MASTER DOCKER: FROM ZERO TO PRODUCTION

Docker

พื้นฐานสู่การใช้งานจริง

คู่มือ Docker ฉบับสมบูรณ์ภาษาไทย

ไม่มีพื้นฐาน Container ก็เริ่มได้

ตั้งแต่ติดตั้ง Docker ครั้งแรก
จนรัน Application ได้จริงบนเครื่องของคุณ

ดร.อาณัติ รัตนถิรกุล
Tech Educator & Digital Learning Expert

Vol.1

SYS ADMIN

Docker พื้นฐานสู่การใช้งานจริง

ชื่อชุด: Master Docker Series

พิมพ์ครั้งที่ 1 : พฤษภาคม 2569

โดย ดร. อาณัติ รัตนถิรกุล

ราคา 349.- บาท

พิสูจน์อักษร : ธนียา รัตนถิรกุล

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

Docker พื้นฐานสู่การใช้งานจริง.-- กรุงเทพฯ : ซิสแอดมินโนว์เลดจ์, 2569.

248 หน้า.-- (Master Docker Series).

1. โปรแกรมคอมพิวเตอร์. 2. เทคโนโลยีสารสนเทศ. I. ชื่อเรื่อง.

005.3

ISBN 978-616-631-940-8

จัดทำโดย

ดร.อาณัติ รัตนถิรกุล

จัดจำหน่ายโดย

ซิสแอดมิน โนว์เลดจ์

เลขที่ 18 ซอยสุเหร่าหัวหมากน้อย แขวงหัวหมาก เขตบางกะปิ

กรุงเทพฯ 10240

โทรศัพท์ 084-1629068 อีเมล book@sysadmin.in.th

Docker พื้นฐานสู่การใช้งานจริง

คำนำ

ในยุคที่ซอฟต์แวร์ต้องพัฒนาและส่งมอบอย่างรวดเร็ว ความสามารถในการทำให้ระบบ “ทำงานได้เหมือนกันทุกที่” กลายเป็นปัจจัยชี้วัดความสำเร็จของทั้งนักพัฒนาและองค์กร ปัญหาคลาสสิกอย่าง *“ทำงานบนเครื่องเราได้ แต่เครื่องอื่นพัง”* คืออุปสรรคที่ทำให้เสียเวลาและต้นทุนโดยไม่จำเป็น **Docker** เข้ามาแก้โจทย์นี้ด้วยแนวคิด Containerization ที่ช่วยแพ็กแอปพลิเคชันพร้อมสภาพแวดล้อมให้พร้อมใช้งานทันที ไม่ว่าคุณจะอยู่ในขั้นตอนพัฒนา ทดสอบ หรือใช้งานจริง หนังสือเล่มนี้จึงถูกออกแบบมาเพื่อช่วยให้คุณเข้าใจ Docker อย่างเป็นระบบ และเริ่มต้นได้อย่างมั่นใจตั้งแต่ศูนย์

หนังสือ “Docker พื้นฐานสู่การใช้งานจริง (Docker Basics)” เล่มนี้เน้นการเรียนรู้แบบลงมือทำ (Hands-on) ควบคู่กับการอธิบายแนวคิดที่จำเป็น คุณจะได้เรียนตั้งแต่ภาพรวม DevOps โครงสร้างการทำงานของ Docker ไปจนถึงการใช้งานคำสั่งจริง การสร้าง Image การจัดการ Container รวมถึงการเชื่อมต่อระบบด้วย Volume และ Network และปิดท้ายด้วย Mini Project ที่คุณสามารถสร้าง Web Server ด้วยตัวเองได้จริง เนื้อหาทั้งหมดถูกจัดลำดับจากง่ายไปยาก เพื่อให้คุณเรียนรู้ได้อย่างต่อเนื่องและนำไปประยุกต์ใช้ได้ทันทีในงานจริง

เป้าหมายของหนังสือเล่มนี้ไม่ใช่เพียงให้คุณ “รู้จัก Docker” แต่คือทำให้คุณ “ใช้งาน Docker ได้จริง” และพร้อมต่อยอดสู่ระดับที่สูงขึ้น หากคุณกำลังมองหาทักษะที่จะช่วยยกระดับงานพัฒนา ระบบ และอาชีพของคุณ หนังสือเล่มนี้คือจุดเริ่มต้นที่เหมาะสมที่สุด ขอเพียงคุณเปิดใจและลงมือทำไปพร้อมกัน คุณจะค้นพบว่า Docker ไม่ได้ยากอย่างที่คิด และคุณสามารถก้าวสู่โลก DevOps ได้อย่างมั่นใจ

