Techniques, and Applications.* IEEE Access, 2021. **9**: p. 59564-59585.
16. DAKI, H., EL HANNANI, A., AQQAL, A., HAIDINE, A. and DAHBI, A., *Big Data management in smart grid: concepts, requirements and implementation. Journal of Big Data.* Journal of Big Data, 2017. **4**.
17. *Install Drill Introduction*. [cited 2022 May 30]; Available from: https://drill.apache.org/docs/install-drill-introduction/.
18. *Setting up a Storm Cluster*. [cited 2022 May 30]; Available from: https://storm.apache.org/releases/2.1.0/Setting-up-a-Storm-cluster.html.
19. PIPATTANASOMPORN, M., CHITALIA, G., SONGSIRI, J., ASWAKUL, C., PORA, W., SUWANKAWIN, S., AUDOMWONGSEREE, K. and HOONCHAROEN, N., *CU-BEMS, smart building electricity consumption and indoor environmental sensor datasets.* Scientific Data, 2020. **7**(1): p. 1-14.
20. Kim, D., Muhammad, H., Kim, E., Helal, S., and Lee, C., *TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform.* Applied Sciences, 2019. **9**(1): p. 191.

21. *OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC*. 2021 [cited 2022 May 30]; Available from: https://www.oasis-open.org/committees/tosca.
22. Larsson, L., Gustafsson, H., Klein, C., and Elmroth, E. *Decentralized Kubernetes Federation Control Plane*. in *2020 IEEE/ACM 13Th International Conference On Utility And Cloud Computing (UCC)*. 2020.
23. *GitHub - kubernetes-sigs/kubefed: Kubernetes Cluster Federation*. 2021 [cited 2022 May 30]; Available from: https://github.com/kubernetes-sigs/kubefed.
24. Jimenez, L.L., and Schelen, O. *DOCMA: A Decentralized Orchestrator for Containerized Microservice Applications*. in *2019 IEEE Cloud Summit*. 2019.
25. *Docker Hub*. 2021 [cited 2022 May 30]; Available from: https://hub.docker.com/search?type=image&category=base.
26. *Why Rancher?* [cited 2022 May 30]; Available from: https://rancher.com/why-rancher.
27. THIRASUPA, R., HOMCHAN, M., KWANKAJORNKEAT, S. and ASWAKUL, C. *Development of IoTcloudServe@TEIN Smart-Energy@Chula Service Gateway : Case Study of Secured On-Demand Building Energy Management System Data Platform Using NETPIE*. in *2019 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. 2019.
28. *Don't Block the Event Loop (or the Worker Pool) | Node.js*. 2021 [cited 2022 May 30]; Available from: https://nodejs.org/en/docs/guides/dont-block-the-event-loop/.
29. CHOOPUTTIPONG, K.a.A., C. *Development of data analytic program for building energy management system with wasted energy analysis using motion sensor*. in *39th Electric. Eng. Conf*. 2016.
30. *Performing rolling updates*. 2022 [cited 2022 May 30]; Available from: https://cloud.google.com/kubernetes-engine/docs/how-to/updating-apps.
31. rancher. *High Availability with Embedded DB*. [cited 2022 May 30]; Available from: https://rancher.com/docs/k3s/latest/en/installation/ha-embedded/.
32. rancher. *High Availability with an External DB*. [cited 2022 May 30]; Available from: https://rancher.com/docs/k3s/latest/en/installation/ha/.

