1. การจัดการระบบ

การพัฒนาและจัดการแอปพลิเคชันในยุคปัจจุบันได้มีการเปลี่ยนแปลงไปจากวิธีการเดิมๆ ด้วยการนำเทคโนโลยีใหม่ๆ มาใช้ ซึ่งทำให้กระบวนการพัฒนาและการดูแลระบบมีประสิทธิภาพมากขึ้น โดยเฉพาะการใช้ Container, Docker, Microservice, และ API Gateway ซึ่งแต่ละเทคโนโลยีก็มีบทบาทที่สำคัญในการสร้างแอปพลิเคชันที่สามารถปรับตัวได้ง่ายและสามารถขยายได้อย่างยืดหยุ่น

1.1. Container คืออะไร?

Container คือ เทคโนโลยีที่ทำให้การรันแอปพลิเคชันเป็นไปอย่างมีประสิทธิภาพโดยการแยกแอปพลิเคชันและไลบรารีที่จำเป็นออกจากระบบปฏิบัติการหลัก (Host OS) ไว้ในสภาพแวดล้อมที่เป็นอิสระ Container จะบรรจุโค้ด ไลบรารี และการตั้งค่าทั้งหมดที่จำเป็นต้องใช้สำหรับการทำงาน ทำให้แอปพลิเคชันนั้นสามารถทำงานได้ไม่ว่าจะรันในสภาพแวดล้อมไหนก็ตาม เช่น บนเครื่องพัฒนาของโปรแกรมเมอร์ หรือบนเซิร์ฟเวอร์การผลิต

การใช้ Container ช่วยให้

- ☐ **ลดความขัดแย้ง** ในการพัฒนาระหว่างทีม เพราะทีมพัฒนาไม่ต้องกังวลว่าแอปพลิเคชันจะทำงานไม่ได้บนเครื่องอื่น
- ☐ **ปรับขนาดได้ง่าย** ด้วยการโคลน Container เพิ่มเมื่อมีความต้องการใช้งานสูง
- ☐ **การจัดการทรัพยากรมีประสิทธิภาพขึ้น** เนื่องจาก Container ใช้ทรัพยากรของระบบปฏิบัติการร่วมกัน ลดความซ้ำซ้อนของระบบที่ต้องจัดเตรียมทรัพยากรแยกเหมือนใน Virtual Machine

1.2. เปรียบเทียบระหว่างเทคโนโลยี Container และ Virtual Machine

การเปรียบเทียบระหว่าง Container และ Virtual Machine (VM) เป็นหัวข้อสำคัญในการเลือกใช้สถาปัตยกรรมสำหรับการรันแอปพลิเคชัน เนื่องจากทั้งสองเทคโนโลยีนี้มีวิธีการทำงานและประโยชน์ที่แตกต่างกัน การทำความเข้าใจความแตกต่างระหว่าง Container และ VM จะช่วยให้ผู้พัฒนาและผู้ดูแลระบบสามารถเลือกใช้เทคโนโลยีที่เหมาะสมกับงานมากที่สุด

1.3. ภาพรวมของ Container และ Virtual Machine

Virtual Machine (VM) คือ การจำลองระบบคอมพิวเตอร์ทั้งหมดที่รันอยู่บนฮาร์ดแวร์เสมือน โดย VM จะมีระบบปฏิบัติการที่แยกต่างหากจากระบบปฏิบัติการหลัก (Host OS) และทำงานได้อย่างอิสระในลักษณะของคอมพิวเตอร์เสมือน Container คือ เทคโนโลยีที่แบ่งแยกแอปพลิเคชันและไลบรารีที่จำเป็นออกจากระบบปฏิบัติการหลักในรูปแบบที่เบากว่า VM โดย Container ใช้ระบบปฏิบัติการร่วมกับ Host OS และมีการแยกแอปพลิเคชันจากกันในระดับของโปรเซส

ต่อไปนี้คือการเปรียบเทียบความแตกต่างระหว่าง Container และ VM โดยพิจารณาในหลายๆ ด้าน

1. **สถาปัตยกรรม**

- **VM**: ทำงานบน Hypervisor (เช่น VMware, Hyper-V หรือ KVM) ซึ่งจัดการการสร้างและการรัน VM แต่ละตัว โดย VM แต่ละตัวจะมี Guest OS เป็นระบบปฏิบัติการที่รันภายในของตัวเอง พร้อมด้วย Kernel, แอปพลิเคชัน, และการตั้งค่าเฉพาะ ซึ่งหมายความว่า VM มีการแยกทรัพยากรอย่างสมบูรณ์
- **Container**: ทำงานบน Container Engine (เช่น Docker, Kubernetes) โดยไม่ต้องใช้ Hypervisor และใช้ Host OS ร่วมกันทุก Container แต่มีการแยกแอปพลิเคชันและไลบรารีในระดับโปรเซส ทำให้ Container มีขนาดเล็กกว่าและมีการทำงานที่เบากว่า VM

2. **ประสิทธิภาพ**

- **VM**: เนื่องจาก VM มี Guest OS ที่เป็นระบบปฏิบัติการเต็มรูปแบบและต้องใช้ Hypervisor ในการรัน VM การใช้ทรัพยากร เช่น CPU และหน่วยความจำจึงมีค่าใช้จ่ายสูงและทำให้ประสิทธิภาพการทำงานช้าลง โดยเฉพาะเมื่อต้องรันหลาย VM พร้อมกัน
- **Container**: Container ใช้ทรัพยากรร่วมกับ Host OS และมีขนาดเล็กกว่า VM จึงใช้หน่วยความจำและ CPU น้อยลง ซึ่งส่งผลให้ Container มีการทำงานที่รวดเร็วและมีประสิทธิภาพสูงกว่า VM

3. **การใช้งานทรัพยากร (Resource Efficiency)**

- **VM**: แต่ละ VM ต้องจัดสรรทรัพยากรแยกต่างหาก เช่น CPU, RAM และ Disk โดยการรัน VM หลายๆ ตัวพร้อมกันอาจส่งผลให้เกิดการใช้ทรัพยากรสูง ซึ่งไม่สามารถใช้งานทรัพยากรได้อย่างมีประสิทธิภาพเต็มที่เสมอไป
- **Container**: Container ใช้ทรัพยากรร่วมกับ Host OS ทำให้ประหยัดพื้นที่และหน่วยความจำมากกว่า การสร้างและรัน Container หลายๆ ตัวในเวลาเดียวกันยังใช้ทรัพยากรได้น้อยกว่า และสามารถปรับขนาดการใช้งานได้ง่าย

4. **การเริ่มต้นและการหยุดการทำงาน (Startup and Shutdown Time)**

- **VM**: เนื่องจากแต่ละ VM มี Guest OS เต็มรูปแบบ การเริ่มต้นและหยุด VM จะใช้เวลาค่อนข้างนาน ต้องผ่านกระบวนการบูทและการโหลด OS ซึ่งจะกินเวลาและทำให้การตอบสนองช้าลง
- **Container**: Container ใช้เวลาในการเริ่มต้นและหยุดทำงานเร็วกว่ามาก เนื่องจาก Container ไม่ต้องบูท OS ใหม่ แต่เป็นการเรียกโปรเซสในระบบปฏิบัติการหลักโดยตรง ทำให้เหมาะสมกับแอปพลิเคชันที่ต้องการความรวดเร็วและการตอบสนองทันที

5. **ขนาดของ Image และการจัดเก็บ**

- **VM**: VM มีขนาดใหญ่เนื่องจากมีการเก็บข้อมูลของ Guest OS, Kernel, และข้อมูลการติดตั้งต่างๆ ซึ่งส่งผลให้การย้ายหรือแชร์ VM Image ทำได้ยากและใช้เวลาในการส่งถ่ายข้อมูลมากขึ้น
- **Container**: Container Image มีขนาดเล็กกว่ามากเพราะไม่ต้องเก็บ OS ทั้งระบบ แต่จะบรรจุเฉพาะแอปพลิเคชันและไลบรารีที่จำเป็น การแชร์หรือย้าย Container Image จึงทำได้รวดเร็วและง่ายขึ้น

6. **ความสามารถในการพกพา (Portability)**

- ☐ **VM**: VM สามารถรันบนระบบที่รองรับ Hypervisor ใดๆ ที่สามารถโฮสต์ VM ได้ แต่ยังมีข้อจำกัดเนื่องจากต้องพิจารณาเวอร์ชันของ Hypervisor และการตั้งค่าที่ใช้
- ☐ **Container**: Container มีความสามารถในการพกพาสูงมาก เนื่องจากรันบน Container Engine ที่รองรับทุกแพลตฟอร์ม เช่น Docker ทำให้แอปพลิเคชันสามารถย้ายไปยังสภาพแวดล้อมที่ต่างกันได้ง่ายและทำงานได้เหมือนกันทุกที่

7. **การแยกความปลอดภัย (Isolation and Security)**

- ☐ **VM**: VM มีการแยกความปลอดภัยที่ดีกว่า เนื่องจาก Guest OS แต่ละตัวมี Kernel เป็นของตนเอง และทำงานในลักษณะอิสระ การโจมตีจาก VM หนึ่งไปยัง VM อื่นหรือ Host OS จึงทำได้ยากกว่า
- ☐ **Container**: Container มีการแยกความปลอดภัยในระดับโปรเซส ซึ่งไม่สมบูรณ์เท่า VM เนื่องจากทุก Container ใช้ Host OS ร่วมกัน ทำให้หากมีการเจาะระบบหรือโจมตีในระดับ Kernel ของ Host OS อาจส่งผลต่อ Container ทั้งหมดได้

1.4. สรุปเปรียบเทียบ

คุณลักษณะ	Virtual Machine (VM)	Container
การจำลองระบบ	จำลองระบบปฏิบัติการทั้งหมด	ใช้เคอร์เนลร่วมกับโฮสต์
ระบบปฏิบัติการ	แต่ละ VM มีระบบปฏิบัติการของตัวเอง	ใช้ระบบปฏิบัติการเดียวกับโฮสต์
ประสิทธิภาพ	ใช้ทรัพยากรมากเพราะต้องรัน OS ทั้งระบบ	ใช้ทรัพยากรน้อยเพราะแชร์ OS กับโฮสต์
ขนาดไฟล์	ขนาดใหญ่ (รวม OS และแอปพลิเคชัน)	ขนาดเล็ก (มีเพียงแอปพลิเคชันและไลบรารี)
เวลาในการเริ่มต้น	ช้ากว่า เนื่องจากต้องบูต OS	เร็วกว่า เพราะรันแค่กระบวนการของแอป
การแยกการทำงาน	แยกการทำงานได้เต็มรูปแบบ	แยกผ่าน namespace และ cgroups
การพกพา (Portability)	ขึ้นอยู่กับ Hypervisor และ OS	ง่ายต่อการพกพาผ่าน Docker หรือ Podman
การใช้งาน	เหมาะสำหรับแอปพลิเคชันขนาดใหญ่หรือหลาย OS	เหมาะสำหรับแอปพลิเคชันที่ปรับขยายได้
การจัดการ	ซับซ้อนกว่า เพราะต้องจัดการทั้ง VM และ OS	ง่ายกว่า โดยใช้เครื่องมือจัดการ Container
ทรัพยากรที่ใช้	ใช้ CPU, RAM และ Storage มากกว่า	ใช้น้อยกว่าเพราะไม่มี OS เต็มรูปแบบ
ตัวอย่างเครื่องมือ	VMware, Hyper-V, VirtualBox	Docker, Kubernetes, Podman

1.5. Docker คืออะไร?

Docker เป็นแพลตฟอร์มที่ช่วยให้การพัฒนาและจัดการแอปพลิเคชันเป็นไปอย่างรวดเร็วและมีประสิทธิภาพ โดย Docker ใช้เทคโนโลยี Containerization เพื่อสร้างและรันแอปพลิเคชันในรูปแบบที่เรียกว่า Container ซึ่งเป็นหน่วยการทำงานที่มีความเป็น

อิสระและมีทุกสิ่งที่จำเป็นสำหรับการทำงานของแอปพลิเคชัน เช่น โค้ด, ไลบรารี, ระบบไฟล์ และตัวแปรที่จำเป็น ทำให้สามารถพกพาและรันได้อย่างยืดหยุ่นในทุกระบบที่ติดตั้ง Docker ไว้

1.5.1. แนวคิดเบื้องต้นเกี่ยวกับ Docker

Docker เป็นแพลตฟอร์มที่ช่วยให้การสร้าง รัน และจัดการแอปพลิเคชันในรูปแบบ Container เป็นไปอย่างสะดวกและมีประสิทธิภาพ ซึ่งแตกต่างจากการใช้งาน Virtual Machine ที่ต้องจำลองระบบปฏิบัติการแยกกัน Docker จะใช้เคอร์เนลของ Host OS ร่วมกันในระดับโปรเซส ดังนั้น Container จึงมีน้ำหนักเบาและใช้ทรัพยากรน้อยกว่า VM อย่างมาก ทำให้สามารถรันหลายๆ Container พร้อมกันได้บนฮาร์ดแวร์เดียว

1.5.2. ส่วนประกอบสำคัญของ Docker

1. **Docker Engine**: เป็นซอฟต์แวร์หลักของ Docker ที่ทำหน้าที่รันและจัดการ Container ประกอบด้วยสองส่วนสำคัญ คือ

- ☐ Docker Daemon (dockerd): เป็นตัวควบคุมการทำงานของ Docker โดยทำหน้าที่ในการสร้าง จัดการ และรัน Container รวมถึงทำงานร่วมกับ Docker CLI
- ☐ Docker CLI: เป็นเครื่องมือบรรทัดคำสั่งที่ใช้ในการสื่อสารกับ Docker Daemon ทำให้เราสามารถสั่งการ เช่น การสร้างหรือรัน Container ได้

2. **Docker Image**: Docker Image เป็นต้นแบบของ Container ที่บรรจุโค้ด ไลบรารี และทรัพยากรที่จำเป็นสำหรับการทำงานของแอปพลิเคชัน โดยแต่ละ Image จะถูกสร้างจาก Dockerfile ซึ่งเป็นไฟล์ที่กำหนดขั้นตอนการสร้าง Image เช่น การติดตั้งแพ็กเกจต่างๆ การคัดลอกไฟล์ และการกำหนดค่าที่จำเป็น

3. **Docker Container**: Container คือ Instance ของ Docker Image ที่รันอยู่จริง ทำให้เราสามารถใช้งานแอปพลิเคชันหรือบริการต่างๆ ได้ในรูปแบบของ Container ซึ่งทำให้มีความยืดหยุ่นในการจัดการและสามารถรันแอปพลิเคชันหลายตัวในสภาพแวดล้อมที่แยกจากกันได้อย่างปลอดภัย