อาณัติ รัตนถิรกุล

SYSADMIN KNOWLEDGE

วิธีใช้หนังสือเล่มนี้ให้ได้ผลลัพธ์สูงสุด

เพื่อให้คุณได้ประโยชน์จากหนังสือเล่มนี้อย่างเต็มที่ แนะนำให้เรียนรู้ตามแนวทางนี้

อ่านเนื้อหาไปพร้อมกับ “ลงมือทำ” ทุกบท

ทดลองแก้ไขและปรับค่าต่าง ๆ ด้วยตัวเอง

ไม่ต้องกลัว Error เพราะทุกปัญหาคือการเรียนรู้

ทำ Lab ให้ครบ โดยเฉพาะ Mini Project ในบทท้าย

💡 **เคล็ดลับ:** อย่าเพียงแค่อ่านผ่าน ให้เปิดเครื่องแล้วลองทำทันที คุณจะเข้าใจ Docker ได้เร็วกว่าที่คิด

หนังสือเล่มนี้เหมาะกับใคร

1. นักศึกษา IT / Computer Science
2. โปรแกรมเมอร์ที่ต้องการเข้าใจ Deployment
3. System Administrator ที่ต้องการเรียนรู้ Container
4. DevOps Beginner ที่อยากเริ่มต้นอย่างถูกต้อง
5. ผู้ที่ต้องการเพิ่มทักษะด้าน Cloud และ Infrastructure

เป้าหมายของหนังสือเล่มนี้

เมื่อคุณเรียนจบเล่มนี้ คุณจะสามารถ

1. เข้าใจแนวคิด Containerization อย่างแท้จริง
2. ใช้งาน Docker CLI ได้อย่างมั่นใจ

3. สร้างและบริหารจัดการ Container ได้ด้วยตัวเอง
4. ออกแบบ Environment สำหรับพัฒนาแอปได้
5. พร้อมต่อยอดไปสู่ Docker Compose และระบบระดับ Production ในเล่มถัดไป

ก้าวต่อไปของคุณ

หนังสือเล่มนี้เป็นเพียง “ก้าวแรก” ของการเดินทาง
หลังจากนี้ คุณสามารถต่อยอดไปยัง
เล่ม 2 Docker ขั้นสูงและ Docker Compose สำหรับงานจริง
เล่ม 3 Docker สำหรับ Production และ DevOps Integration
ซึ่งจะพาคุณไปสู่ระดับการใช้งานจริงในองค์กร และการ Deploy ระบบแบบมืออาชีพ

ท้ายที่สุดนี้ ผู้เขียนหวังเป็นอย่างยิ่งว่า หนังสือเล่มนี้จะไม่ใช่แค่ “หนังสืออ่าน”
แต่จะเป็น “เครื่องมือ” ที่ช่วยให้คุณลงมือสร้างระบบจริงได้ด้วยตัวเอง

Docker ไม่ใช่เรื่องยาก ถ้าคุณเริ่มต้นอย่างถูกวิธี และลงมือทำจริง

ขอให้คุณสนุกกับการเรียนรู้ และเติบโตไปพร้อมกับโลกของ DevOps

สารบัญ

เรื่อง	หน้า

เรื่อง **หน้า**

บทที่ 1

รู้จัก Docker และ DevOps

1.1 พัฒนาการของการพัฒนาแอปพลิเคชัน

1.2 DevOps คืออะไร และมีประโยชน์อย่างไร

1.3 ปัญหาของการ Deploy แอปแบบเดิม

1.4 การถือกำเนิดของ Docker และแนวคิด Containerization

1.5 ข้อแตกต่างของ Docker กับ Virtual Machine

1.6 โครงสร้างพื้นฐานของ Docker

1.7 Use Case ของ Docker ในองค์กรจริง

ในยุคดิจิทัลที่ระบบสารสนเทศกลายเป็นหัวใจสำคัญขององค์กร ความสามารถในการพัฒนา ทดสอบ และนำแอปพลิเคชันขึ้นใช้งานจริงได้อย่างรวดเร็วและมีเสถียรภาพ คือปัจจัยที่กำหนดความสามารถในการแข่งขันขององค์กรโดยตรง

