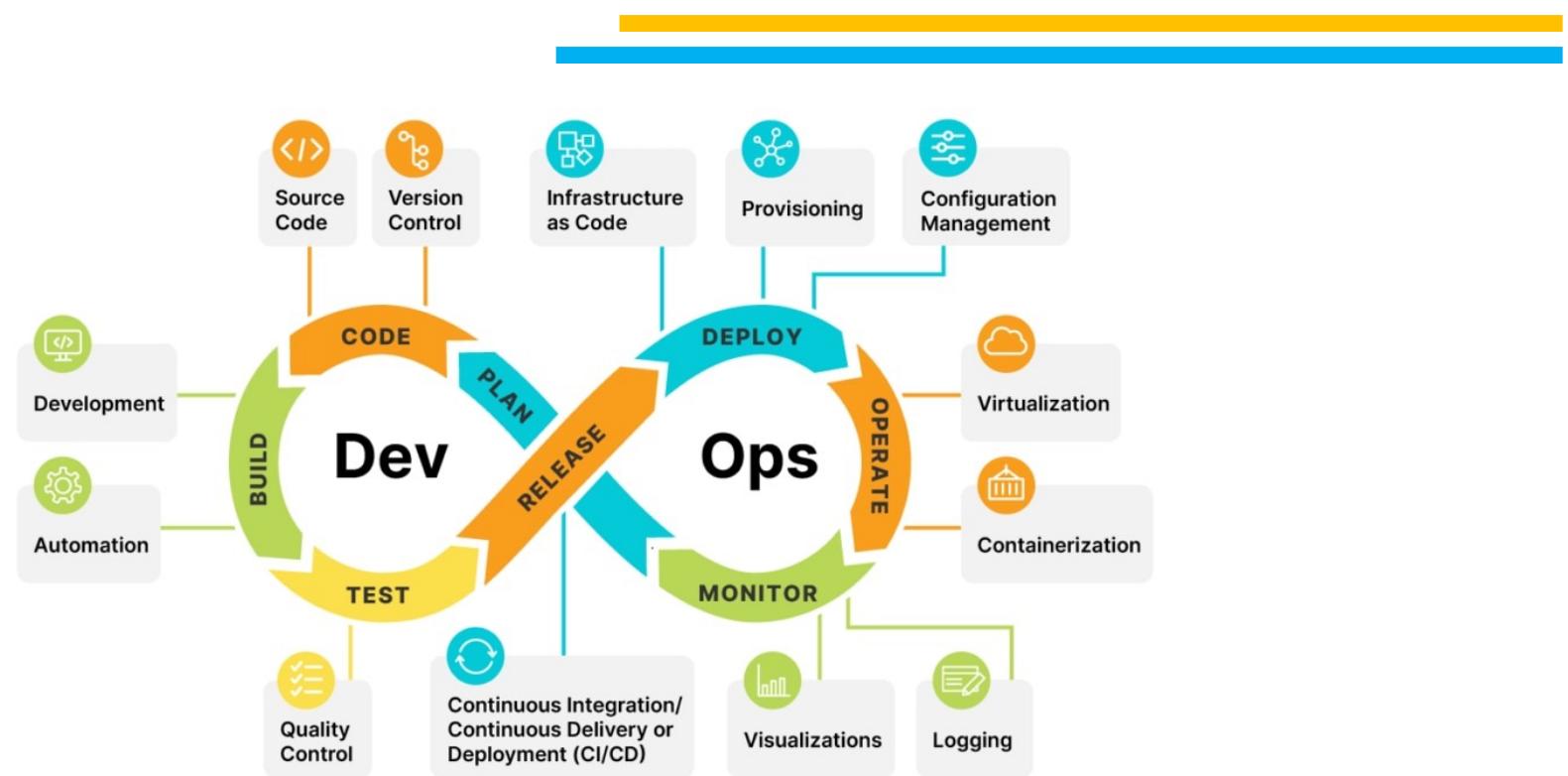


ความรู้เบื้องต้นเกี่ยวกับการพัฒนาและปฏิบัติการ (DevOps)