4. **Docker Compose**: เป็นเครื่องมือที่ใช้สำหรับจัดการหลายๆ Container ที่ต้องการทำงานร่วมกัน โดยสามารถกำหนดการตั้งค่าของแต่ละ Container ได้ในไฟล์เดียว (ไฟล์ docker-compose.yml) เช่น การเชื่อมต่อกับเครือข่าย การกำหนดค่าทรัพยากร และการตั้งค่าอื่นๆ เหมาะสำหรับการพัฒนาและทดสอบแอปพลิเคชันที่มีหลายองค์ประกอบ เช่น การรันฐานข้อมูล และเว็บเซิร์ฟเวอร์พร้อมกัน

5. **Docker Hub**: เป็นบริการคลังเก็บ Image ที่มีอยู่บนอินเทอร์เน็ต Docker Hub ช่วยให้ผู้ใช้สามารถดาวน์โหลด Image ที่มีอยู่แล้ว หรืออัปโหลด Image ของตัวเองได้ ซึ่งทำให้นักพัฒนาสามารถเข้าถึง Image ต่างๆ ได้อย่างสะดวกโดยไม่ต้องสร้าง Image ใหม่จากศูนย์

1.5.3. การทำงานของ Docker

1. สร้าง Image จาก Dockerfile: นักพัฒนาสามารถสร้าง Docker Image โดยใช้ Dockerfile ซึ่งเป็นไฟล์ที่ประกอบไปด้วยคำสั่งและขั้นตอนการติดตั้งและการตั้งค่าต่างๆ เช่น การติดตั้งไลบรารี การคัดลอกไฟล์ และการกำหนดคำสั่งที่ต้องรันเมื่อเริ่มต้น Container
2. สร้างและรัน Container: เมื่อมี Image แล้ว สามารถสร้าง Container ได้โดยการใช้คำสั่ง เช่น docker run ซึ่งจะสร้าง Instance ของ Image ที่ทำงานอยู่จริง Container นี้จะทำงานเป็นแอปพลิเคชันที่แยกออกจากระบบปฏิบัติการหลัก แต่ยังสามารถสื่อสารกับ Host ได้ตามการตั้งค่า
3. การจัดการ Container ด้วย Docker Compose: หากแอปพลิเคชันต้องการหลาย Container ทำงานร่วมกัน เช่น มีทั้งฐานข้อมูลและ API นักพัฒนาสามารถใช้ Docker Compose เพื่อสร้างและจัดการ Container หลายๆ ตัวพร้อมกันได้ ทำให้การตั้งค่าทั้งหมดถูกรวมอยู่ในไฟล์เดียว และสามารถเรียกใช้ทุก Container ได้ด้วยคำสั่งเดียว (docker-compose up)
4. การจัดการและการปรับแต่ง Container: Docker รองรับการควบคุมการใช้ทรัพยากรของ Container เช่น CPU และ RAM ซึ่งช่วยให้การทำงานของแอปพลิเคชันมีประสิทธิภาพมากขึ้น โดยสามารถกำหนดการใช้ทรัพยากรให้เหมาะสมตามความต้องการ

1.5.4. ข้อดีของ Docker

1. พกพาสะดวกและสภาพแวดล้อมคงที่: Docker ช่วยให้นักพัฒนาสามารถรันแอปพลิเคชันใน Container เดียวกันในทุกๆ สภาพแวดล้อม เช่น การพัฒนา การทดสอบ และการใช้งานจริง โดยที่ไม่ต้องปรับแต่งใหม่
2. การจัดการทรัพยากรที่มีประสิทธิภาพ: Container ใช้ทรัพยากรน้อยกว่า Virtual Machine เพราะไม่ต้องมี Guest OS ทำให้สามารถรันหลายๆ Container พร้อมกันบนเครื่องเดียวได้อย่างมีประสิทธิภาพ
3. การเริ่มต้นและหยุดใช้งานรวดเร็ว: Container สามารถเริ่มต้นและหยุดใช้งานได้รวดเร็ว เพราะไม่ต้องรันระบบปฏิบัติการใหม่ จึงเหมาะกับงานที่ต้องการปรับเปลี่ยนบ่อย เช่น DevOps
4. รองรับการทำงานแบบ Microservices: Docker เหมาะกับการทำงานในสถาปัตยกรรมแบบ Microservices ที่ต้องการแยกแอปพลิเคชันออกเป็นส่วนย่อยๆ แต่ละส่วนทำงานแยกจากกันและมีความเป็นอิสระ
5. การควบคุมและจัดการทรัพยากรง่าย: Docker ช่วยให้สามารถควบคุมการใช้ทรัพยากรของแต่ละ Container ได้อย่างละเอียด เช่น การกำหนดขีดจำกัดการใช้ CPU และ RAM เพื่อให้แอปพลิเคชันมีประสิทธิภาพที่ดีที่สุด

Docker เป็นเทคโนโลยีที่ช่วยให้การพัฒนาและการจัดการแอปพลิเคชันเป็นไปอย่างรวดเร็วและยืดหยุ่นด้วยการใช้ Container ซึ่งเป็นหน่วยการทำงานที่เบากว่า Virtual Machine ช่วยให้การพัฒนาแอปพลิเคชันหลายๆ ส่วนทำงานอย่างเป็นอิสระในรูปแบบ Microservices แต่การใช้งาน Docker ก็มีข้อจำกัดเช่นกัน ซึ่งควรพิจารณาตามความเหมาะสม

1.6. Microservice คืออะไร?

Microservices คือสถาปัตยกรรมการออกแบบซอฟต์แวร์ที่เน้นการแบ่งแยกส่วนของระบบออกเป็นบริการย่อยๆ (Service) ที่เป็นอิสระต่อกัน โดยแต่ละบริการจะรับผิดชอบเฉพาะงานหรือฟังก์ชันหนึ่งๆ ของระบบ และสามารถสื่อสารกับบริการอื่นๆ ผ่านโปรโตคอล API (เช่น HTTP/REST, gRPC, หรือ Message Queue) ทำให้ระบบมีความยืดหยุ่นสูง สามารถพัฒนา ทดสอบ และ

ปรับปรุงแต่ละส่วนแยกกันได้ รวมทั้งมีความสามารถในการสเกล (Scale) ตามความต้องการได้ดีกว่าการออกแบบระบบแบบ Monolithic ที่รวมทุกฟังก์ชันไว้ในแอปพลิเคชันเดียว

1.6.1. แนวคิดและการทำงานของ Microservices

ในการออกแบบแบบ Microservices ระบบจะถูกแบ่งเป็นบริการย่อยๆ ซึ่งแต่ละบริการจะรับผิดชอบหน้าที่อย่างใดอย่างหนึ่ง เช่น การจัดการข้อมูลผู้ใช้ การประมวลผลข้อมูล การจัดการคำสั่งซื้อ เป็นต้น การแบ่งแยกแบบนี้ทำให้ทุกบริการมีความเป็นอิสระ สามารถปรับแต่งและพัฒนาแยกจากกันได้โดยไม่กระทบส่วนอื่นๆ ของระบบ ตัวอย่างเช่น:

- ☐ **User Service** – จัดการข้อมูลผู้ใช้และการยืนยันตัวตน
- ☐ **Product Service** – จัดการข้อมูลสินค้า
- ☐ **Order Service** – จัดการข้อมูลคำสั่งซื้อ

1.6.2. องค์ประกอบสำคัญของ Microservices

1. **การแยกเป็นอิสระ (Decentralized Services)**: แต่ละบริการจะทำงานแยกจากกันโดยสมบูรณ์ สามารถพัฒนาและปรับปรุงโดยทีมงานที่ดูแลเฉพาะบริการนั้นๆ ได้โดยไม่ส่งผลกระทบต่อบริการอื่นๆ

2. **การสื่อสารผ่าน API**: Microservices สื่อสารระหว่างกันโดยใช้ API เช่น RESTful, gRPC, หรือใช้ Message Queue อย่าง RabbitMQ, Kafka เป็นต้น ทำให้บริการเหล่านี้สามารถแลกเปลี่ยนข้อมูลระหว่างกันได้แม้จะอยู่บนสถาปัตยกรรมหรือภาษาโปรแกรมที่ต่างกัน

3. **ฐานข้อมูลแยกส่วน (Decentralized Data Management)**: แต่ละบริการมีฐานข้อมูลเป็นของตัวเอง ทำให้สามารถใช้ฐานข้อมูลประเภทที่เหมาะสมกับงานของตน เช่น บริการจัดการผู้ใช้อาจใช้ฐานข้อมูล SQL ในขณะที่บริการวิเคราะห์ข้อมูลอาจใช้ NoSQL หรือฐานข้อมูลเชิงกราฟ (Graph Database)

4. **ความสามารถในการสเกล (Scalability)**: Microservices สามารถเพิ่มหรือลดจำนวน Instance ของบริการได้ตามความต้องการ เช่น หากบริการจัดการคำสั่งซื้อต้องรองรับปริมาณการใช้งานมาก สามารถเพิ่มจำนวนของ Order Service ได้โดยไม่ต้องเพิ่มบริการอื่นๆ

5. **การใช้เทคโนโลยีที่เหมาะสม (Polyglot Programming)**: ในสถาปัตยกรรม Microservices แต่ละบริการสามารถเลือกใช้ภาษาโปรแกรมและเทคโนโลยีที่เหมาะสมกับงานของตัวเอง เช่น ใช้ Python สำหรับการวิเคราะห์ข้อมูล ใช้ Java หรือ .NET สำหรับระบบหลัก เป็นต้น

6. **การติดตั้งและปรับปรุงง่าย (Continuous Deployment)**: แต่ละบริการสามารถปรับปรุง แก้ไข และติดตั้งใหม่ได้อิสระ โดยไม่จำเป็นต้องปล่อยซอฟต์แวร์ทั้งหมดใหม่ ซึ่งช่วยให้การปรับปรุงหรือแก้ไขบั๊กสามารถทำได้ทันที

1.6.3. ข้อดีของ Microservices

1. **ความยืดหยุ่นในการพัฒนาและการทำงานเป็นทีม**: ทีมพัฒนาสามารถรับผิดชอบเฉพาะบริการที่ตนพัฒนาได้โดยตรง ทำให้แต่ละทีมสามารถเลือกเทคโนโลยีและการทำงานที่เหมาะสมได้ ช่วยให้ระบบพัฒนาได้เร็วขึ้นและมีคุณภาพสูงขึ้น

2. **ความสามารถในการสเกลสูง**: Microservices สามารถเพิ่มหรือลดจำนวนของบริการต่างๆ ตามการใช้งานจริงได้ การสเกลนี้ช่วยให้เรารองรับปริมาณการใช้งานที่เพิ่มขึ้นได้โดยไม่ต้องเพิ่มกำลังทุกส่วนของระบบ

3. **รองรับการเปลี่ยนแปลงและปรับปรุงได้ง่าย**: การอัปเดตหรือปรับปรุงสามารถทำได้เฉพาะในบางบริการ ไม่จำเป็นต้องปรับปรุงทั้งระบบ จึงช่วยลดความเสี่ยงและเวลาในการปรับปรุง

4. **ความทนทาน (Fault Tolerance)**: หากบริการหนึ่งล้มเหลว บริการอื่นๆ ที่เหลือจะยังทำงานต่อไปได้ ซึ่งช่วยลดผลกระทบจากปัญหาที่เกิดขึ้นกับบริการบางส่วน

1.6.4. ข้อเสียของ Microservices

1. **ความซับซ้อนในการออกแบบและจัดการ**: เนื่องจาก Microservices ต้องมีการเชื่อมโยงบริการหลายๆ ตัว การออกแบบและจัดการสถาปัตยกรรมจึงซับซ้อนมากขึ้น ทำให้ต้องมีการวางแผนที่ดีในเรื่องของการจัดการบริการ การตรวจสอบ และการบำรุงรักษา

2. **ภาระในการสื่อสารระหว่างบริการ**: การสื่อสารระหว่างบริการต้องอาศัยโปรโตคอล เช่น HTTP, gRPC, หรือ Message Queue ซึ่งทำให้เกิดความหน่วง (Latency) และทำให้ต้องมีการจัดการกับการเกิดความล้มเหลวในการสื่อสาร

3. **ค่าใช้จ่ายในการบำรุงรักษาสูงขึ้น**: เนื่องจากแต่ละบริการมีฐานข้อมูลและทรัพยากรแยกกัน ทำให้มีค่าใช้จ่ายและความยุ่งยากในการบำรุงรักษามากขึ้น รวมถึงการจัดการด้านความปลอดภัยและการสำรองข้อมูลที่ซับซ้อน

4. **การทดสอบยากขึ้น**: การทดสอบ Microservices ต้องตรวจสอบการทำงานของแต่ละบริการและการทำงานร่วมกัน ซึ่งทำให้ต้องออกแบบการทดสอบที่ซับซ้อนมากขึ้น รวมถึงต้องใช้เครื่องมือเฉพาะสำหรับการทดสอบระบบที่มีหลายบริการ

1.6.5. การใช้งาน Microservices ในโลกจริง

1. **การพัฒนาแอปพลิเคชันแบบ Cloud-native**: Microservices เป็นพื้นฐานของการพัฒนาแอปพลิเคชันแบบ Cloud-native ที่ใช้ Cloud Infrastructure ในการบริหารจัดการ เช่น การใช้ Kubernetes เพื่อจัดการบริการในรูปแบบ Container ซึ่งทำให้การปรับขนาดและการจัดการทรัพยากรเป็นไปอย่างมีประสิทธิภาพ

2. **การจัดการทราฟฟิกด้วย API Gateway**: ระบบที่ใช้ Microservices จะมีบริการหลายตัวทำงานร่วมกัน ดังนั้นการใช้ API Gateway ช่วยจัดการการเข้าถึงบริการทั้งหมดให้เป็นระบบระเบียบ และจัดการทราฟฟิกที่เข้าถึงบริการต่างๆ ได้อย่างมีประสิทธิภาพ

3. **การนำมาใช้ในองค์กรที่มีการเปลี่ยนแปลงบ่อย**: เนื่องจาก Microservices สามารถปรับปรุงเฉพาะส่วนได้ จึงเหมาะกับองค์กรที่ต้องการเปลี่ยนแปลงและปรับปรุงซอฟต์แวร์อย่างต่อเนื่องเพื่อรองรับตลาดหรือการพัฒนาใหม่ๆ