บทนี้จะพาคุณย้อนดูพัฒนาการของการพัฒนาแอปพลิเคชัน ตั้งแต่ยุคดั้งเดิมจนถึงแนวคิดสมัยใหม่ พร้อมอธิบายว่าทำไม DevOps และ Docker จึงกลายเป็นเทคโนโลยีหลักที่ขาดไม่ได้ในโลก IT ปัจจุบัน

บทนี้เน้น **แนวคิดและภาพรวม (Conceptual)** เพื่อสร้างรากฐานก่อนลงมือปฏิบัติในบทถัดไป อ่านให้เข้าใจ "ทำไม" ก่อน "อย่างไร" เสมอ

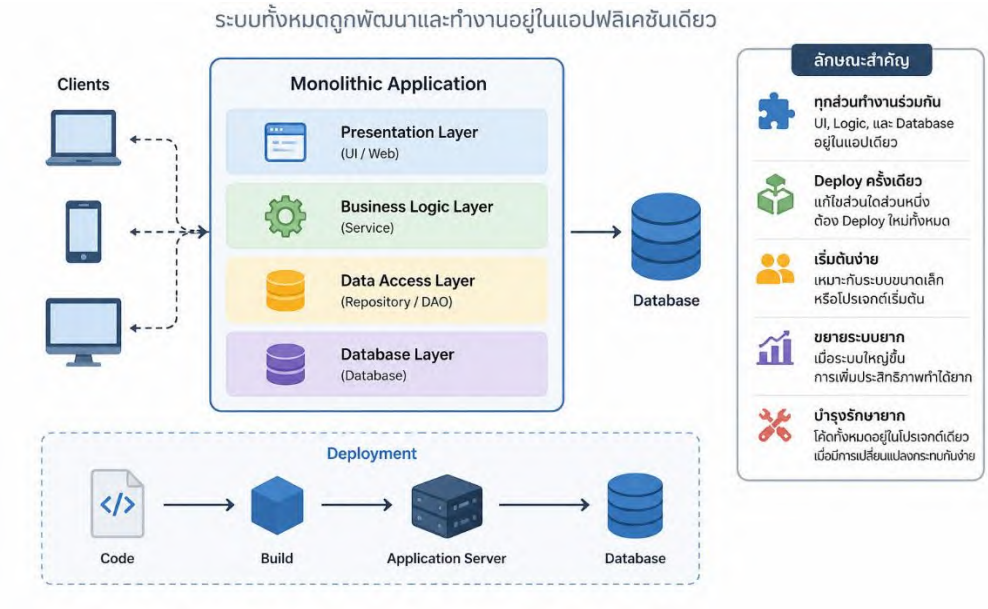
1.1 พัฒนาการของการพัฒนาแอปพลิเคชัน

การพัฒนาแอปพลิเคชันมีการเปลี่ยนแปลงอย่างต่อเนื่องตามความต้องการของธุรกิจและเทคโนโลยีที่ก้าวหน้า จากระบบที่มีโครงสร้างเรียบง่ายไปสู่ระบบที่มีความยืดหยุ่นและสามารถขยายตัวได้สูง

ปัจจุบันเทคโนโลยี Container และ Docker ได้กลายเป็นองค์ประกอบสำคัญของการพัฒนาและการปรับใช้แอปพลิเคชันสมัยใหม่ โดยสามารถอธิบายพัฒนาการของรูปแบบการพัฒนาแอปพลิเคชันได้เป็น 5 ยุคสำคัญ ดังนี้

1. ยุคโมโนลิทิก (Monolithic)

แนวคิด: ระบบทั้งหมดอยู่ในโปรแกรมเดียว



ยุคโมโนลิทิก (Monolithic Architecture) เป็นรูปแบบการพัฒนาแอปพลิเคชันในยุคเริ่มต้น โดยระบบทั้งหมดจะถูกรวมอยู่ภายในโปรแกรมเดียว (Single Application) ประกอบด้วยส่วนแสดงผล (User Interface), ตรรกะทางธุรกิจ (Business Logic) และการเชื่อมต่อฐานข้อมูล (Database Access) อยู่ภายในโค้ดชุดเดียวกัน

ข้อดี

- โครงสร้างเข้าใจง่าย เหมาะกับทีมเล็ก
- เริ่มต้นพัฒนาได้เร็ว
- Deploy ในขั้นตอนเดียว