เนื้อหา

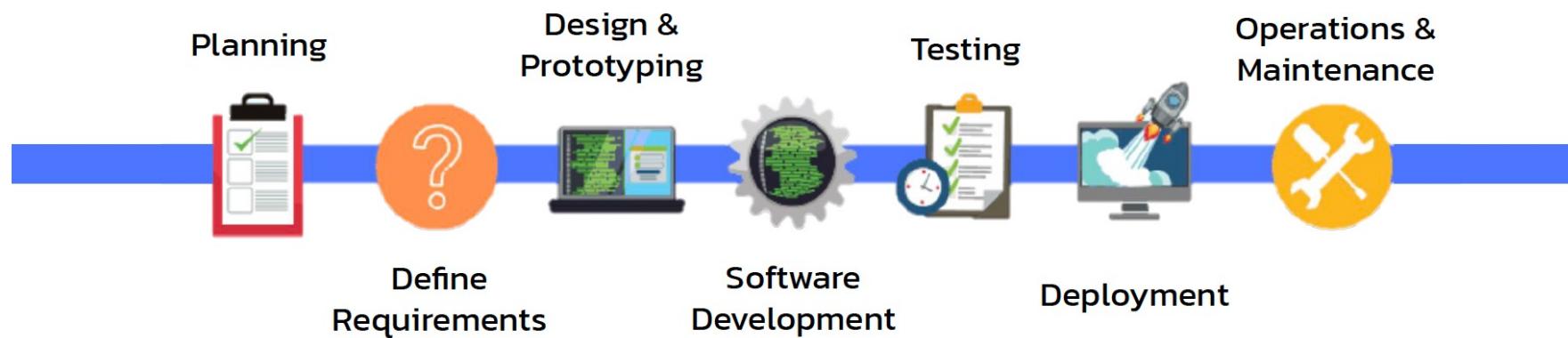


- DevOps คือ อะไร?
- ทำไมต้องเป็น DevOps?
- DevOps Culture
- DevOps Practices
- DevOps Tool
- อาชีพที่กำลังมาแรงในสายงาน IT : DevOps Engineering

DevOps គឹវ ឧបន៍?

Software Development Life Cycle (SDLC)

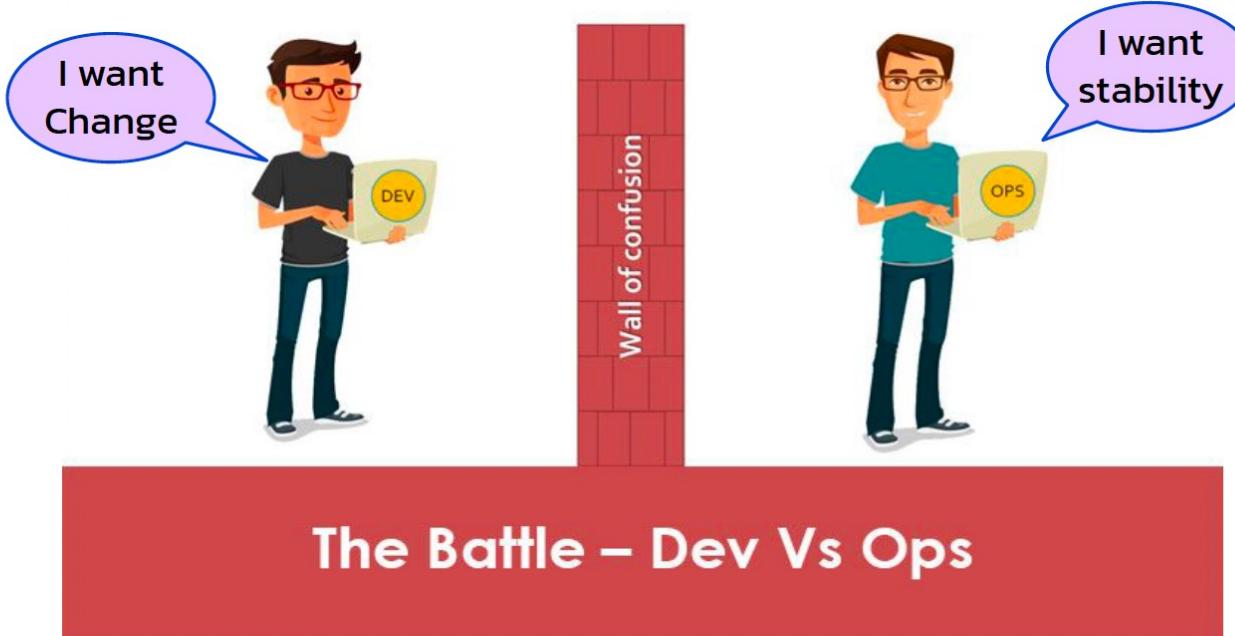
Seven Phases of Software Development Life Cycle