1.6.6. การเปรียบเทียบ Microservices กับ Monolithic

คุณสมบัติ	Monolithic Architecture	Microservices Architecture
การพัฒนา	ทุกฟังก์ชันรวมอยู่ในแอปพลิเคชันเดียว	แต่ละฟังก์ชันแยกเป็นบริการย่อยๆ
การสเกล	สเกลทั้งแอปพลิเคชันพร้อมกัน	สเกลเฉพาะบริการที่ต้องการได้
ความซับซ้อน	จัดการง่าย แต่ไม่ยืดหยุ่น	ซับซ้อนกว่าแต่มีความยืดหยุ่นสูง
การปรับปรุง	ต้องปรับปรุงทั้งแอปพลิเคชัน	ปรับปรุงเฉพาะบริการได้โดยไม่กระทบส่วนอื่น
การพัฒนาแบบทีม	ทีมต้องทำงานร่วมกันบนแอปพลิเคชันเดียว	ทีมสามารถพัฒนาและจัดการบริการแยกกัน
การปรับใช้ (Deployment)	ปรับใช้ทั้งหมดพร้อมกันเป็นแพ็กเกจเดียว	ปรับใช้ทีละบริการได้ตามความต้องการ
การบำรุงรักษา	ซับซ้อนเมื่อแอปพลิเคชันขยายขนาด	บำรุงรักษาแต่ละบริการได้แยกกัน
ความทนทาน	ปัญหาส่วนใดส่วนหนึ่งอาจส่งผลกระทบทั้งระบบ	หากบริการใดล้มเหลว บริการอื่นยังทำงานได้
การใช้เทคโนโลยี	ใช้เทคโนโลยีเดียวทั้งระบบ	แต่ละบริการเลือกใช้เทคโนโลยีที่เหมาะสมได้

Microservices เป็นสถาปัตยกรรมที่ช่วยให้แอปพลิเคชันมีความยืดหยุ่นและสามารถปรับเปลี่ยนตามความต้องการขององค์กรได้ ทำให้แต่ละส่วนของระบบเป็นอิสระจากกัน ซึ่งเหมาะกับการพัฒนาแอปพลิเคชันที่มีการเปลี่ยนแปลงบ่อย หรือแอปพลิเคชันขนาดใหญ่ที่ต้องการประสิทธิภาพในการจัดการทรัพยากรและการสเกล

1.7. API Gateway คืออะไร?

API Gateway คือส่วนกลางในการจัดการการสื่อสารระหว่างผู้ใช้ (Client) และบริการต่างๆ ภายในระบบที่ใช้สถาปัตยกรรมแบบ Microservices โดยมีหน้าที่รับคำขอจากผู้ใช้ ส่งต่อคำขอนั้นไปยังบริการต่างๆ ที่เกี่ยวข้อง และส่งผลลัพธ์กลับไปยังผู้ใช้ ซึ่งช่วยให้การสื่อสารภายในระบบเป็นไปอย่างมีประสิทธิภาพและง่ายต่อการจัดการ นอกจากนี้ API Gateway ยังมีฟังก์ชันในการจัดการการควบคุมการเข้าถึง (Access Control), การรักษาความปลอดภัย (Security), การจัดการการสเกล (Load Balancing) และการแคชข้อมูล (Caching) ทำให้เป็นส่วนสำคัญของระบบ Microservices ที่มีการจัดการหลายบริการ

1.7.1. หน้าที่และการทำงานของ API Gateway

1. การควบคุมเส้นทาง (Routing): API Gateway ทำหน้าที่จัดการเส้นทางของคำขอที่เข้ามา โดยจะแยกแยะและส่งคำขอนั้นไปยังบริการที่เกี่ยวข้อง เช่น หากผู้ใช้ขอข้อมูลผู้ใช้งาน API Gateway จะส่งคำขอนั้นไปยัง User Service เป็นต้น

2. การแปลงข้อมูล (Transformation): ในบางครั้ง คำขอและการตอบกลับจำเป็นต้องมีการแปลงข้อมูลหรือจัดรูปแบบให้ตรงกับความต้องการของผู้ใช้ เช่น การแปลงข้อมูลจาก XML เป็น JSON เพื่อความสะดวกในการใช้งาน

3. การควบคุมการเข้าถึง (Access Control): API Gateway ทำหน้าที่ตรวจสอบสิทธิ์และการรับรองตัวตน (Authentication & Authorization) ของผู้ใช้ เช่น ตรวจสอบว่าเป็นผู้ใช้ที่ได้รับอนุญาตหรือไม่ก่อนส่งคำขอไปยังบริการต่างๆ ทำให้ระบบมีความปลอดภัยมากขึ้น

4. การรักษาความปลอดภัย (Security): API Gateway สามารถจัดการการป้องกันภัยคุกคามต่างๆ เช่น การป้องกันการโจมตีแบบ DDoS, การใช้ HTTPS เพื่อลดความเสี่ยงในการแอบดักข้อมูลระหว่างการสื่อสาร และการจำกัดอัตราการเรียกใช้งาน (Rate Limiting) เพื่อป้องกันการใช้งานที่มากเกินไป

5. การจัดการแคช (Caching): API Gateway มีการเก็บข้อมูลที่ถูกเรียกใช้บ่อยในหน่วยความจำ เพื่อเพิ่มประสิทธิภาพในการให้บริการ โดยไม่ต้องส่งคำขอไปยังบริการต้นทางทุกครั้ง ช่วยลดภาระงานของบริการต่างๆ

6. การควบคุมการสเกล (Load Balancing): API Gateway ช่วยกระจายคำขอที่เข้ามายังเซิร์ฟเวอร์หรือบริการต่างๆ เพื่อให้โหลดการทำงานมีความสมดุลและช่วยให้ระบบรองรับปริมาณการใช้งานที่สูงขึ้นได้อย่างมีประสิทธิภาพ

7. การตรวจสอบและวิเคราะห์ (Monitoring & Analytics): API Gateway สามารถรวบรวมข้อมูลการใช้งาน เช่น จำนวนคำขอ, ระยะเวลาการตอบกลับ, และสถิติการใช้งาน ซึ่งข้อมูลเหล่านี้สามารถนำไปวิเคราะห์เพื่อปรับปรุงบริการและแก้ไขปัญหาที่อาจเกิดขึ้นได้

1.7.2. ข้อดีของการใช้ API Gateway

1. เพิ่มความปลอดภัยให้กับระบบ: API Gateway ช่วยให้สามารถรวมการจัดการด้านความปลอดภัยไว้ในจุดเดียว เช่น การจัดการการตรวจสอบสิทธิ์ (Authentication), การกำหนดนโยบายการเข้าถึง (Authorization Policy) ช่วยลดความเสี่ยงจากภัยคุกคาม

2. จัดการการสื่อสารที่ซับซ้อนได้ง่ายขึ้น: API Gateway ทำหน้าที่เป็นศูนย์กลางการสื่อสารกับบริการย่อย ทำให้การจัดการคำขอและการกำหนดเส้นทาง (Routing) เป็นไปอย่างง่ายและมีระเบียบ

3. เพิ่มประสิทธิภาพด้วยการแคช: คำขอบางคำขอ เช่น ข้อมูลที่มีการเปลี่ยนแปลงน้อย สามารถแคชไว้ใน API Gateway เพื่อลดการเรียกไปยังบริการต้นทาง ทำให้ระบบเร็วขึ้น

4. รองรับการสเกลที่ดีขึ้น: API Gateway ช่วยกระจายโหลดไปยังบริการต่างๆ และช่วยลดภาระการประมวลผลในระบบ ลดความเสี่ยงที่จะเกิดปัญหาโหลดเกินในบางบริการ

5. รวมการตรวจสอบการใช้งานไว้ในจุดเดียว: ช่วยให้นักพัฒนาสามารถตรวจสอบการใช้งานระบบแบบรวมศูนย์ ซึ่งช่วยในการตรวจหาปัญหาได้รวดเร็ว รวมถึงการบันทึกข้อมูลการใช้งานที่ช่วยวิเคราะห์และปรับปรุงบริการได้

1.7.3. ข้อเสียของการใช้ API Gateway

1. เป็น Single Point of Failure: หาก API Gateway เกิดปัญหา ระบบทั้งหมดอาจไม่สามารถทำงานได้ ดังนั้นจึงต้องมีการวางแผนการทำงานแบบสำรอง (Redundancy) หรือการกระจายการใช้งานเพื่อป้องกันปัญหา
2. เพิ่มความซับซ้อนของระบบ: การเพิ่ม API Gateway จะทำให้ระบบมีโครงสร้างซับซ้อนมากขึ้น โดยเฉพาะเมื่อมีการใช้งานร่วมกับหลายบริการและมีการกำหนดนโยบายต่างๆ มากมาย
3. เกิดความล่าช้าเพิ่มขึ้น (Latency): การที่คำขอทั้งหมดต้องผ่าน API Gateway ก่อนที่จะไปยังบริการปลายทาง อาจทำให้เกิดความล่าช้าในระบบโดยเฉพาะหากมีการแปลงข้อมูลหรือแคชที่ซับซ้อน

1.7.4. การใช้งาน API Gateway ในโลกจริง

- ☐ การจัดการการเข้าถึงหลายบริการในระบบ Microservices: API Gateway ช่วยให้ผู้ใช้สามารถเข้าถึงบริการต่างๆ ของระบบที่มีโครงสร้างแบบ Microservices ได้ผ่านจุดเดียว เช่น การเข้าสู่ระบบผู้ใช้ ที่ API Gateway สามารถรับคำขอและส่งต่อไปยังบริการย่อย เช่น บริการตรวจสอบตัวตน บริการจัดการข้อมูลผู้ใช้ เป็นต้น
- ☐ การจัดการแอปพลิเคชันที่รองรับผู้ใช้หลายประเภท: เช่น ระบบอีคอมเมิร์ซที่ต้องรองรับทั้งผู้ใช้ทั่วไปและแอปพลิเคชันที่เป็น Partner สามารถกำหนดสิทธิ์และการเข้าถึงได้แตกต่างกันผ่าน API Gateway เพื่อควบคุมการเข้าถึง
- ☐ การจัดการทราฟฟิกในระบบที่มีปริมาณการเข้าถึงสูง: เช่น บริการที่ใช้ในแอปพลิเคชันแบบ SaaS (Software as a Service) API Gateway จะช่วยกระจายโหลดและควบคุมปริมาณการเข้าถึง

API Gateway เป็นส่วนสำคัญในระบบ Microservices ที่ช่วยจัดการการสื่อสารระหว่างผู้ใช้และบริการต่างๆ ภายในระบบ ทำให้การจัดการการเข้าถึงมีความปลอดภัย และระบบมีความเป็นระเบียบและสามารถปรับแต่งตามความต้องการได้ เช่น การควบคุมการเข้าถึง การแปลงข้อมูล การจัดการการแคชและการกระจายโหลด การใช้ API Gateway ช่วยให้การจัดการระบบขนาดใหญ่ที่มีหลายบริการเป็นไปอย่างมีประสิทธิภาพ

1.8. สรุปการนำเอาเทคโนโลยี Container, Docker, Microservices และ API Gateway มาใช้ในการพัฒนาระบบ

การใช้ Docker Containers, Microservices, และ API Gateway เป็นแนวทางการพัฒนาแอปพลิเคชันในยุคปัจจุบันเป็นแนวทางที่ได้รับความนิยมสูงมาก เนื่องจากมันช่วยปรับปรุงในหลายๆ ด้าน ทั้งในเรื่อง ประสิทธิภาพ, ความทันสมัย, และ ความปลอดภัย นี่คือการสรุปข้อดีของการใช้แนวทางเหล่านี้:

1. **ประสิทธิภาพ (Performance)**

- ☐ Docker Containers: การใช้ Docker ช่วยให้การพัฒนาและการดำเนินงานสามารถแยกกันได้ชัดเจนระหว่างบริการต่างๆ ทำให้สามารถประมวลผลแยกกันได้ดี การจัดการกับ resource และการขยายขนาด (scaling) ทำได้ง่ายและมีประสิทธิภาพสูงขึ้น เนื่องจาก containers สามารถทำงานได้บนเครื่องหลายเครื่องหรือหลายสภาพแวดล้อมได้โดยไม่ต้องแก้ไขโค้ด
- ☐ Microservices: การแบ่งแอปพลิเคชันออกเป็น Microservices ช่วยให้สามารถพัฒนาและปรับปรุงบริการแต่ละตัวได้อย่างรวดเร็วและมีประสิทธิภาพ ไม่ต้องปรับแก้ระบบทั้งหมด ระบบสามารถขยายได้ตามความต้องการ (elastic scaling) และรองรับการทำงานในหลายภูมิภาคหรือเซิร์ฟเวอร์
- ☐ API Gateway: ทำหน้าที่เป็นตัวกลางในการจัดการการเรียกใช้งานบริการต่างๆ ของ Microservices ช่วยลดภาระการจัดการการเชื่อมต่อที่ซับซ้อนระหว่าง microservices และช่วยเพิ่มความเร็วในการประมวลผลการเรียก API รวมถึงสามารถปรับแต่งการใช้ API ได้อย่างมีประสิทธิภาพ

2. **ความทันสมัย (Modernization)**

- ☐ การใช้ Docker และ Microservices ช่วยให้การพัฒนาระบบสามารถปรับตัวได้อย่างรวดเร็วตามเทคโนโลยีใหม่ๆ ได้ง่าย เพราะแต่ละ Microservice สามารถใช้เทคโนโลยีหรือเวอร์ชันที่ต่างกันได้ตามความเหมาะสม
- ☐ API Gateway ทำให้การเชื่อมต่อระหว่างบริการต่างๆ สามารถทำได้ในลักษณะของ “Single Entry Point” ช่วยเพิ่มความสะดวกในการจัดการและบำรุงรักษาระบบ
- ☐ แนวทางเหล่านี้ช่วยให้นักพัฒนาสามารถใช้ CI/CD (Continuous Integration / Continuous Deployment) เพื่ออัปเดตระบบได้บ่อยและสะดวก ทำให้สามารถนำเทคโนโลยีและฟีเจอร์ใหม่ๆ มาใช้งานได้อย่างรวดเร็ว