ข้อจำกัด

- แก้ส่วนเล็ก ๆ ต้อง Deploy ทั้งระบบใหม่
- ขยายระบบ (Scale) เฉพาะส่วนทำไม่ได้
- ทีมใหญ่ขึ้น ⟶ conflict ในโค้ดบ่อยมาก

โครงสร้างโดยทั่วไปการพัฒนาแอปพลิเคชันในยุคยุคโมโนลิทิก

แม้จะเป็นก้อนเดียว แต่ภายในมักแบ่งเป็น 3 ชั้น (Logical Layer)

1. Presentation Layer ส่วนติดต่อผู้ใช้ (Web / Desktop UI)
2. Business Logic Layer กฎการทำงานหลักของระบบ
3. Data Access Layer ติดต่อฐานข้อมูล (เช่น MySQL, PostgreSQL)

ทั้งหมดอยู่ใน Codebase เดียวกัน และรันบน Application Server เดียว เช่น

- Web Interface
- Business Logic

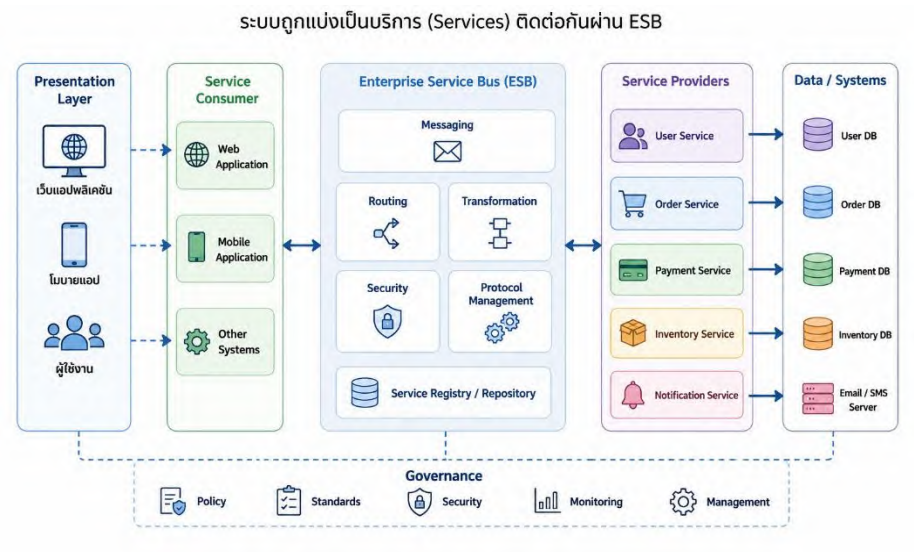
- Database Access

ทั้งหมดถูกพัฒนาอยู่ในโปรแกรมเดียวกันและทำงานร่วมกันภายในเซิร์ฟเวอร์เดียว

⚠ **ข้อควรระวัง:** Monolithic ไม่ได้ "แย่" เสมอ สำหรับโปรเจกต์เล็กหรือ MVP (Minimum Viable Product) ยังคงเป็นทางเลือกที่เหมาะสม อย่า over-engineer ตั้งแต่เริ่มต้น

2. ยุค SOA (Service-Oriented Architecture)

แนวคิด: แยกระบบเป็นบริการ (Services) สื่อสารผ่าน ESB



ยุค SOA (Service-Oriented Architecture) เกิดขึ้นเพื่อลดข้อจำกัดของ Monolithic โดยแยกระบบออกเป็นบริการ (Service) แต่ละส่วน ซึ่งสามารถเรียกใช้งานกันผ่านเครือข่าย โดยใช้มาตรฐานกลาง เช่น SOAP หรือ REST