DevOps គី អេឡា?(ពេល)

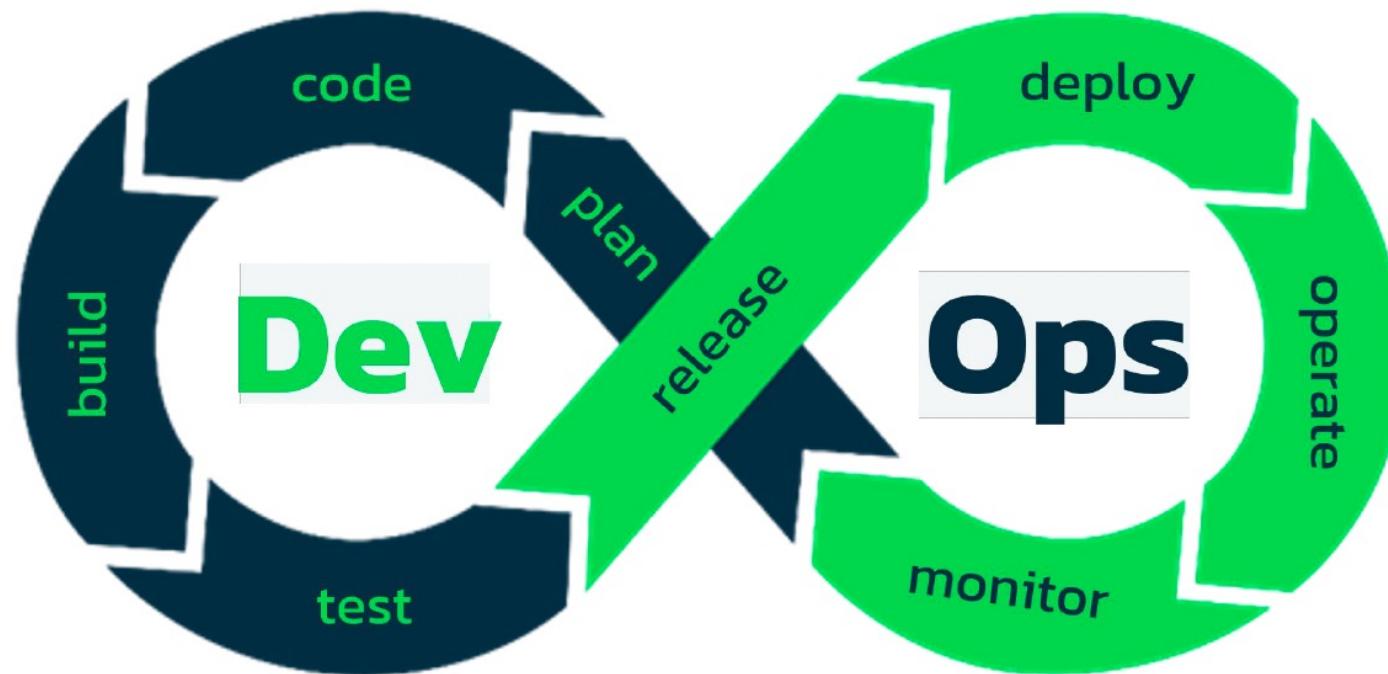


Developer vs Operation



DevOps คือ อะไร?(ต่อ)