3. **ความปลอดภัย (Security)**

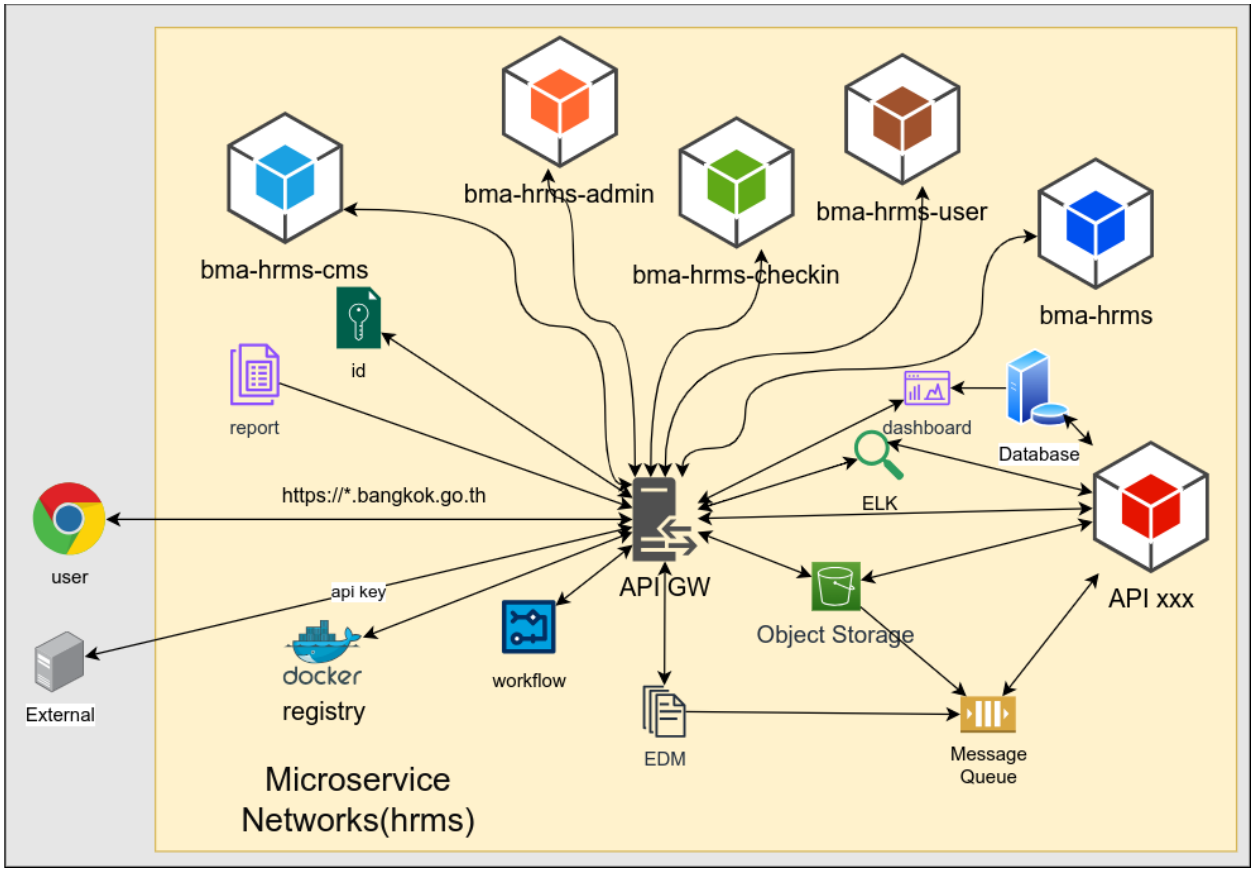
- ☐ Docker: Docker ช่วยแยกแต่ละบริการใน containers ทำให้การจัดการสิทธิ์การเข้าถึงหรือการใช้งานระบบสามารถทำได้ดีขึ้น ถ้าเกิดการเจาะระบบใน container หนึ่งก็ไม่สามารถเข้าถึงข้อมูลใน container อื่นได้ทันที
- ☐ Microservices: การแยกบริการออกเป็น Microservices ช่วยจำกัดการเข้าถึงข้อมูลและลดการเสี่ยงจากการโจมตีแบบ “single point of failure” เนื่องจากระบบแยกออกเป็นส่วนๆ ที่สามารถควบคุมการเข้าถึงได้อย่างชัดเจน
- ☐ API Gateway: API Gateway ทำหน้าที่เป็นตัวกลางในการรับส่งข้อมูลและการเข้าถึง APIs ซึ่งช่วยให้สามารถควบคุมการเข้าถึงด้วยการใช้งานเช่นการพิสูจน์ตัวตน (Authentication), การอนุญาต (Authorization), การป้องกันการโจมตี (Rate Limiting), และการตรวจสอบความปลอดภัย (Security Auditing)

การใช้ Docker Containers, Microservices, และ API Gateway ไม่เพียงแต่ช่วยเพิ่ม ประสิทธิภาพ ในการพัฒนาและการขยายระบบ แต่ยังช่วยในการทำให้ระบบมีความ ทันสมัย และสามารถรองรับเทคโนโลยีใหม่ๆ ได้อย่างง่ายดาย พร้อมทั้งสามารถรักษาความ ปลอดภัย ของระบบได้อย่างมีประสิทธิภาพ

การนำเทคโนโลยีเหล่านี้มาปรับใช้ในการพัฒนาระบบทำให้การสร้างแอปพลิเคชันที่มีความยืดหยุ่นสูง, ปรับขยายได้ตามต้องการ และตอบสนองต่อความปลอดภัยในยุคดิจิทัลได้เป็นอย่างดี

1.9. ขั้นตอนการติดตั้งระบบ

ผู้อ่านจำเป็นต้องมีความรู้และเข้าใจคำสั่ง และระบบไฟล์ของ Linux เป็นอย่างดีมาก่อน เอกสารจะแสดงวิธีการติดตั้งระบบของ BMA HRMS บนเทคโนโลยีคอนเทนเนอร์ มีการปูพื้นคำสั่งพื้นฐานของ Docker ที่จำเป็นให้ ลักษณะการติดตั้งและใช้งานจะอยู่ในรูปแบบเดียวกัน ทำให้สามารถจัดการบริการเพิ่มเติมที่จะเกิดขึ้นอนาคตได้ นอกจากการใช้งานผ่านคำสั่งแล้วสามารถใช้ Portainer จัดการผ่าน Web UI ได้ การติดตั้งจะอยู่ในเอกสารนี้



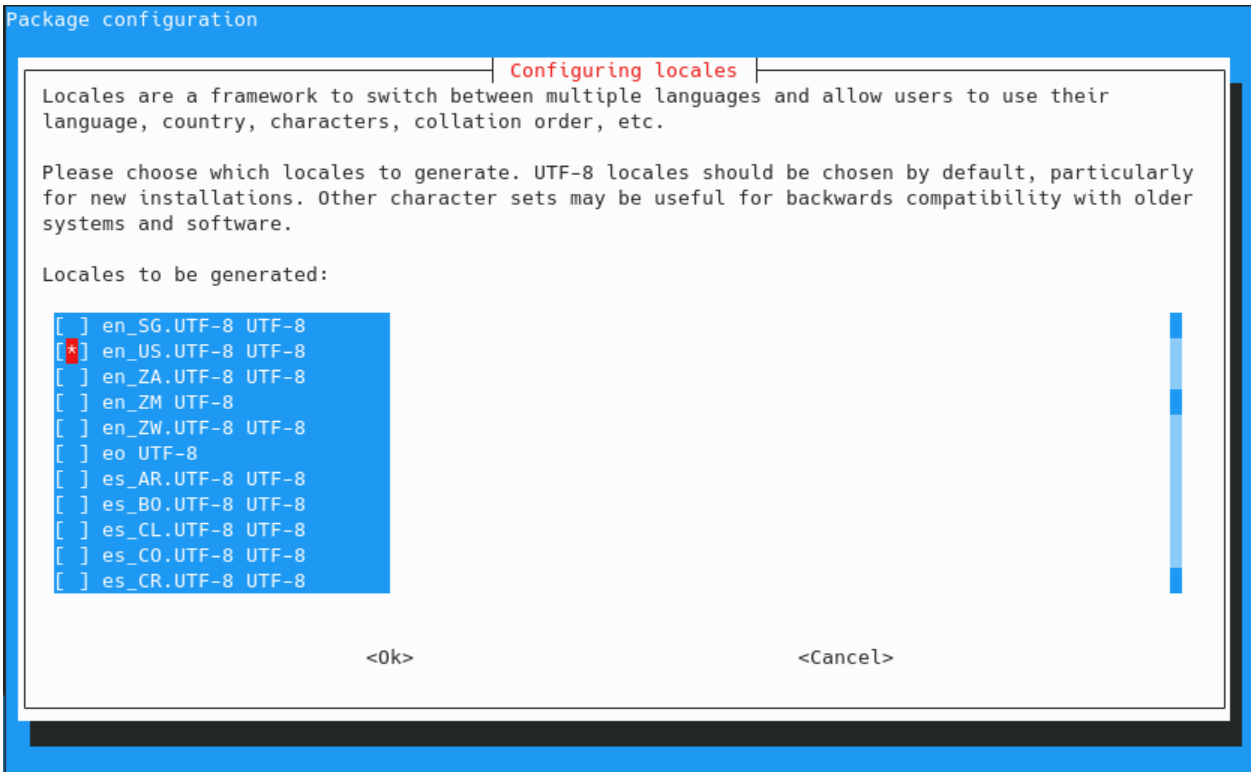
รูป 4-1 Network Diagram

ในสารนี้ในรูปแบบ Web, MS Word หรือ PDF อาจจะมีการบังคับตัดขึ้นบรรทัดใหม่, ใส่รูปแบบข้อความ, White space ฯลฯ ซึ่งอาจจะกระทบการทำงานของคำสั่ง หรือไฟล์สำหรับตั้งค่าระบบได้ การ Copy/Pate ควรตรวจสอบความถูกต้องทุกครั้งก่อนนำไปใช้งาน เอกสารจาก Markdown หรือไฟล์สคริปต์หรือคอนฟิกของระบบจริงๆ จะมีโอกาสผิดพลาดน้อยกว่า

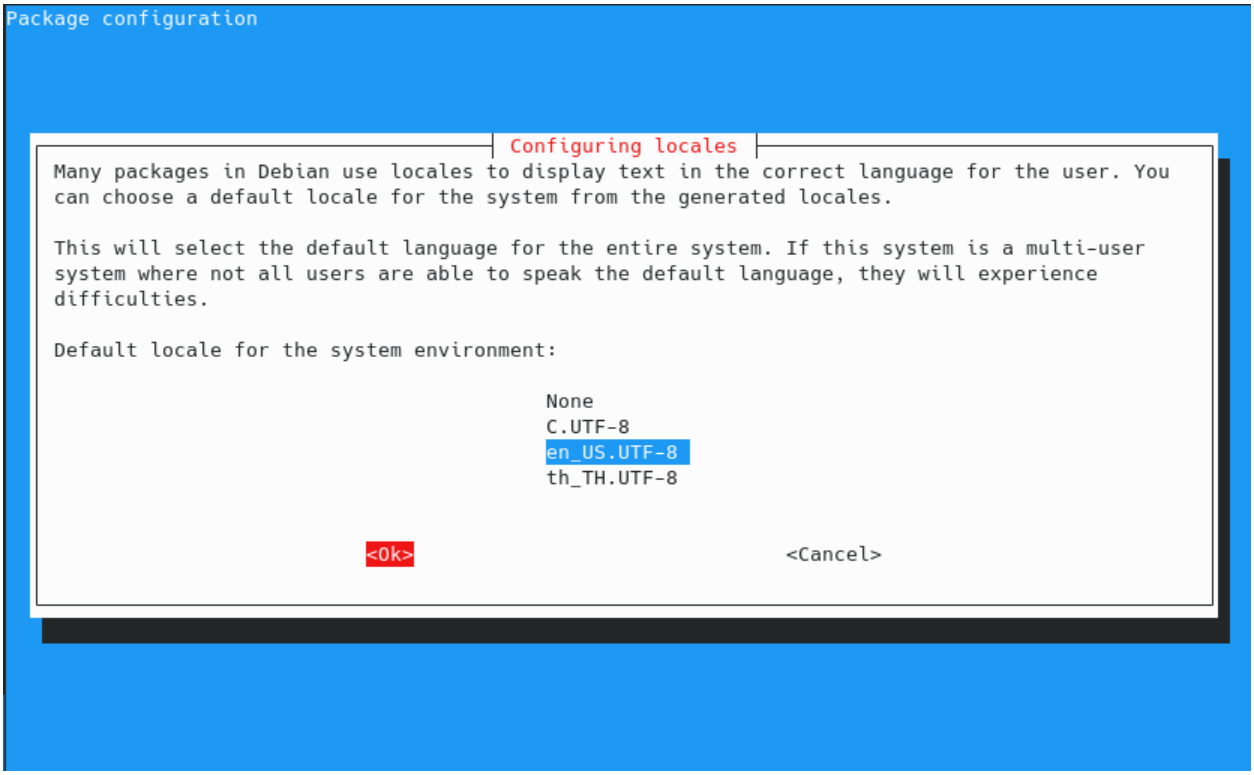
1.9.1. Linux VM

เครื่องโฮสเป็น debian 12 แบบมาตรฐาน ติดตั้งด้วยวิธีการของ Data Center ตั้งค่า locale และ Time Zone ให้เหมาะสำหรับประเทศไทย

```
sudo dpkg-reconfigure locales
sudo timedatectl set-timezone Asia/Bangkok
date
```



รูป 4-2 เลือกlocale



รูป 4-3 locale ตั้งต้น

1.9.2. ติดตั้ง Docker

Docker คอนเทนเนอร์สามารถเรียนรู้ ตั้งค่า ดูแล รักษาได้ง่ายและรวดเร็ว รองรับการขยายตัว สามารถเพิ่มคลัสเตอร์ได้ง่าย และสามารถปรับเปลี่ยนเป็น Kubernates ได้ในอนาคต สำหรับเอกสารนี้จะเป็นการติดตั้ง Docker บน debian 12 สำหรับบนวินโดว์ (เพื่อการทดสอบเท่านั้น)ให้ติดตั้ง docker บน WSL2 การติดตั้งโดยละเอียดดูได้จาก คู่มือในเว็บไซต์หลักของ Docker ตัวอย่างด้านล่างเป็นวิธีการติดตั้งแบบย่อผ่าน convenience script ทำตามวิธีการนี้

```
sudo apt install curl
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker user_name
sudo usermod -aG sudo user_name
```

แทน user_name ด้วยยูสเซอร์ที่ใช้ดูแลระบบ ในตัวอย่างจะให้อยู่ในกลุ่ม docker และ sudo เพื่อให้มีสิทธิ์ในจัดการ service และไฟล์ที่เกิดจาก docker