ข้อดี

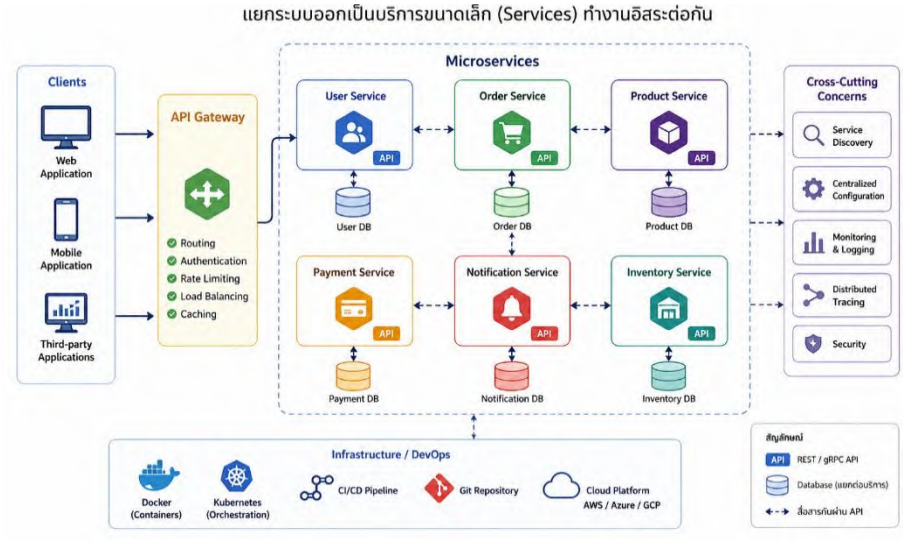
- ระบบถูกแบ่งออกเป็น Service ตามหน้าที่
- นำ Service กลับมาใช้ซ้ำ (Reuse) ได้
- เหมาะกับองค์กรขนาดใหญ่ที่มีหลายระบบเชื่อมต่อกัน

ข้อจำกัด

- ESB กลายเป็น Single Point of Failure
- การบริหารจัดการ Service ต้องพึ่งพาระบบส่วนกลาง
- Configuration ซับซ้อนมาก
- Deploy ยังไม่คล่องตัวพอ

3. ยุคไมโครเซอร์วิส (Microservices)

แนวคิด: บริการย่อยอิสระ ไม่ขึ้นกัน



ยุคไมโครเซอร์วิส (Microservices Architecture) เป็นการพัฒนาต่อยอดจาก SOA โดยแต่ละ Service มีขนาดเล็กลงมาก และ **เป็นอิสระจากกันอย่างสมบูรณ์** ทั้งการพัฒนา, Deploy, Scaling, และแม้แต่เทคโนโลยีที่ใช้ โดยเน้นการแยกระบบออกเป็นบริการย่อยขนาดเล็ก (Small and Independent Services) ซึ่งแต่ละบริการสามารถ

- พัฒนา แก้ไข และ Deploy ได้อย่างอิสระ
- ใช้ภาษาและเทคโนโลยีที่แตกต่างกันได้
- ขยายระบบเฉพาะส่วนที่จำเป็น

แนวคิดนี้ทำให้การพัฒนาแอปพลิเคชันมีความคล่องตัวสูง แต่ก็เพิ่มความท้าทายด้านการ Deploy และบริหารจัดการระบบจำนวนมาก

ข้อดี

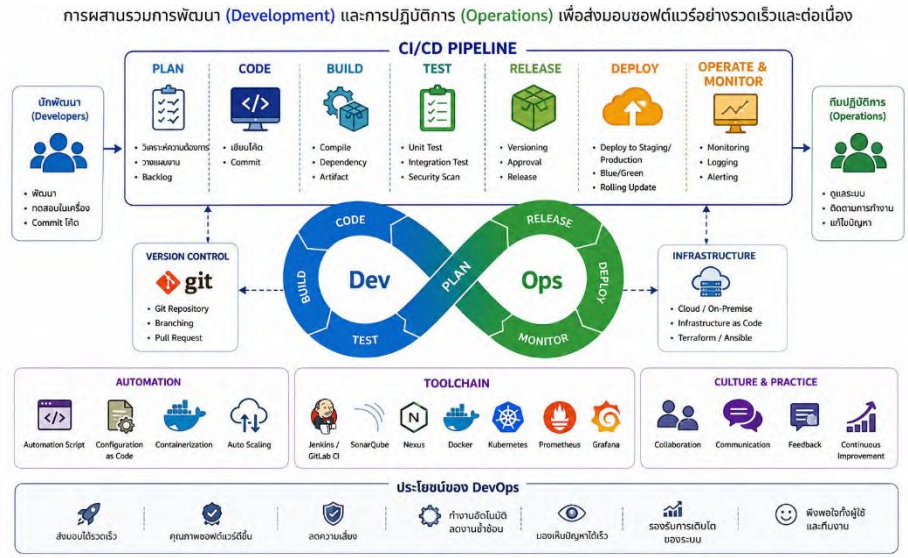
- Deploy เฉพาะ Service ที่เปลี่ยนได้
- Scale เฉพาะส่วนที่โหลดสูงได้
- แต่ละทีมทำงาน Service ของตัวเองอิสระ