Devops เกิดมาได้ยังไง

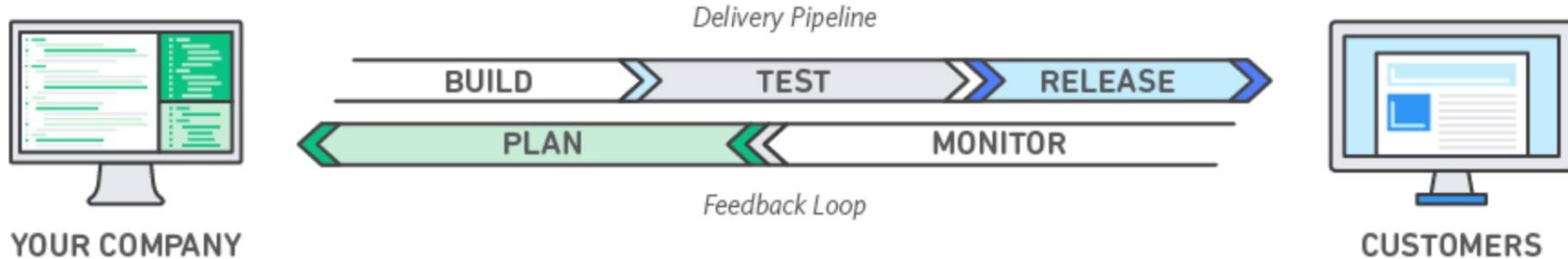


DevOps คือ อะไร?(ต่อ)

- DevOps คือ การผสมผสานของศาสตร์วัฒนธรรม การปฏิบัติและเครื่องมือที่เพิ่มความสามารถขององค์กรในการส่งมอบแอปพลิเคชันและบริการที่มีความเร็วสูง ทำให้เกิดการพัฒนาและปรับปรุงผลิตภัณฑ์อย่างรวดเร็วกว่าองค์กรที่ใช้กระบวนการพัฒนาซอฟต์แวร์และการจัดการโครงสร้างทางเทคโนโลยีในรูปแบบเดิม ความเร็วนี้ทำให้องค์กรสามารถให้บริการลูกค้าได้ดีขึ้นและแข่งขันได้มีประสิทธิภาพมากขึ้น ซึ่งหลักเกณฑ์หลักของ DevOps คือ การทำงานร่วมกัน การทำงานแบบอัตโนมัติและวัฒนธรรมในการพัฒนาที่มีการปรับปรุงตลอดเวลา

DevOps คือ อะไร?(ต่อ)

- DevOps เป็นแนวความคิดเชิงวัฒนธรรม (Culture) และแนวทางปฏิบัติ (Practice) ที่เกิดมาเพื่อส่งเสริมการทำงานของ Development Team และ Operations Team เพื่อให้สามารถผลิตและส่งมอบ Software ไปยังผู้ใช้ได้อย่างรวดเร็วและมีประสิทธิภาพมากที่สุด โดยมีเป้าหมายคือการลดต้นทุนของเวลา (Time) และ ค่าใช้จ่าย (Cost) ที่ไม่จำเป็นออกไป โดยอาศัยเครื่องมือ (Tools) ต่าง ๆ ที่มีอยู่มำทำให้เกิดความราบรื่นในระหว่างกระบวนการพัฒนาไปจนถึงการ Deploy ระบบขึ้น Production



Development Team และ Operations Team หมายถึง ใคร

- แต่ละองค์กรอาจจะมอง Development Team และ Operations Team ไม่เหมือนกัน ทั้งนี้ขึ้นอยู่กับบทบาท ความรับผิดชอบ และ Scope งาน ที่คน ๆ นั้นได้รับว่ามีส่วนช่วยเหลือ หรือร่วมพัฒนาระบบมากแค่ไหน

Development Team



- Developer
- Designer
- Software Tester
- QA (Quality Assurance)
- BA (Business Analyst)
- PM (Project Manager)
- PO (Product Owner)
- หรือคนที่ร่วมกันพัฒนาระบบนี้ ๆ ขึ้นมาโดยตรง

Operations Team



- System Administrator
- Database Administrator
- Network Engineer
- Marketing
- ประชาสัมพันธ์
- Stakeholder (ผู้มีส่วนได้ส่วนเสีย)
- และคนอื่น ๆ ที่เกี่ยวข้องกับ Project นั้น (ที่ไม่ได้อยู่ใน Development Team)