1.9.3. คำสั่ง Docker พื้นฐาน

คำสั่งพื้นฐานของ docker ที่ใช้

```
docker ps          # ดูรายการคอนเทนเนอร์ที่รันอยู่ในระบบ
docker images      # ดูรายการอิมเมจที่มีในระบบ
docker pull [image_name:tag]    # ดึงอิมเมจมาในระบบ
docker network ls                          # แสดงรายการเน็ตเวิร์กของ docker
docker network create [network_name] # สร้างเน็ตเวิร์กของ docker
```

1.9.4. การใช้งานไฟล์ Docker Compose

1.9.4.1. Docker Compose File คืออะไร?

Docker Compose File เป็นไฟล์ที่มีรูปแบบเป็น YAML (มักมีชื่อไฟล์ docker-compose.yml หรือ compose.yaml) ซึ่งใช้สำหรับการกำหนดค่าและจัดการการทำงานของ **หลายคอนเทนเนอร์** ในโปรเจกต์เดียวกัน ช่วยให้นักพัฒนาสามารถกำหนดบริการต่างๆ ที่คอนเทนเนอร์จะต้องรัน เช่น ฐานข้อมูล เว็บเซิร์ฟเวอร์ และบริการอื่นๆ ได้ในที่เดียว

1.9.4.2. Docker Compose File มีไว้ทำไม?

1. จัดการบริการหลายตัวพร้อมกัน เมื่อโปรเจกต์มีหลายคอนเทนเนอร์ เช่น ฐานข้อมูล (MySQL), แอปพลิเคชัน (Node.js), และเครื่องมืออื่นๆ Docker Compose จะช่วยจัดการทุกอย่างในไฟล์เดียว
2. ลดความซับซ้อนของคำสั่ง แทนที่จะต้องใช้คำสั่ง docker run หลายครั้งเพื่อรันคอนเทนเนอร์ต่างๆ สามารถใช้คำสั่งเดียว (docker-compose up) เพื่อรันทุกบริการพร้อมกัน
3. กำหนดค่าที่ซับซ้อน สามารถกำหนดค่าเครือข่าย, การเชื่อมโยงคอนเทนเนอร์, พอร์ต, Volume, และ Environment Variable ได้ในไฟล์เดียว
4. ทำให้การพกพาและย้ายโปรเจกต์ง่ายขึ้น แชร์ docker-compose.yml ไฟล์ให้ผู้อื่นเพื่อสร้างสภาพแวดล้อมที่เหมือนกันได้อย่างง่ายดาย

1.9.4.3. Docker Compose File ใช้งานเพื่ออะไร?

1. การตั้งค่าบริการในระบบที่ซับซ้อน เช่น การตั้งค่าแอปพลิเคชันที่ต้องใช้ฐานข้อมูล Redis และ RabbitMQ ร่วมกัน
2. การพัฒนาและทดสอบในเครื่อง สร้างสภาพแวดล้อมที่เหมือนกับ Production บนเครื่องนักพัฒนา
3. การจัดการ Volume และ Network กำหนด Volume สำหรับเก็บข้อมูล และ Network เพื่อให้บริการต่างๆ สื่อสารกัน
4. รองรับการสเกลระบบ สามารถเพิ่มจำนวนคอนเทนเนอร์ในแต่ละบริการได้ง่ายโดยใช้ docker-compose up –scale

1.9.4.4. ตัวอย่างของไฟล์ docker-compose.yaml

เป็นตัวอย่างแบบง่ายเพื่อใช้งาน เซอร์วิส web ใช้อิมเมจโปรแกรม nginx เนื้อหาไฟล์ในโฟลเดอร์ ./html จะไปปรากฏบนคอนเทนเนอร์ที่ /usr/share/nginx/html เปิดพอร์ต 9082 บนเครื่องโฮส ถ้าไม่สั่งหยุดการทำงาน nginx จะเริ่มตัวเองเองถ้าโฮสเปิดขึ้นเครื่องใหม่ ทำงานในเน็ตเวิร์ก hrms ที่สร้างจากภายนอก

การใช้งานใช้เว็บเบราว์เซอร์ไปที่ http://IP:9082

```
services:
 web:
  image: nginx
  volumes:
   - ./html:/usr/share/nginx/html:ro
  ports:
   - "9082:80"
  restart: unless-stopped
  networks:
   hrms:
networks:
 hrms:
  external: true
```

1.9.4.5. การใช้งานคำสั่ง docker compose

docker compose จะรันคอนเทนเนอร์ของ docker หลายๆตัวพร้อมกันได้ โดยดูการตั้งค่าจากไฟล์ compose.yaml หรือ docker-compose.yaml ที่อยู่ในโฟลเดอร์ที่เรียกคำสั่ง ในการทำงานทั่วไปเพื่อง่ายต่อการใช้งานจะใช้ docker compose เป็นหลัก ซึ่งจะมีผลเฉพาะ service ที่เขียนไว้ในไฟล์ compose

คำสั่ง docker compose พื้นฐาน (ต้องมีไฟล์ compose.yaml หรือ docker-compose.yaml ในโฟลเดอร์ที่เรียกคำสั่ง)

```
docker compose ps       # ดูรายการ service ที่ทำงานอยู่
docker compose up -d # เรียกทุก service ขึ้นมาทำงาน
docker compose up -d [service_name] # เรียกเฉพาะ service ที่ต้องการขึ้นมาทำงาน

docker compose down # หยุดและลบบริการทั้งหมด

docker compose logs # ดู Log ของบริการทั้งหมด

docker compose logs -f [service_name] # ดู Log เฉพาะ service ที่ต้องการ
```

โฟลเดอร์ที่ใช้เริ่มต้นสร้างโปรแกรม บริการต่างๆในหัวข้อต่อๆไปจะสร้างภายใต้ ~/docker/hrms และใช้เน็ตเวิร์กชื่อ hrms ในการสื่อสารระหว่างกัน ทำให้สามารถเรียกใช้ชื่อ service แทน IP Address ได้เหมือนมี DNS ภายใน

```
docker network create hrms # สร้างเน็ตเวิร์ก hrms
mkdir -p ~/docker/hrms     # สร้างโฟลเดอร์
cd ~/docker/hrms           # เข้าไปในโฟลเดอร์
```

ทดสอบการทำงานของ docker ให้สร้างโฟลเดอร์ simple-web มีไฟล์ compose.yaml สร้างโฟลเดอร์ html แล้วสร้างไฟล์ html/index.html ในโฟลเดอร์นั้นพร้อมเนื้อหาเรียกคำสั่ง docker compose up -d

```
mkdir -p simple-web/html
cd simple-web
nano compose.yaml      # สร้างไฟล์ compose
nano html/index.html # สร้างหน้าเว็บสำหรับทดสอบ
```

1.10. Services

การออกแบบใช้สถาปัตยกรรม Microservice จะประกอบจากหลาย Service เพื่อความเป็นระเบียบจะแบ่งไว้สามโฟลเดอร์ apisix(API Gateway), bmahrms-service(3rd Party service), bmahrms(HRMS services)

```
mkdir -P apisix
mkdir -p bmahrms-service/{edm_config,elasticsearch_config,init_mysql,keycloak_config,report-server-templates}
mkdir -p bmahrms-service/report-server-templates/{docx,xlsx}
mkdir -p bmahrms
```

1.10.1. APISIX (API Gateway)

ทำหน้าที่จัดการ https, limit, security, load balance,reverse proxy, และ route ใน Microservice ก่อนใช้งานให้ตั้งค่าโดเมนกับ Public IP ให้เรียบร้อยแล้ว Forward Port 80/443 มาที่เครื่องนี้ การตั้งค่าต่างๆจะทำผ่าน Web API ในตัวอย่างจะใช้ curl

```
mkdir -p apisix/apisix_conf/
nano apisix/apisix_conf/config.yaml # configuration
nano apisix/compose.yaml            # compose file
docker compose up -d
```

apisix/apisix_conf/config.yaml เป็นไฟล์คอนฟิกของ APISIX แก้ค่า put_admin_api_key_here, allow_admin ให้เหมาะสม

```
apisix:
 node_listen: 9080 # APISIX listening port
```

```
 enable_ipv6: false
 enable_control: true
 control:
  ip: "0.0.0.0"
  port: 9092
deployment:
 admin:
  allow_admin: # https://nginx.org/en/docs/http/ngx_http_access_module.html#allow
   - 0.0.0.0/0 # We need to restrict ip access rules for security. 0.0.0.0/0 is for te
st.
  admin_key:
   - name: "admin"
     key: put_admin_api_key_here
     role: admin # admin: manage all configuration data
 etcd:
  host: # it's possible to define multiple etcd hosts addresses of the same et
cd cluster.
   - "http://etcd:2379" # multiple etcd address
  prefix: "/apisix" # apisix configurations prefix
  timeout: 30 # 30 seconds
plugin_attr:
 prometheus:
  export_addr:
   ip: "0.0.0.0"
   port: 9091
```

apisix/compose.yaml จะมีสอง service ตัว apisix จะเป็นโปรแกรมหลัก ส่วน etcd ฐานข้อมูลแบบกระจายตัว ใช้สำหรับเก็บการตั้งค่าของ apisix เน็ตเวิร์ก apisix จะเป็นการสื่อสารระหว่างสองเซอร์วิสเท่านั้น ส่วนเน็ตเวิร์ก hrms จะเป็นการสื่อสารระหว่าง apisix กับโปรแกรมต่างๆของระบบ

```
services:
 apisix:
  image: apache/apisix:${APISIX_IMAGE_TAG:-3.9.0-debian}
  restart: always
  environment:
   - TZ=Asia/Bangkok
  volumes:
```

```
    - ./apisix_conf/config.yaml:/usr/local/apisix/conf/config.yaml:ro
    depends_on:
     - etcd
    ports:
     - "9180:9180/tcp"
     - "80:9080/tcp"
     - "9091:9091/tcp"
     - "443:9443/tcp"
     - "9092:9092/tcp"
    networks:
     hrms:
     apisix:
  etcd:
    image: bitnami/etcd:3.5.11
    restart: always
    volumes:
     - etcd_data:/bitnami/etcd
    environment:
     ETCD_ENABLE_V2: "true"
     ALLOW_NONE_AUTHENTICATION: "yes"
     ETCD_ADVERTISE_CLIENT_URLS: "http://etcd:2379"
     ETCD_LISTEN_CLIENT_URLS: "http://0.0.0.0:2379"
     TZ: "Asia/Bangkok"
    ports:
     - "2379:2379/tcp"
    networks:
     apisix:
networks:
  hrms:
    external: true
  apisix:
    driver: bridge

volumes:
  etcd_data:
    driver: local
```

วิธีการติดตั้ง certificate(HTTPS) เนื่องจากของ *bangkok.go.th มี intermediate certificate ให้นำ root CA รวมกับ intermediate เพื่อให้พร้อมใช้งาน การตั้งค่า APISIX จะทำผ่าน Web API ใช้ curl และ api-key ที่ตั้งไว้

```
# รวม intermediate เข้ากับ certificate หลัก
cat star_bangkok.go.th_2024.crt 'GeoTrust TLS RSA CA G1.crt' >  star_bangkok.go.th_2024.ca-bundle
# ตั้งค่าใน APISIX
curl http://127.0.0.1:9180/apisix/admin/ssls/1 \
-H 'X-API-KEY: put_admin_api_key_here' -X PUT -d '
{
  "cert":"'"$(cat star_bangkok.go.th_2024.ca-bundle)"'",
  "key":"'"$(cat star_bangkok.go.th_2024.key)"'",
  "snis":["*.bangkok.go.th"]
}'
```

ถ้ามีการเซต route ผ่าน APISIX สามารถทดสอบ ผ่าน curl ได้ด้วยคำสั่งนี้(การเซตค่าจะอยู่ในหัวข้ออื่น)

```
curl https://bma-hrms-id.bangkok.go.th -vvv
```

1.10.2. 3rd Party(bmahrms-service)

โปรแกรมกลุ่มนี้ที่พัฒนาโดย 3rd Party หรือ Frappet ไม่ได้มีฟังก์ชันงานเจาะจงสำหรับ HRMS ถูกใช้โดยโปรแกรมของ HRMS(เช่นฐานข้อมูล,เว็บเซิร์ฟเวอร์ ฯลฯ) การตั้งค่าจะแยกมาใส่โฟลเดอร์ bmahrms-service การติดตั้งให้นำ compose.yaml และคอนฟิกมาใส่โฟลเดอร์นี้ให้เรียบร้อยก่อนใช้งาน โปรแกรมมีรายการดังนี้ดังนี้

1.10.2.1. รายการโปรแกรม 3rd Party

- RabbitMQ (bmahrms-mq) ระบบจัดคิวในการสื่อสารใน Microservice มีความน่าเชื่อถือสูง สามารถรับโหลดหนักๆ ในช่วงเวลาสั้นๆได้ โดยไม่ต้องเพิ่มทรัพยากรโดยไม่จำเป็น เช่นระบบการลงเวลา

- Keycloak (bmahrms-id) ใช้สำหรับการ login ในระบบ (Identity Server) เพื่อความปลอดภัยการสื่อสารเฉพาะภายใน จะเป็น https จะสร้าง self-signed certificate ขึ้นมาตัวอย่างนี้มีอายุ 10 ปี

```
cd bmahrms-service/keycloak_config
openssl genrsa -out keycloak.key 2048
openssl req -new -x509 -sha256 -key keycloak.key -out keycloak.crt -days 3650
cp keycloak.key keycloak.pem
cat keycloak.crt >> keycloak.pem
cd ..
```

- **Portainer (bmahrms-portainer)** ระบบจัดการ container มี UI ใช้งานง่าย สามารถสั่ง/เปิด/ปิด/แก้ไข การทำงานของคอนเทนเนอร์ได้ แทนการใช้คำสั่ง docker เฉพาะผู้ดูแลระดับสูงถึงจะเข้าใช้งานส่วนนี้

- **MySQL (bmahrms-postgres)** เป็นฐานข้อมูลของระบบ ในโฟลเดอร์ init_mysql/*.sql เป็นที่เก็บ SQL Script สำหรับสร้างฐานข้อมูลตั้งต้น จะถูกเรียกใช้ครั้งแรกตอนเริ่มฐานข้อมูลเป็นข้อมูลตั้งต้นของระบบ เนื่องจากไฟล์มีขนาดใหญ่หลายบรรทัดจะไม่ใส่ในเอกสารนี้