ข้อจำกัด

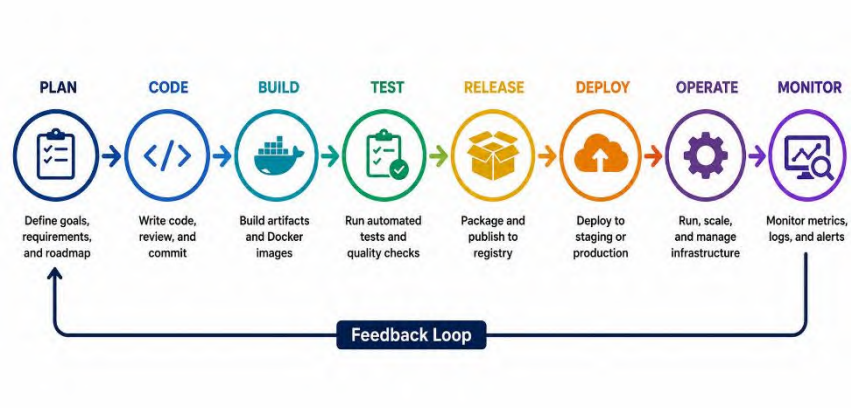
- จำนวน Service มากขึ้น ⟶ บริหารยากขึ้นมาก
- นี่คือจุดที่ **Docker เข้ามามีบทบาทสำคัญ** ในฐานะเครื่องมือ Deploy ที่สม่ำเสมอ

4. ยุคเดฟออปส์ (DevOps)

แนวคิด: ทลายกำแพงระหว่าง Dev และ Ops



ก่อนหน้านี้ ทีมพัฒนา (Dev) และทีมปฏิบัติการ (Ops) ทำงานแยกกันเหมือน Silo - Dev อยากเปลี่ยนบ่อย ๆ แต่ Ops อยากให้ระบบนิ่ง ความขัดแย้งนี้ทำให้การ Deploy ช้าลงอย่างมาก



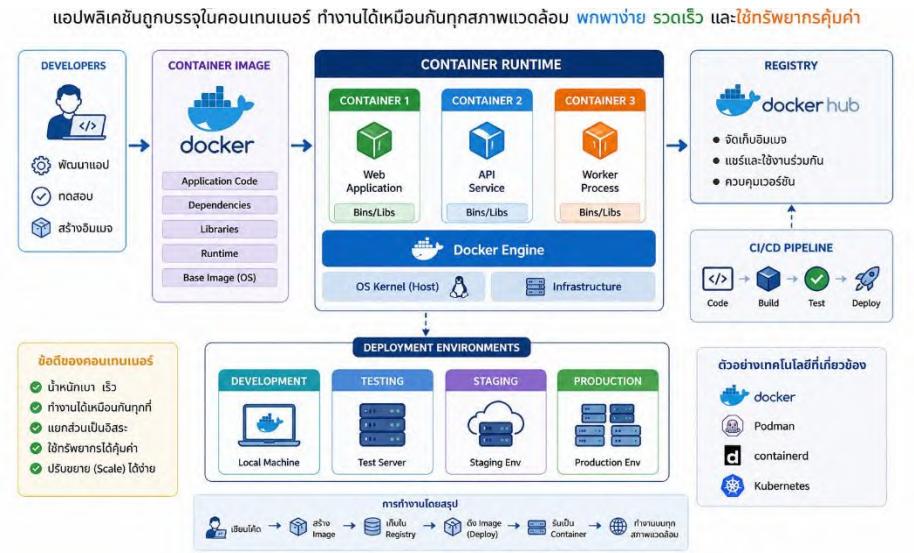
วงจรนี้เรียกว่า **CI/CD Pipeline** (Continuous Integration / Continuous Deployment)

ประโยชน์ที่ได้

- ความเร็ว (Speed): Deploy บ่อยขึ้น ได้ฟีเจอร์ใหม่เร็วขึ้น
- ความเสถียร (Reliability): Automation ลด Human Error
- การทำงานร่วมกัน (Collaboration): Dev + Ops เข้าใจกัน
- ความปลอดภัย (DevSecOps): ตรวจ Security ตั้งแต่ขั้นเขียนโค้ด
- กู้คืนเร็ว (Fast Recovery): Rollback ได้ทันที