ทำไมต้องเป็น DevOps

- ซอฟต์แวร์และอินเทอร์เน็ตได้เปลี่ยนแปลงโลกและอุตสาหกรรมต่าง ๆ ตั้งแต่การซื้อขายสินค้าไปจนถึงความบันเทิงและการธนาคาร
- ซอฟต์แวร์ไม่ได้เป็นเพียงสิ่งสนับสนุนการดำเนินธุรกิจอีกต่อไป แต่มีบทบาทสำคัญในทุกรายละเอียดของธุรกิจ
 - ใช้ซอฟต์แวร์สำหรับติดต่อกับลูกค้าผ่านซอฟต์แวร์ที่ให้บริการออนไลน์หรือแอปพลิเคชันและบนอุปกรณ์ต่าง ๆ
 - ใช้ซอฟต์แวร์เพื่อเพิ่มประสิทธิภาพการดำเนินงานตลอดห่วงโซ่มูลค่า เช่น โลจิสติกส์ การสื่อสารและการดำเนินงาน
 - ใช้ซอฟต์แวร์ในการเปลี่ยนวิธีการในการออกแบบ พัฒนา และส่งมอบผลิตภัณฑ์
 - ส่งผลให้ปรับตัวในโลกปัจจุบันต้องเปลี่ยนวิธีการในการพัฒนาและส่งมอบซอฟต์แวร์

วิัฒนาการของ DevOps

- DevOps (Development and Operations) เกิดขึ้นจากความต้องการในการแก้ปัญหาและปรับปรุงกระบวนการพัฒนาซอฟต์แวร์แบบเดิมที่มีปัญหานำมาจากการทำงานร่วมกันระหว่างทีมพัฒนาและทีมดำเนินงานทางไอที
- นิยามแรกของ DevOps เริ่มต้นในปี 2010 โดยชุมชน IT และนักพัฒนาที่เห็นถึงความจำเป็นของการปรับปรุงกระบวนการเพื่อให้การส่งมอบซอฟต์แวร์เป็นไปได้อย่างมีประสิทธิภาพและรวดเร็วมากขึ้น

วิัฒนาการของ DevOps(ต่อ)

- ต้นกำเนิดของ DevOps
 - มาจากการที่บริษัทต่าง ๆ เกิดปัญหาในการพัฒนาซอฟต์แวร์ซึ่งไม่สามารถที่จะพัฒนา และปล่อย (Launch) Product ได้เร็วเท่าที่ควรจะเป็น
 - วิศวกรทั่วโลกจึงได้พยายามคิดค้นวิธีการและแนวทางปฏิบัติต่าง ๆ ที่เห็นว่าดีเพื่อนำมาแก้ไขปัญหาที่เกิดขึ้น โดยมีวัตถุประสงค์เพื่อให้สามารถที่จะพัฒนาและส่งมอบซอฟต์แวร์ได้เร็ว ลดต้นทุนเวลา ค่าใช้จ่าย และมีประสิทธิภาพในการทำงาน
 - แนวความคิดและแนวทางปฏิบัตินี้เกิดการบอกรือและทำกันจนกลายเป็นแนวทางปฏิบัติ (Practice) และ วัฒนธรรม (Culture)
 - และเรียกและนิยามวัฒนธรรมนี้กันว่า DevOps