- **Frappet Report Server (bmahrms-report-server)** เป็น Microservice ใช้สำหรับการออกรายงาน ไม่จำเป็นต้องมี domain ของตัวเอง ให้เป็น path ในโดเมนที่มีอยู่ได้ ให้นำไฟล์ต้นแบบซึ่งสร้างจาก MS Word และ Excel มาไว้ที่โฟลเดอร์นี้ report-server-template/docx/files.docx, report-server-template/xlsx/files.xlsx

- **MiniO (bmahrms-s3)** เป็น Object Storage แบบเดียวกับ AWS s3 มีประสิทธิ์ภาพสูงสามารถรับโหลดได้หนักและมีความปลอดภัยกว่าเก็บด้วยระบบไฟล์ทั่วไป สามารถขยายเพิ่มได้ในอนาคต ถูกใช้ในหลายระบบที่ต้องการเก็บไฟล์ Route จะมีส่วน API กับ console การทำ route จะให้ console อยู่ใต้ subfoler https://domain/console ในการติดตั้งบน production อาจจะไม่ได้อยู่ใน node เดียวกับ hrms และ apisix ดังนั้นควรอ้างเป็นชื่อโดเมนเต็มแทน

- **Windmill** รันเวิร์กโฟลว์การทำงานอัตโนมัติจากสคริปต์ หลักๆใช้เพื่อรันตัวสำรองข้อมูล

- **Frappet EDM (Enterprise Document Management)** เป็นระบบจัดการเอกสารภายใน ประกอบไปด้วยโปรแกรมหลายตัว

    - EDM ตัวโปรแกรมหลัก
    - Elasticsearch ฐานข้อมูลดัชนีเอกสารเพื่อการค้นหาภาษาไทยแบบซับซ้อน อ้างผ่าน IP/PORT ไม่มี Route สู่ภายนอก
    - Kibana ใน WebUI การจัดการ Elasticsearch มี Route ใช้เฉพาะนักพัฒนา
    - RabbitMQ จัดการคิวงาน มี Route ใช้เฉพาะนักพัฒนา
    - MiniO ใช้เพื่อเก็บไฟล์ต่างๆ มีหน้า console ใช้เฉพาะนักพัฒนา

นอกเหนือจาก compose.yaml มีไฟล์คอนฟิกสองไฟล์

```
nano edm_config/keycloak.json
nano elasticsearch_config/config.yaml
```

edm_config/keycloak.json

```
{
 "realm": "EDM",
```

```
"auth-server-url": "https://bma-hrms-id.bangkok.go.th/",
"ssl-required": "external",
"resource": "EDM-V1",
"public-client": true,
"confidential-port": 0
}
```

elasticsearch_config/config.yaml

```
network.host: 0.0.0.0
s3.client.default.endpoint: bma-hrms-s3.bangkok.go.th
```

☐ **Kibana** เป็น UI สำหรับ Elasticsearch สำหรับนักพัฒนามาตรวจสอบค่า ควรใช้จาก VPN และเน็ตเวิร์กภายในเท่านั้น เพื่อความปลอดภัย จะใช้ API gateway ทำ Basic Authentication และ https สำหรับนักพัฒนาเท่านั้น ใช้ผ่าน VPN หรือเน็ตเวิร์กภายในเท่านั้น

1.10.2.2. ตัวแปรแวดล้อม 3rd Party service

การตั้งค่าและรหัสผ่านของคอนเทนเนอร์ผ่านตัวแปรแวดล้อมทำได้สองแบบ - ทำแต่ละ service ใน compose.yaml - แยกออกมาเป็นไฟล์เพื่อใช้ร่วมกันหลาย Service ก็ได้ ในตัวอย่างนี้จะใช้ ไฟล์ service.env ให้แก้ตัวแปรให้เหมาะสม KEYCLOAK_ADMIN_PASSWORD, KC_DB_PASSWORD, POSTGRES_PASSWORD, MINIO_ROOT_PASSWORD RABBITMQ_DEFAULT_PASS, MYSQL_ROOT_PASSWORD, MYSQL_PASSWORD

docker/bmahrms-service/service.env ``` # Generic TZ=Asia/Bangkok # Keycloak KEYCLOAK_ADMIN=admin KEYCLOAK_ADMIN_PASSWORD=put_keycloak_admin_password_here

Keycloak KC_DB=postgres #KC_DB_URL=jdbc:postgresql://postgres:5432/keycloak KC_DB_URL_HOST=bmahrms-postgres KC_DB_URL_DATABASE=keycloak KC_DB_PASSWORD=put_keycloak_db_password_here KC_DB_USERNAME=keycloak KC_DB_SCHEMA=public

PostgreSQL (keycloak) POSTGRES_DB=keycloak POSTGRES_USER=keycloak POSTGRES_PASSWORD=put_keycloak_db_password_here

MiniO MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=put_minio_admin_password_here MINIO_BROWSER_REDIRECT_URL=https://bmahrms-s3.bangkok.go.th/console/

RabbitMQ RABBITMQ_DEFAULT_USER=admin RABBITMQ_DEFAULT_PASS=put_rabbitmq_admin_password_here

MySQL MYSQL_ROOT_PASSWORD=put_mysql_admin_password_here MYSQL_DATABASE=bma_ehr MYSQL_USER=frappet MYSQL_PASSWORD=put_mysql_db_password_here

Elasticsearch & Kibana xpack. security. enabled= false discovery. type= single-node ELASTICSEARCH_HOSTS=http://bmahrms-elasticsearch:9200

REDIS REDIS_PORT= 6379 REDIS_DATABASES= 16 ``` #### compose. yaml ข อ ง 3rd Party service docker/bmahrms-service/compose.yaml จะใช้ docker volume (keycloak_data, minio_data ฯลฯ) แทน file system (ตัวอย่างก่อนหน้า) เพื่อประสิทธิ์ภาพสูงกว่าในการเก็บข้อมูลที่มีการเขียนอ่านบ่อย

ให้แก้ค่า PUBLIC_KEY MINIO_ACCESS_KEY,MINIO_SECRET_KEY, ipaddress_bma_hrms_id ให้เหมาะสม

```
networks:
 hrms:
  external: true
#clear all volume: docker volume rm $(docker volume ls -q | grep bmahrms-service)
# clear one volume docker volume rm bmahrms-service_mysql_data
volumes:
 keycloak_data:
 minio_data: {}
 rabbitmq_data:
 elasticsearch_data:
 postgres_data:
 mysql_data:
 kibana_data:
services:
 bmahrms-id:
  image: quay.io/keycloak/keycloak:22.0.0
  restart: unless-stopped
  env_file: "service.env"
  depends_on:
   - bmahrms-postgres
  volumes:
   - /etc/localtime:/etc/localtime:ro
   - keycloak_data:/opt/keycloak/data
   - ./keycloak_config:/keycloak_config:ro
  command:
   - start-dev
   --features=token-exchange
   --https-certificate-file=/keycloak_config/keycloak.crt
   --https-certificate-key-file=/keycloak_config/keycloak.pem
```

```
  --http-enabled=true
 networks:
  - hrms
bmahrms-postgres:
 image: postgres:15.3
 restart: unless-stopped
 env_file: "service.env"
 volumes:
  - postgres_data:/var/lib/postgresql/data
 networks:
  - hrms
bmahrms-s3:
 image: minio/minio:RELEASE.2024-07-16T23-46-41Z
 env_file: "service.env"
    # mem_limit: "1g"
 restart: unless-stopped
 command: server /data --console-address ":9001"
 volumes:
  - minio_data:/data
 networks:
  - hrms
bmahrms-mq:
 image: rabbitmq:3-management-alpine
 env_file: "service.env"
 mem_limit: "1g"
 restart: unless-stopped
 ports:
  - 5672:5672
      # -9122:15672 # UI
 volumes:
  - rabbitmq_data:/var/lib/rabbitmq/
 networks:
  - hrms
bmahrms-mysql:
 image: mysql:8.0.17
 env_file: "service.env"
```

```
    command: ["--max_connections=5000"]
    restart: unless-stopped
    ports:
     - "9123:3306"
    volumes:
     - mysql_data:/var/lib/mysql
     - ./init_mysql:/docker-entrypoint-initdb.d/:ro
    networks:
     - hrms
  bmahrms-report-server:
    image: docker.frappet.com/demo/report-server:latest
    restart: unless-stopped
    mem_limit: 1000m
    volumes:
     - ./report-server-templates:/app/templates
    networks:
     - hrms
  bmahrms-elasticsearch:
    image: docker.frappet.com/core/elasticsearch-icu:8.14.3
    env_file: "service.env"
    mem_limit: "4g"
    volumes:
     - elasticsearch_data:/usr/share/elasticsearch/data
     - ./elasticsearch_config/config.yaml:/usr/share/elasticsearch/config/elasticsearch.y
ml
    restart: unless-stopped
       # disable below ulimits if run inside LXC
    ulimits:
     memlock:
      soft: -1
      hard: -1
    networks:
     - hrms
  bmahrms-kibana:
    image: docker.elastic.co/kibana/kibana:8.14.3
    env_file: "service.env"
    depends_on:
```

```
  - bmahrms-elasticsearch
 volumes:
  - kibana_data:/usr/share/kibana/data
 ports:
  - 2130:5601
 restart: always
 networks:
  - hrms
bmahrms-edm:
 image: docker.frappet.com/edm/core
 restart: unless-stopped
 volumes:
  - ./edm_config/keycloak.json:/app/static/keycloak.json:ro
 environment:
  - PUBLIC_KEY=put_realm_edm_public_key_here
  - REALM_URL=https://bma-hrms-id.bangkok.go.th/realms/EDM
  - PREFERRED_AUTH=online
  - MANAGEMENT_ROLE=doc-management
  - MINIO_HOST=bmahrms-s3.bangkok.go.th
  - MINIO_PORT=443
  - MINIO_SSL=true
  - MINIO_ACCESS_KEY=put_minio_access_key_here
  - MINIO_SECRET_KEY=put_minio_secret_key_here
      # Bucket notification event needed to be configured
      # Can use prepare script to create bucket
  - MINIO_BUCKET=edm
  - ELASTICSEARCH_PROTOCOL=http
  - ELASTICSEARCH_HOST=bmahrms-elasticsearch
  - ELASTICSEARCH_PORT=9200
      # Can use prepare script
  - ELASTICSEARCH_INDEX=edm-index
  - AMQ_URL=amqp://admin:put_rabbitmq_admin_password_here@bmahrms-mq:5672
  - AMQ_QUEUE=edm
  - NODE_TLS_REJECT_UNAUTHORIZED=false
 extra_hosts:
  - bma-hrms-id.bangkok.go.th:ipaddress_bma_hrms_id
 depends_on:
```

```
  - bmahrms-postgres

 networks:
  - hrms
 bmahrms-portainer:
  image: portainer/portainer-ce:2.20.3
  volumes:
  - portainer_data:/data
  -/var/run/docker.sock:/var/run/docker.sock
  restart: unless-stopped
  networks:
  - hrms
```

รันคำสั่งดังนี้ในตัวอย่างเรียกใช้งานเฉพาะ Keycloak จะใช้ 2 service ตัวอย่างนี้จะเรียกใช้ฐานข้อมูลแล้วค่อยเรียกใช้โปรแกรม keycloak

```
docker compose up -d bmahrms-postgres
docker compose up -d bmahrms-id
```

1.10.2.3. Route ของ 3rd Service

```
# Keycloak Route
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-id",
 "host": "bma-hrms-id.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "scheme": "https",
  "type": "roundrobin",
  "nodes": {
   "bmahrms-id:8443": 1
     }
   }
}'
# Portainer route
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
```

```
{
 "id": "bmahrms-portainer",
 "host": "bmahrms-portainer.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-portainer:9000": 1
     }
   }
}'
# Report Server Route
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-report-server",
 "hosts": ["bma-hrms.bangkok.go.th","bma-hrms-user.bangkok.go.th"],
 "uris": ["/api/v1/report-template/*"],
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-report-server:80": 1
     }
   }
}'

# MiniO Rote
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-s3",
 "host": "bma-hrms-s3.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-s3:9000": 1
     }
```

```
  }
}'
# Health test
curl -k -I https://bma-hrms-s3.bangkok.go.th/minio/health/live
# MinIO Console Route
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-s3-console",
 "host": "bma-hrms-s3.bangkok.go.th",
 "uri": "/console/*",
 "enable_websocket": true,
 "plugins": {
  "proxy-rewrite": {
   "regex_uri": ["^/console/(.*)","/$1"]
     }
  },
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-s3:9001": 1
     }
  }
}'
# EDM Route
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-edm",
 "host": "bma-hrms-edm.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-edm:80": 1
     }
  }
}'
# Kibana Route
```