5. ยุคคอนเทนเนอร์ (Containerization)

แนวคิด: แอปพลิเคชัน + ทุก Dependency = Container หน่วยเดียว



ยุคคอนเทนเนอร์ (Containerization Era) เป็นยุคที่เทคโนโลยี Docker เข้ามามีบทบาทสำคัญ โดยแนวคิด Containerization คือการ

บรรจุแอปพลิเคชันพร้อมไลบรารีและ Dependency ทั้งหมดไว้ในหน่วยเดียวที่เรียกว่า “คอนเทนเนอร์” (Container)

💡 **เคล็ดลับ:** ลองนึกภาพว่า Container คือ "กล่องที่บรรจุทุกอย่างที่แอปต้องการ" เปิดกล่องที่ไหน แอปก็ทำงานเหมือนกันทุกที่

ลักษณะเด่นของยุคคอนเทนเนอร์ ได้แก่

- ลดปัญหา “ทำงานบนเครื่องหนึ่ง แต่อีกเครื่องไม่ทำงาน”
- Deploy ได้รวดเร็วและทำซ้ำได้
- เหมาะกับระบบ Microservices และ Cloud

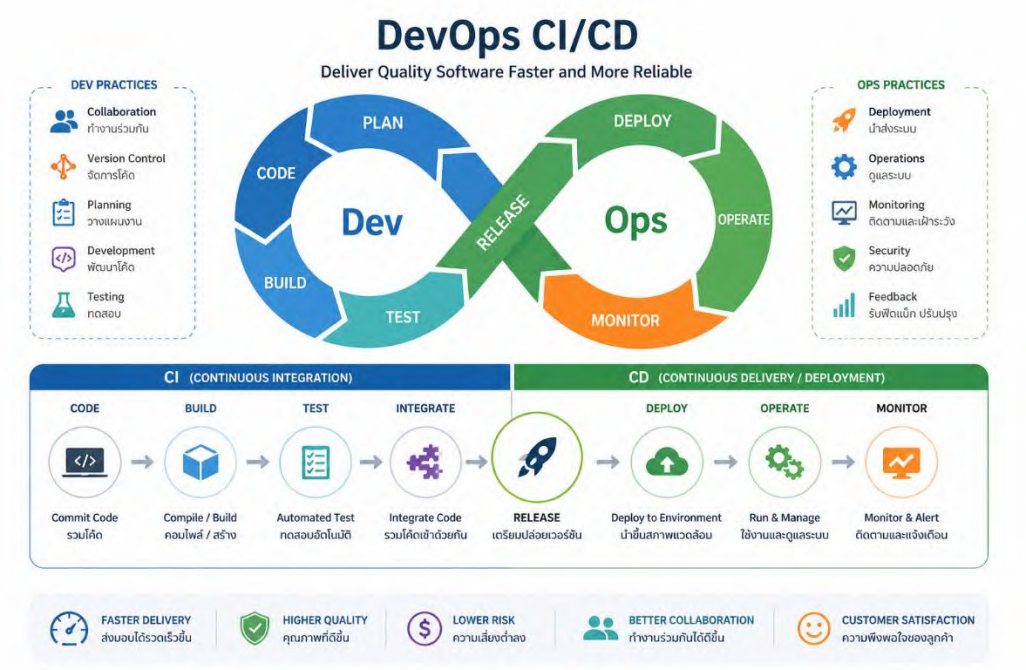
Docker จึงกลายเป็นเครื่องมือหลักที่เชื่อมโยงแนวคิด DevOps เข้ากับการใช้งานจริงในองค์กร

เมื่อระบบแตกเป็นหลายบริการ การทำงานแบบแยกส่วนระหว่างทีมพัฒนา (Development) และทีมปฏิบัติการ (Operations) กลายเป็นอุปสรรคสำคัญ ส่งผลให้เกิดแนวคิด DevOps เพื่อเชื่อมช่องว่างดังกล่าว

กรณีศึกษา

องค์กรหนึ่งพัฒนาเว็บแอปสำเร็จแล้ว แต่ใช้เวลาหลายสัปดาห์กว่าจะ Deploy ขึ้น Production เนื่องจาก