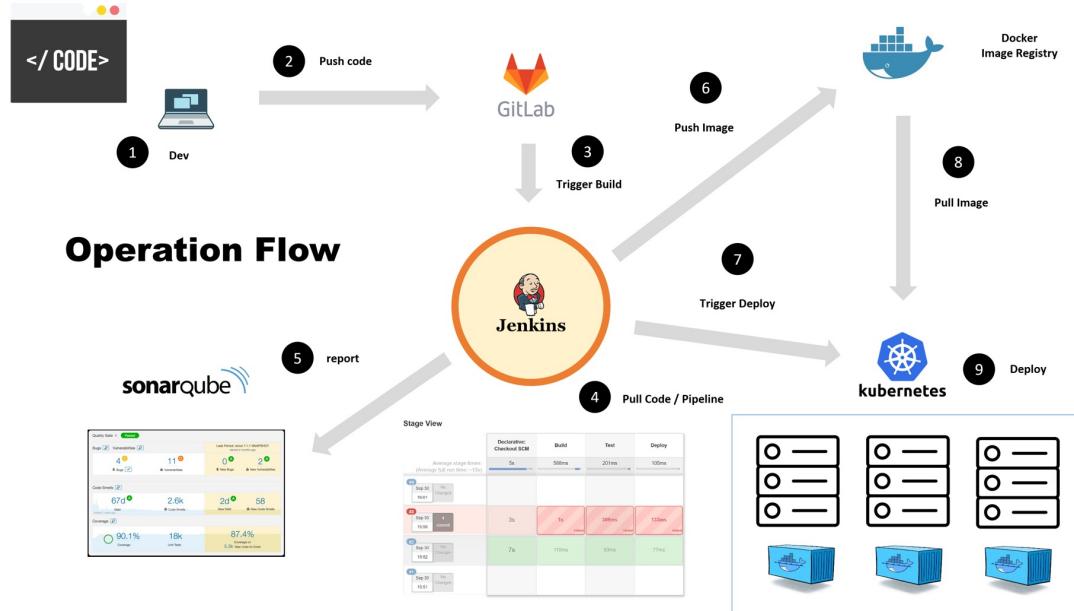
DevOps ทำงานอย่างไร

- ภายในแบบจำลอง DevOps ทีมพัฒนาและทีมดำเนินงานไม่ได้ทำงานแยกกัน (siloes) อีกต่อไป บางครั้งทีมเหล่านี้ถูกผสมเข้าด้วยกันเป็นทีมเดียวที่รับผิดชอบกิจกรรมตลอด life cycle ของแอปพลิเคชัน ตั้งแต่การพัฒนาและทดสอบไปจนถึงการ deploy สู่การใช้งานและการพัฒนาทักษะต่าง ๆ ก็ไม่ถูกจำกัดไว้แค่การทำงานเพียงฟังก์ชันเดียวอีกต่อไป
- ในแบบจำลอง DevOps, ทีมประกันคุณภาพและทีมด้านความมั่นคงปลอดภัยอาจเข้าไปผนวกกับทีมพัฒนาและทีมดำเนินงานมากขึ้น และเกิดขึ้นตลอดกระบวนการพัฒนาแอปพลิเคชัน กลยุทธ์เป้าหมายสำคัญอีกเป้าหมายหนึ่งของ DevOps ซึ่งจะเรียกว่า DevSecOps
- ทีมเหล่านี้ใช้วิธีการอัตโนมัติกับกระบวนการในรูปแบบเดิมที่ใช้การทำงานแบบ manual และซ้ำโดยใช้เทคโนโลยีและเครื่องมือที่ช่วยให้การทำงานและการพัฒนาแอปพลิเคชันเป็นไปอย่างรวดเร็วและมีความน่าเชื่อถือ เครื่องมือเหล่านี้ยังช่วยให้วิศวกรสามารถทำงานอย่างอิสระ เช่น การ deploy โค้ดหรือการจัดหาโครงสร้างพื้นฐาน ที่ปกติจะต้องการความช่วยเหลือจากทีมอื่น ๆ ซึ่งสิ่งเหล่านี้จะเพิ่มความเร็วในการทำงานของทีมได้ในอนาคต

รูปแบบการทำงาน และเครื่องมือที่ใช้

- DevOps ไม่ได้มีรูปแบบหรือเครื่องมือที่ใช้ในการทำงานที่ตายตัว บางบริษัทอาจจะทำ DevOps ในรูปแบบหนึ่ง บางบริษัทก็ใช้เครื่องมืออีกเครื่องมือหนึ่ง
- แต่สุดท้ายแล้วก็มีเป้าหมายในการทำ DevOps แบบเดียวกัน นั่นก็คือ การทำให้ Developer สามารถที่จะปล่อย Software ออกมาได้เร็ว และมีประสิทธิภาพมากที่สุด พร้อมทั้งช่วยลดต้นทุนด้านเวลา และค่าใช้จ่าย ต่าง ๆ ที่จะเกิดขึ้นในระบบ

DevOps ด้วยระบบอัตโนมัติ (Automation)