33. *What is etcd?* 2022 [cited 2022 May 30]; Available from: https://etcd.io/.
34. *Creating Highly Available Clusters with kubeadm*. 2022 [cited 2022 May 30]; Available from: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/.
35. *What is failure tolerance?* 2021 [cited 2022 May 30]; Available from: https://etcd.io/docs/v3.3/faq/.
36. *Raft library*. [cited 2022 May 30]; Available from: https://github.com/etcd-io/etcd/tree/main/raft.
37. *Raspberry Pi Zero W*. [cited 2022 May 30]; Available from: https://www.raspberrypi.com/products/raspberry-pi-zero-w/.
38. Labs, R. *rancher/k3s Tags | Docker Hub*. [cited 2022 May 30]; Available from: https://hub.docker.com/r/rancher/k3s/tags.
39. *Raspberry Pi Zero 2 W*. [cited 2022 May 30]; Available from: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/.
40. *Raspberry Pi 4 Tech Specs*. [cited 2022 May 30]; Available from: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.
41. *Raspberry Pi Documentation*. [cited 2022 May 30]; Available from: https://www.raspberrypi.com/documentation/computers/getting-started.html.
42. Saengkaenpetch, K. *OF@TEIN++ Multi-Tenant-Portal Backend App*. 2021 [cited 2022 May 30]; Available from: https://github.com/OFTEIN-NET/OFTEIN-MultiTenantPortal/tree/main/backend-app.
43. Muhammad, U. *OF@TEIN++ Multi-Tenant-Portal Frontend App*. 2021 [cited 2022 May 30]; Available from: https://github.com/OFTEIN-NET/OFTEIN-MultiTenantPortal/tree/main/frontend-app.
44. *Releases k3s-io/k3s GitHub*. [cited 2022 May 30]; Available from: https://github.com/k3s-io/k3s/releases.
45. *Rancher Docs: Advanced Options and Configuration*. [cited 2022 May 30]; Available from: https://rancher.com/docs/k3s/latest/en/advanced/.
46. Firshman, B. *containerd Client CLI*. 2016 [cited 2022 May 30]; Available from: https://github.com/projectatomic/containerd/blob/master/docs/cli.md.

47. *Network ports*. 2022 [cited 2022 May 30]; Available from: https://developers.cloudflare.com/fundamentals/get-started/network-ports/.
48. *Releases · rancher/k3os. GitHub*. 2021 [cited 2022 May 30]; Available from: https://github.com/rancher/k3os/releases.
49. *Installation Options*. 2022 [cited 2022 May 30]; Available from: https://rancher.com/docs/k3s/latest/en/installation/install-options/.
50. *VM instance pricing*. 2022 [cited 2022 May 30]; Available from: https://cloud.google.com/compute/vm-instance-pricing.
51. *Google Cloud Pricing Calculator*. [cited 2022 May 30]; Available from: https://cloud.google.com/products/calculator.
52. *Vultr Pricing*. 2022 [cited 2022 May 30]; Available from: https://www.vultr.com/pricing/.
53. *All networking pricing*. 2022 [cited 2022 May 30]; Available from: https://cloud.google.com/vpc/network-pricing.
54. *Terms of Service*. [cited 2022 May 30]; Available from: https://www.vultr.com/legal/tos/.
55. *Longhorn. Longhorn*. 2021 [cited 2022 May 30]; Available from: https://longhorn.io/.

ประวัติผู้เขียน

ชื่อ-สกุล	กิตติพัฒน์ แสงแก่นเพ็ชร์
วัน เดือน ปี เกิด	11 ธันวาคม 2535
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษา	สำเร็จการศึกษาปริญญาตรี สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ที่อยู่ปัจจุบัน	69/423 หมู่บ้านเพอร์เฟคเพลส ซอยรามคำแหง 164 ถนนรามคำแหง แขวง/เขตมีนบุรี กรุงเทพฯ 10510
ผลงานตีพิมพ์	Kittipat Saengkaenpetch and Chaodit Aswakul. 2021. Cloud-Based Smart Energy Framework for Accelerated Data Analytics with Parallel Computing of Orchestrated Containers: Study Case of CU-BEMS. In 2021 3rd International Conference on Advanced Information Science and System (AISS 2021), November 26-28, 2021, Sanya, China. ACM, New York, NY, USA, 6 Pages. https://doi.org/10.1145/3503047.3503088