- สภาพแวดล้อม Dev กับ Server จริงไม่เหมือนกัน
- ต้องตั้งค่าระบบใหม่ทุกครั้ง
- เกิดข้อผิดพลาดซ้ำ ๆ ระหว่างการ Deploy



1.2 DevOps คืออะไร และมีประโยชน์อย่างไร

ความหมายของ DevOps

DevOps (มาจากคำว่า **Development** + **Operations**) คือ แนวทางในการทำงานที่รวมทีมพัฒนาซอฟต์แวร์ (Dev) และทีมดูแลระบบ (Ops) เข้าด้วยกัน เพื่อให้การพัฒนา ขยายผล และแก้ไขซอฟต์แวร์เป็นไปอย่างรวดเร็วและมีคุณภาพสูงขึ้น

โดยปกติในอดีต สองทีมนี้มักจะแยกจากกัน (Silos) ทำให้เกิดความล่าช้าในการส่งมอบงาน แต่ DevOps ใช้เครื่องมือและกระบวนการอัตโนมัติเข้ามาช่วยทลายกำแพงนี้

📌 **สาระสำคัญ:** DevOps ไม่ใช่แค่เครื่องมือ มันคือ **วัฒนธรรม + กระบวนการ + เครื่องมือ** ที่ต้องทำงานสอดประสานกัน

หัวใจสำคัญของ DevOps วงจร CI/CD

กระบวนการของ DevOps มักถูกนำเสนอด้วยสัญลักษณ์ "Infinity" (เลข 8 แนวนอน) ซึ่งแสดงถึงการทำงานที่เป็นวงจรต่อเนื่องไม่สิ้นสุด ประกอบด้วยขั้นตอนหลักๆ ดังนี้

- **Plan & Code:** วางแผนและเขียนโค้ด
- **Build & Test:** สร้าง Software package และทดสอบอัตโนมัติ (CI - Continuous Integration)
- **Release & Deploy:** ส่งมอบและติดตั้งซอฟต์แวร์ไปยังระบบจริง (CD - Continuous Deployment)
- **Operate & Monitor:** ดูแลการทำงานและเก็บข้อมูลเพื่อนำกลับไปปรับปรุง

ประโยชน์ของการนำ DevOps มาใช้

การเปลี่ยนมาใช้ DevOps ไม่ใช่แค่เรื่องของเครื่องมือ แต่เป็นเรื่องของผลลัพธ์ทางธุรกิจที่ชัดเจน

1. **ความเร็ว (Speed)** สามารถปล่อยฟีเจอร์ใหม่ๆ หรือแก้ไขบั๊กได้บ่อยและเร็วขึ้น (High Deployment Frequency)
2. **ความเสถียร (Reliability)** การใช้การทดสอบอัตโนมัติช่วยลดความผิดพลาดที่เกิดจากมนุษย์ (Human Error) ทำให้ระบบมีปัญหาน้อยลง

3. **การทำงานร่วมกัน (Improved Collaboration)** ลดความขัดแย้งระหว่างทีม Dev ที่เน้น "การเปลี่ยนแปลง" และทีม Ops ที่เน้น "ความนิ่งของระบบ"
4. **ความปลอดภัย (Security)** เมื่อนำแนวคิด **DevSecOps** มาใช้ จะมีการตรวจความปลอดภัยตั้งแต่ขั้นตอนการเขียนโค้ด ไม่ใช่รอตรวจตอนสุดท้าย
5. **การกู้คืนระบบ (Fast Recovery)** หากเกิดข้อผิดพลาด ระบบ DevOps ที่ดีจะสามารถ Rollback หรือย้อนกลับไปยังเวอร์ชันก่อนหน้าได้ทันที

หมายเหตุ

DevOps คือการทำให้ "คน + กระบวนการ + เครื่องมือ" ทำงานสอดประสานกัน เพื่อส่งมอบซอฟต์แวร์ที่มีคุณภาพให้กับผู้ใช้งานได้เร็วที่สุดเท่าที่จะเป็นไปได้

เครื่องมือสำคัญในโลก DevOps

เครื่องมือ	หน้าที่	ตัวอย่าง
Version Control	ควบคุมเวอร์ชันโค้ด	Git, GitHub
CI/CD	Automate Build/Test/Deploy	Jenkins, GitLab CI, GitHub Actions
Container	บรรจุแอปพร้อม Dependency	Docker, Podman