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-kibana",
 "host": "bma-hrms-kibana.bangkok.go.th",
 "uri": "/*",
 "plugins": {
   "basic-auth": {}
  },
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-kibana:5601": 1
     }
   }
}'
# Kibana Basic Authentication
curl http://127.0.0.1:9180/apisix/admin/consumers \
-H 'X-API-KEY: put_admin_api_key_here' -X PUT -d '
{
 "username": "admin",
 "plugins": {
   "basic-auth": {
     "username": "admin",
     "password": "kibana-password-here"
         }
     }
}'
```

1.10.3. BMA HRMS

ระบบที่พัฒนาเพื่อ HRMS โดยเฉพาะ จะมี 3 ไฟล์

- ☐ be.env ค่าคอนฟิกของ Backend
- ☐ fe.env ค่าคอนฟิกของ Frontend
- ☐ compose.yaml คอนฟิกของ Docker

โปรแกรม Frntend เป็นส่วนของหน้าเว็บเพื่อใช้งาน

- ☐ bmahrms เว็บไซต์สำหรับ Officer
- ☐ bmahrms-user เว็บไซต์ ระบบบริการเจ้าของข้อมูล
- ☐ bmahrms-admin ระบบ Admin
- ☐ bmahrms-checkin เว็บไซต์ ระบบลงเวลา
- ☐ bmahrms-publish เว็บไซต์ เผยแพร่ผลงาน
- ☐ bmahrms-candidate-register เว็บไซต์สำหรับรับสมัครสอบคัดเลือก
- ☐ bmahrms-qualifying-exam-cms เว็บไซต์ศูนย์ข้อมูลการสรรหาบุคคลของกรุงเทพมหานคร(CMS)
- ☐ bmahrms-manual ระบบคู่มือ
- ☐ bmahrms-faq เว็บไซต์ FAQ
- ☐ bmahrms-logs-n-backup Backup and Logs

โปรแกรม Backend ถูกเรียกใช้โดย Frontend อีกที

- ☐ bmahrms-report
- ☐ bmahrms-recruit
- ☐ bmahrms-insignia
- ☐ bmahrms-recruit-exam
- ☐ bmahrms-org-employee
- ☐ bmahrms-placement
- ☐ bmahrms-retirement
- ☐ bmahrms-report-v2
- ☐ bmahrms-probation
- ☐ bmahrms-command
- ☐ bmahrms-discipline
- ☐ bmahrms-evaluation
- ☐ bmahrms-leave
- ☐ bmahrms-org

- ☐ bmahrms-salary
- ☐ bmahrms-development
- ☐ bmahrms-kpi

1.10.3.1. ตัวแปรแวดล้อมของ BMA HRMS (Frontend)

docker/bmahrms/fe.env

แก้ตัวแปร VITE_CLIENTSECRET_KEYCLOAK KC_SERVICE_ACCOUNT_SECRET

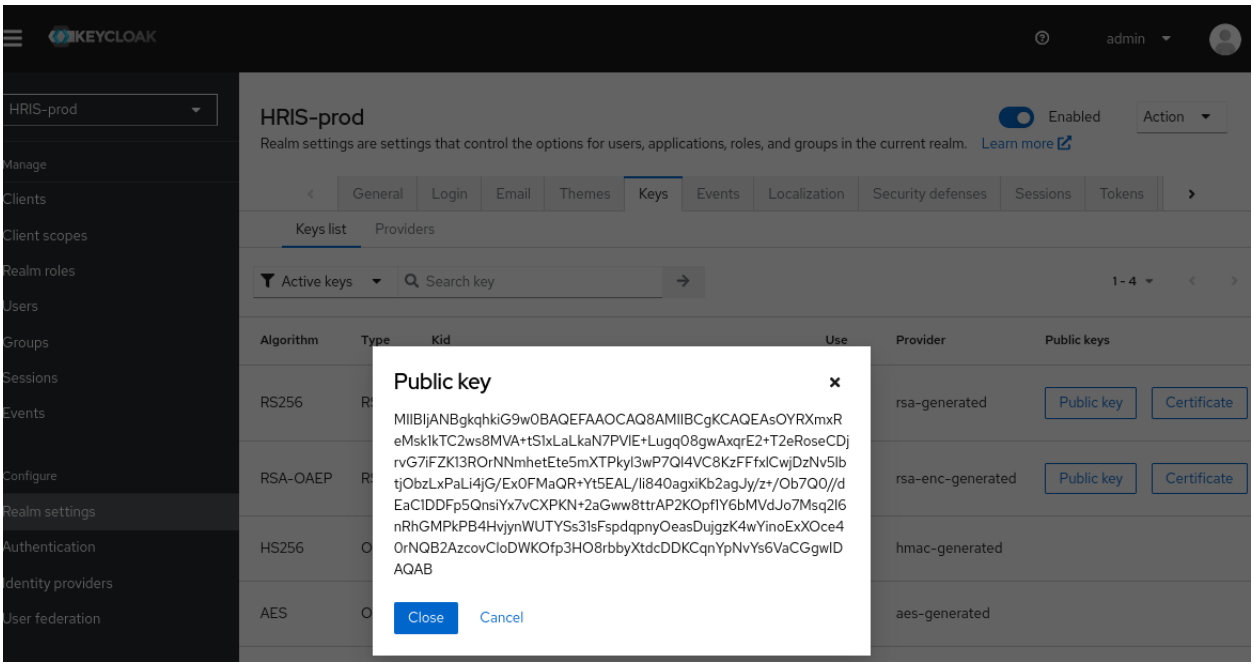
```
# Frontend Global
TZ=Asia/Bangkok
VITE_COMPETITIVE_EXAM_PANEL=https://bma-hrms-dashboard.bangkok.go.th/goto/KO0GpSu4z?orgId=1
VITE_QUALIFY_DISABLE_EMAM_PANEL=https://bma-hrms-dashboard.bangkok.go.th/goto/dQQzpIX4z?orgId=1
VITE_QUALIFY_EXAM_PANEL=https://bma-hrms-dashboard.bangkok.go.th/goto/cj1ZtIX4z?orgId=1
VITE_S3CLUSTER_PUBLIC_URL=https://bma-hrms-s3.bangkok.go.th/bma-ehr-fpt/organization/strueture/
VITE_REALM_KEYCLOAK=bma-ehr
VITE_URL_KEYCLOAK=https://bma-hrms-id.bangkok.go.th
VITE_CLIENTID_KEYCLOAK: "gettoken"
VITE_CLIENTSECRET_KEYCLOAK: put_client_secret_keycloak_here
VITE_API_REPORT_URL=https://bma-hrms.bangkok.go.th/api/v1/report-template
# เผยแพร่ผลงาน
VITE_API_PUBLISH_URL=https://bma-hrms-publish.bangkok.go.th


# Admin
VITE_API_URI_CONFIG=https://bma-hris.bangkok.go.th/api/v1
KC_URL=https://bma-hrms-id.bangkok.go.th
KC_REALM=bma-ehr
KC_SERVICE_ACCOUNT_CLIENT_ID=bma-ehr-dev
KC_SERVICE_ACCOUNT_SECRET=put_service_account_secret_here
MANAGEMENT_ROLE=storage_management
```

1.10.3.2. ตัวแปรแวดล้อมของ BMA HRMS (Backendend)

docker/bmahrms/be.env

แก้ค่าของ DB_PASSWORD,AUTH_PUBLIC_KEY, MINIO_KEY, MINIO_SECRET, STORAGE_SECRET,KC_SERVICE_ACCOUNT_SECRET,REDIS_HOST,KEYCLOAK_KEY,AUTH_PUBLIC_KEY, PUBLIC_KEY

ค่าของ public key ของ keycloak ให้สำเนาจากส่วนนี้



รูป 4-4 Keycloak Publickey

```
# Backend Global
TZ=Asia/Bangkok
ASPNETCORE_ENVIRONMENT=Development
APP_HOST=0.0.0.0
APP_PORT=80
HOST=0.0.0.0
PORT=80
DB_HOST=bmahrms-mysql
DB_PORT=3306
DB_USERNAME=root
DB_NAME=bma_ehr_organization_demo
DB_PASSWORD=DB_PASSWORD
AUTH_REALM_URL=https://bmahrms-id.bangkok.go.th/realms/bma-ehr
AUTH_PUBLIC_KEY=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkOrQxwoFeJE5TPXeS
GL8IRDYVEL1OzmBuywNAzsL9vdcDUrjRA+Ocfp1pwLxXbNPacIgVmP1G+Cf3Steyp5ZeYAGL3u/MRH
hUZA1tRbJ9sT+i/JzubmJCs9uF2pV9dLyT6kP7ZjKXmFBzHcXTS/GZMRtz8HZjB6ZzegFtV2oIrQq4+
```

zo4BlzIAI6vaZfKn/JrTjJEbti0RQcqWfIldE918SQvZHzi3Upgn+OBopSRCrF7z3a7FEjYI1Vny42
f5dBUcAmDDxOKqoJN09ArYchWkiDGvfpVarsclodptAGKwLuIDwDm9T9cnBUcBzWKNkwpQD3LvydO
FVJRBqHAOrfEQIDAQAB
AUTH_PREFERRED_MODE=online
API=https://bma-hrms.bangkok.go.th/api/v1/
SECRET_KEY=put_secret_key_here
MINIO_HOST=https://bmahrms-s3.bangkok.go.th
MINIO_PORT=9000
MINIO_ACCESS_KEY=MINIO_KEY
MINIO_SECRET_KEY=MINIO_SECRET
MINIO_BUCKET=bma-ehr-fpt
API_URL=https://bmahrms.bangkok.go.th/api/v1
STORAGE_URL=https://bmahrms-edm.bangkok.go.th/api
STORAGE_REALM_URL=https://edm-id.bangkok.go.th/realms/EDM
STORAGE_SECRET=put_storage_secret_here
KC_URL=https://bmahris-id.bangkok.go.th
KC_REALM=bma-ehr
KC_SERVICE_ACCOUNT_CLIENT_ID=bma-ehr-dev
KC_SERVICE_ACCOUNT_SECRET=put_account_secret_here
MANAGEMENT_ROLE=storage_management
REDIS_HOST=192.168.1.81
REDIS_PORT=6379

KEYCLOAK_URL=https://bma-hrms-id.bangkok.go.th/realms/bma-ehr
KEYCLOAK_KEY=CNnpzJkwbk7jvI5Lb6gP5_Ok0WpDJJ50Mn0aD55Bd5E
KEYCLOAK_REALM=bma-ehr

PUBLIC_KEY=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvYg0ZJvH6HgNOzyPp7PCvY
3bJwD9WdsNn6gZbuvIfqJQZ8iSH1t0p3fgODO/fqwcj9UFeh1bVFOSjuW+JpnPehROqzt81KNl9zLL
NXoN4LimReQHaMM3dU7DCbRylgVCouIDvObyjg8G+Cy5lZvFKWym/DPwGVpSdbvDZJ83qxq2dp7GJX
S8PhOvA+MB1K009/jW5pBTUwNArLjoFccr+gIYIiOJDg2rYyIF3fDkwyWkuxr6xRt10+BRJytselwy/
18kbDuJxVPaapdgTXI6wLzx7HWcDk30n5EvhJEumnIPpRst8gucqNYmB4MH+vsyoxV5WLuO3qmVRz
FbtAppRQIDAQAB
REALM_URL=https://bmahrms-id.bangkok.go.th/realms/bma-ehr
PREFERRED_MODE=online
PREFERRED_AUTH=online

```
ELASTICSEARCH_PROTOCOL=http
ELASTICSEARCH_HOST=bmahrms-elasticsearch
ELASTICSEARCH_PORT=9200
ELASTICSEARCH_INDEX=bma-ehr-log-index
```

1.10.3.3. compose.yaml ของ BMA HRMS

ใช้สำหรับ Backend และ Frontend **docker/bmahrms/compose.yaml**

มี Backend และ Frontend

```yaml
networks:
 hrms:
  external: true
services:
  # Start Frontend Section
  # เว็บไซต์สำหรับ Officer
 bmahrms:
  container_name: bmahrms
  image: docker.frappet.com/ehr/bma-ehr-app:latest
  restart: always
  env_file: "fe.env"
  ports:
  - "6021:80"
  networks:
  - hrms
  # เว็บไซต์สำหรับรับสมัครสอบคัดเลือก
 bmahrms-candidate-register:
  image: docker.frappet.com/ehr/bma-ehr-recruit-exam:latest
  container_name: bmahrms-candidate-register
  env_file: "fe.env"
  restart: always
  environment:
   VITE_API_URI_CONFIG: "https://bmahrms.bangkok.go.th/api"
   VITE_CLIENTID_KEYCLOAK: "bma-ehr-exam-vue3"
  ports:
  - "6022:80"
  networks:
```

```
    - hrms
    # เว็บไซต์ศูนย์ข้อมูลการสรรหาบุคคลของกรุงเทพมหานคร(CMS)
    # CMS ระบบสมัครสอบ
  bmahrms-qualifying-exam-cms:
    image: docker.frappet.com/demo/qualifying-exam-cms:latest
    container_name: bmahrms-qualifying-exam-cms
    restart: always
    env_file: "fe.env"
    ports:
      - "6023:80"
    environment:
      API_CMS_URL: "https://bmahrms.bangkok.go.th/api/v1/cms"
      API_QUALIFYING_URL: "https://bmahrms.bangkok.go.th/api/v1/cms"
      API_COMPETITIVE_URL: "https://bmahrms.bangkok.go.th/api/v1/recruit"
      PUBLIC_URL_REGISTER_QUALIFY_EXAM: "https://bmahrms-apply.bangkok.go.th"
    networks:
      - hrms
    # ระบบคู่มือ
  bmahrms-manual:
    image: docker.frappet.com/ehr/bma-ehr-manual:latest
    container_name: bmahrms-manual
    restart: always
    env_file: "fe.env"
    ports:
      - "6024:80"
    networks:
      - hrms
    # เว็บไซต์ ระบบบริการเจ้าของข้อมูล
  bmahrms-user:
    image: docker.frappet.com/ehr/bma-ehr-user:latest
    container_name: bmahrms-user
    restart: always
    env_file: "fe.env"
    ports:
      - "6025:8087"
    networks:
```

```
    - hrms
    # เว็บไซต์ ระบบลงเวลา
  bmahrms-checkin:
    image: docker.frappet.com/ehr/bma-ehr-checkin:latest
    container_name: bmahrms-checkin
    restart: always
    env_file: "fe.env"
    ports:
    - "6026:80"
    networks:
    - hrms
    # เว็บไซต์ เผยแพร่ผลงาน
  bmahrms-publish:
    image: docker.frappet.com/ehr/bma-ehr-publish:latest
    container_name: bmahrms-publish
    restart: always
    env_file: "fe.env"
    ports:
    - "6027:80"
    environment:
    VITE_API_URI_CONFIG: "https://bmahrms.bangkok.go.th/api/v1"
    VITE_API_PUBLISH_URL: "https://bmahrms.bangkok.go.th/api/v1/evaluation"
    networks:
    - hrms
    # เว็บไซต์ FAQ
  bmahrms-faq:
    image: docker.frappet.com/ehr/bma-ehr-faq:latest
    container_name: bmahrms-faq
    restart: always
    env_file: "fe.env"
    ports:
    - "6028:80"
    networks:
    - hrms
    # ระบบ Admin
  bmahrms-admin:
```

```
  container_name: bmahrms-admin
  image: docker.frappet.com/ehr/bma-ehr-admin:latest
  env_file: "fe.env"
  restart: always
  ports:
   - "6029:8086"
  environment:
   VITE_CLIENTID_KEYCLOAK: "HRIS_ADMIN"
   VITE_API_URI_CONFIG: "https://bma-hrms.bangkok.go.th/api/v1"
   KC_REALM: "bma-ehr"
   KC_SERVICE_ACCOUNT_CLIENT_ID: "bma-ehr-dev"
   KC_SERVICE_ACCOUNT_SECRET: "f2mp7Xj4nz6gbgITve9J7AHXZI8dRhOd"
   MANAGEMENT_ROLE: "storage_management"
  networks:
   - hrms
   # Backup and Logs
 bmahrms-logs-n-backup:
  container_name: bmahrms-logs-n-backup
  image: docker.frappet.com/ehr/bma-ehr-log-backup:latest
  restart: always
  env_file: "fe.env"
  ports:
   - 6030:3000
  environment:
   - KC_REALM_URL=https://bma-hrms-id.bangkok.go.th/realms/bma-ehr
   - ELASTICSEARCH_PROTOCOL=http
   - ELASTICSEARCH_HOST=192.168.1.40
   - ELASTICSEARCH_PORT=9200
   - ELASTICSEARCH_INDEX=bma-ehr-log-dotcom-index
   - APP_HOST=0.0.0.0
   - APP_PORT=3000
   - WINDMILL_URL=http://host.docker.internal:20001
   - WINDMILL_WORKSPACE=bma-ehr
   - WINDMILL_BACKUP_FLOW_PATH=f/flow/full_backup_s3_mysql
   - WINDMILL_RESTORE_FLOW_PATH=f/flow/full_restore_s3_mysql
   - WINDMILL_BACKUP_DELETE_SCRIPT_PATH=f/flow/delete_backup_s3_mysql
   - WINDMILL_BACKUP_LIST_SCRIPT_PATH=f/flow/list_backup_s3_mysql
```

```
    - WINDMILL_API_KEY=I843JmwwcRKouMnE1DkTBRjiKl6yzkXM
    - DB_HOST="bma-mysql-ehr"
    - DB_PORT=3306
    - DB_USERNAME=root
    - DB_PASSWORD=adminVM123
    - DB_LIST=bma_ehr,bma_ehr_development,bma_ehr_discipline,bma_ehr_evaluation,
bma_ehr_exam,bma_ehr_history,bma_ehr_kpi,bma_ehr_leave,bma_ehr_organization,b
ma_ehr_probation,bma_ehr_salary,bma_ehr_support,bma_recruit
    - MAIN_MINIO_USE_SSL=true
    - MAIN_MINIO_HOST=bma-hrms-s3.bangkok.go.th
    - MAIN_MINIO_PORT=443
    - MAIN_MINIO_ACCESS_KEY=put_minio_access_key_here
    - MAIN_MINIO_SECRET_KEY=put_minio_secret_key_here
    - MAIN_MINIO_BUCKET=bma-ehr-fpt
    - BACKUP_MINIO_USE_SSL=true
    - BACKUP_MINIO_HOST=bma-hrms-s3.bangkok.go.th
    - BACKUP_MINIO_PORT=443
    - BACKUP_MINIO_ACCESS_KEY=put_backup_minio_access_key_here
    - BACKUP_MINIO_SECRET_KEY=put_backup_minio_secret_key_here
    - BACKUP_MINIO_BUCKET=bma-ehr-fpt-backup-dotcom
  extra_hosts:
    - host.docker.internal:host-gateway
  networks:
    - hrms
  # End Frontend Section

  # Start Backend Section
bmahrms-report:
  image: docker.frappet.com/ehr/bma-ehr-report-service:latest
  container_name: bmahrms-report
  env_file: "be.env"
  restart: always
    # ports:
    #   - "7020:80"
  volumes:
    - ./wwwroot:/app/wwwroot
    - ./appsettings-report.json:/app/appsettings.json
```

```
    - ./appsettings-report.json:/app/appsettings.Development.json
  networks:
    - hrms

bmahrms-recruit:
  container_name: bmahrms-recruit
  image: docker.frappet.com/ehr/bma-ehr-recruit-service:latest
  restart: always
  env_file: "be.env"
    # ports:
    #   - "7021:80"
  volumes:
    - ./wwwroot:/app/wwwroot
    - ./appsettings-recruit.json:/app/appsettings.json
    - ./appsettings-recruit.json:/app/appsettings.Development.json
  networks:
    - hrms

bmahrms-insignia:
  image: docker.frappet.com/ehr/bma-ehr-insignia-service:latest
  container_name: bmahrms-insignia
  restart: always
  env_file: "be.env"
    # ports:
    #   - "7022:80"
  volumes:
    - ./wwwroot:/app/wwwroot
    - ./appsettings-combine.json:/app/appsettings.json
    - ./appsettings-combine.json:/app/appsettings.Development.json
  networks:
    - hrms

bmahrms-recruit-exam:
  image: docker.frappet.com/ehr/bma-ehr-recruit-exam-service:latest
  container_name: bmahrms-recruit-exam
  restart: always
```

```
    env_file: "be.env"
       # ports:
       #   - "7023:80"
    volumes:
     - ./wwwroot:/app/wwwroot
     - ./appsettings-recruit-exam.json:/app/appsettings.json
     - ./appsettings-recruit-exam.json:/app/appsettings.Development.json
    networks:
     - hrms

  bmahrms-org-employee:
    image: docker.frappet.com/ehr/bma-ehr-org-employee-service:latest
    container_name: bmahrms-org-employee
    restart: always
    env_file: "be.env"
       # ports:
       #   - "7024:80"
    volumes:
     - ./wwwroot:/app/wwwroot
     - ./appsettings-combine.json:/app/appsettings.json
     - ./appsettings-combine.json:/app/appsettings.Development.json
    networks:
     - hrms

  bmahrms-placement:
    image: docker.frappet.com/ehr/bma-ehr-placement-service:latest
    container_name: bmahrms-placement
    restart: always
    env_file: "be.env"
       # ports:
       #   - "7025:80"
    volumes:
     - ./wwwroot:/app/wwwroot
     - ./appsettings-combine.json:/app/appsettings.json
     - ./appsettings-combine.json:/app/appsettings.Development.json
    networks:
```

```
    - hrms

  bmahrms-retirement:
    image: docker.frappet.com/ehr/bma-ehr-retirement-service:latest
    container_name: bmahrms-retirement
    restart: always
    env_file: "be.env"
      # ports:
      #   - "7026:80"
    volumes:
    - ./wwwroot:/app/wwwroot
    - ./appsettings-combine.json:/app/appsettings.json
    - ./appsettings-combine.json:/app/appsettings.Development.json
    networks:
    - hrms

  bmahrms-report-v2:
    image: docker.frappet.com/ehr/bma-ehr-report-v2-service:latest
    container_name: bmahrms-report-v2
    restart: always
    env_file: "be.env"
      # ports:
      #   - "7027:80"
    volumes:
    - ./wwwroot:/app/wwwroot
    - ./appsettings-report-v2.json:/app/appsettings.json
    - ./appsettings-report-v2.json:/app/appsettings.Development.json
    networks:
    - hrms

  bmahrms-probation:
    image: docker.frappet.com/ehr/bma-ehr-node-service:latest
    container_name: bmahrms-probation
    restart: always
    env_file: "be.env"
      # ports:
```

```
    #   - "7028:80"
  environment:
   DB_NAME: "bma_ehr_probation"
  networks:
   - hrms

 bmahrms-command:
  image: docker.frappet.com/ehr/bma-ehr-command-service:latest
  container_name: bmahrms-command
  restart: always
  env_file: "be.env"
    # ports:
    #   - "7029:80"
  volumes:
   - ./wwwroot:/app/wwwroot
   - ./appsettings-combine.json:/app/appsettings.json
   - ./appsettings-combine.json:/app/appsettings.Development.json
  networks:
   - hrms

 bmahrms-discipline:
  image: docker.frappet.com/ehr/bma-ehr-discipline-service:latest
  container_name: bmahrms-discipline
  restart: always
    # env_file: "be.env"
    # ports:
    #   - "7030:80"
  volumes:
   - ./wwwroot:/app/wwwroot
   - ./appsettings-combine.json:/app/appsettings.json
   - ./appsettings-combine.json:/app/appsettings.Development.json
  networks:
   - hrms

 bmahrms-evaluation:
  image: docker.frappet.com/ehr/bma-ehr-evaluation-service:latest
```

```
  container_name: bmahrms-evaluation
  restart: always
  env_file: "be.env"
     # ports:
     #   - "7031:80"
  environment:
   DB_NAME: "bma_ehr_evaluation"
  networks:
   - hrms

 bmahrms-leave:
  image: docker.frappet.com/ehr/bma-ehr-leave-service:latest
  container_name: bmahrms-leave
  restart: always
  env_file: "be.env"
     # ports:
     #   - "7032:80"
  volumes:
   - ./wwwroot:/app/wwwroot
   - ./appsettings-combine.json:/app/appsettings.json
   - ./appsettings-combine.json:/app/appsettings.Development.json
  networks:
   - hrms

 bmahrms-org:
  image: docker.frappet.com/ehr/bma-ehr-org-service:latest
  container_name: bmahrms-org
  restart: always
  env_file: "be.env"
     # ports:
    #   - "7033:80"
  environment:
   DB_NAME: "bma_ehr_organization"
  networks:
   - hrms
```

```
bmahrms-salary:
 image: docker.frappet.com/ehr/bma-ehr-salary-service:latest
 container_name: bmahrms-salary
 restart: always
 env_file:"be.env"
    # ports:
    #   -"7034:80"
 environment:
  DB_NAME:"bma_ehr_salary"
 networks:
  - hrms

bmahrms-development:
 image: docker.frappet.com/ehr/bma-ehr-development-service:latest
 container_name: bmahrms-development
 restart: always
 env_file:"be.env"
    # ports:
    #   -"7035:80"
 environment:
  DB_NAME:"bma_ehr_development"
 networks:
  - hrms

bmahrms-kpi:
 image: docker.frappet.com/ehr/bma-ehr-kpi-service:latest
 container_name: bmahrms-kpi
 restart: always
 env_file:"be.env"
    # ports:
    #   -"7036:80"
 environment:
  DB_NAME:"bma_ehr_kpi"
 networks:
  - hrms
  # End Backend Section
```

1.10.3.4. Route ของ BMA-HRMS frontend

โดเมนสำหรับหน้าโปรแกรมจะมีดังนี้ bmahrms.bangkok.go.th,bmahrms-user.bangkok.go.th,bmahrms-admin. bangkok. go. th,bmahrms-checkin. bangkok. go. th,bmahrms-recruiting. bangkok. go. th,bmahrms-exam.bangkok.go.th จะต้องสร้าง route ให้ตรงกับหน้าเวปที่ใช้งาน ใช้แค่ methods GET

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms",
 "methods": ["GET"],
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms:80": 1
     }
  }
}'

curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-user",
 "methods": ["GET"],
 "host": "bma-hrms-user.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-user:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-admin",
```

```
 "methods": ["GET"],
 "host": "bma-hrms-admin.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-admin:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-checkin",
 "methods": ["GET"],
 "host": "bma-hrms-checkin.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-checkin:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bma-hrms-recruiting",
 "methods": ["GET"],
 "host": "bmahrms-recruiting.bangkok.go.th",
 "uri": "/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-recruiting:80": 1
     }
   }
}'
```

Faq and Manual เป็น Micro Frontend เป็นพาร์ทย่อยของโดเมนหลัก - Faq: /faq - Manual: /manual

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-faq",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/faq/*",
 "plugins": {
  "proxy-rewrite": {
   "regex_uri": ["^/faq/(.*)","/$1"]
     }
   },
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-faq:80": 1
     }
   }
}'
```

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "bmahrms-manual",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/manual/*",
 "plugins": {
  "proxy-rewrite": {
   "regex_uri": ["^/manual/(.*)","/$1"]
     }
   },
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-manual:80": 1
     }
```

```
  }
}'
```

1.10.3.5. Route ของ BMA-HRMS backend

Route ของ API จะอยู่ในรูปแบบ domain/api/v1/*

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-org",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/org/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-org:80": 1
    }
  }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-recruit",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/recruit/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-recruit:80": 1
    }
  }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-report",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/report/*",
 "upstream": {
```

```
  "type": "roundrobin",
  "nodes": {
   "bmahrms-report:80": 1
     }
  }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-insignia",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/insignia/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-insignia:80": 1
     }
  }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-recruit-exam-cms",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/cms/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-recruit-exam:80": 1
     }
  }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-recruit-exam-candidate",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/candidate/*",
```

```
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-recruit-exam:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-org-employee",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/organization-employee/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-org-employee:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-placement",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/placement/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-placement:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-retirement",
 "host": "bma-hrms.bangkok.go.th",
```

```
 "uri": "/api/v1/retirement/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-retirement:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-report-v2",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v2/report/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-report-v2:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-probation",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/probation/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-probation:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-command",
```

```
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/order/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-command:80": 1
     }
   }
}'

curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-command-message",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/message/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-command:80": 1
     }
   }
}'

curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-discipline",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/discipline/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-discipline:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
```

```
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-evaluation",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/evaluation/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-evaluation:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-leave",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/leave/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-leave:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-salary",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/salary/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-salary:80": 1
     }
   }
}'
```

```
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-development",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/development/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-development:80": 1
     }
   }
}'
curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-kpi",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/kpi/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-kpi:80": 1
     }
   }
}'

curl "http://127.0.0.1:9180/apisix/admin/routes" \
-H "X-API-KEY: put_admin_api_key_here" -X PUT -d '
{
 "id": "hrms-backend-metadata",
 "host": "bma-hrms.bangkok.go.th",
 "uri": "/api/v1/metadata/*",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
   "bmahrms-metadata:80": 1
     }
```

```
  }
}'
```

1.11. บทสรุป

ในยุคดิจิทัลที่เทคโนโลยีเปลี่ยนแปลงอย่างรวดเร็ว การออกแบบระบบซอฟต์แวร์ให้ตอบสนองความต้องการที่ซับซ้อนและเปลี่ยนแปลงได้ตลอดเวลาเป็นสิ่งจำเป็น สถาปัตยกรรม Microservice เป็นหนึ่งในแนวทางที่ได้รับการยอมรับอย่างกว้างขวาง เนื่องจากมีจุดเด่นด้านความยืดหยุ่น การปรับขยาย และการแยกส่วนการทำงานของระบบให้เป็นอิสระต่อกัน การนำเทคโนโลยีทันสมัยอย่าง Docker, RabbitMQ, Keycloak และ MinIO มาร่วมใช้งานใน Microservice ทำให้ระบบมีประสิทธิภาพยิ่งขึ้น ทั้งในด้านการพัฒนา การจัดการ และการรองรับการใช้งานในระดับสูง

การผสมผสานเทคโนโลยีอย่าง Docker, RabbitMQ, Keycloak และ MinIO ช่วยยกระดับความสามารถของระบบให้สอดคล้องกับความต้องการทางธุรกิจในยุคที่การแข่งขันสูง สถาปัตยกรรมนี้ไม่เพียงตอบสนองต่อความซับซ้อนของการพัฒนา แต่ยังช่วยเพิ่มความเร็ว ความเสถียร และความปลอดภัยของระบบ ทำให้ Microservice เป็นทางเลือกที่ทันสมัยและเหมาะสมสำหรับการพัฒนาระบบในยุคปัจจุบัน