- แนวคิด Automation เป็นหนึ่งในแนวความคิด และแนวทางปฏิบัติที่ดี (Best Practice) ที่ถูกนำมาใช้กับ DevOps มากที่สุด โดยการ Setup ระบบทั้งหมดให้ทำงานแบบอัตโนมัติ ให้สามารถทำงานแทนคน เพื่อลดความซ้ำซ้อน และข้อผิดพลาดที่อาจจะเกิดจากคนได้
- จากภาพเมื่อเราเขียน Code เสร็จ (1) จากนั้น Save (Push) Code ขึ้นไปเก็บไว้บน Git Server (ที่ฝาก Source Code) (2)
- จะมีระบบอัตโนมัติ มาทำการดึง Source Code ที่เราเขียนไว้เพื่อนำไป Build, Test, Release, พร้อมทั้ง Deploy ขึ้น Server ให้เราเองโดยอัตโนมัติ (3 - 9) โดยไม่ต้องใช้คนทำเหมือนในอดีต

ประโยชน์ของการทำ DevOps แบบ Automation

- Dev งานได้ไวขึ้น
- Deploy งานได้ไวขึ้น
- Tracking ปัญหาได้ไวขึ้น
- แก้ BUG ได้ไวขึ้น
- ลดความซ้ำซ้อนของการทำงาน
- Software มีคุณภาพมากขึ้น
- ส่งงานลูกค้าได้ไวขึ้น

หลักการสำคัญของ DevOps

- การทำงานร่วมกันและการสื่อสาร:** DevOps สนับสนุนการทำงานร่วมกันและการสื่อสารแบบเปิดกว้างระหว่างทีมพัฒนาและทีมดำเนินงาน การสร้างวัฒนธรรมที่สนับสนุนความร่วมมือมีผลต่อประสิทธิภาพและความน่าเชื่อถือในการทำงานร่วมกัน
- การทำงานแบบอัตโนมัติ :** การทำงานแบบอัตโนมัติเป็นหลักการสำคัญของ DevOps มีการใช้เครื่องมือและกระบวนการอัตโนมัติเพื่อลดความซับซ้อนและความผิดพลาดในกระบวนการทำงาน
- การทดสอบแบบต่อเนื่อง :** DevOps ให้ความสำคัญกับการทดสอบที่ส่งผลอย่างรวดเร็วและต่อเนื่อง การทดสอบนี้ช่วยตรวจสอบข้อผิดพลาดและปัญหาต่าง ๆ ในขั้นตอนการพัฒนา
- การจัดการโครงสร้างเป็นรหัส (Infrastructure as Code - IaC):** IaC ช่วยในการจัดการโครงสร้างและการตั้งค่าโครงสร้างในรูปแบบของรหัสทำให้กระบวนการติดตั้งเป็นไปได้อย่างสม่ำเสมอ

หลักการสำคัญของ DevOps(ต่อ)

5. การทดสอบแบบต่อเนื่อง (Continuous Integration - CI): CI ช่วยให้ทีมสามารถสนับสนุนและทำการทดสอบได้ทุกครั้งที่มีการเปลี่ยนแปลงโค้ด ทำให้สามารถตรวจสอบข้อผิดพลาดและแก้ไขได้รวดเร็ว
6. การติดตั้งต่อเนื่อง (Continuous Deployment - CD): CD ทำให้สามารถส่งมอบซอฟต์แวร์ไปยังสภาพแวดล้อมการทดสอบหรือการดำเนินงานโดยอัตโนมัติหลังจากการทดสอบและทดสอบเรียบร้อยแล้ว
7. ความมั่นคงปลอดภัย (DevSecOps): DevOps รวมความมั่นคงปลอดภัยไว้ในกระบวนการ โดยการรวม DevSecOps เพื่อความมั่นใจในความปลอดภัยตลอดกระบวนการ
8. การพัฒนาแบบ Microservices: DevOps ช่วยให้ทีมพัฒนาและทำงานกับระบบ Microservices ได้อย่างมีประสิทธิภาพและยืดหยุ่น
9. การนำเข้าบริการคลาวด์ (Cloud Services): การนำเข้าบริการคลาวด์ช่วยให้ทีมสามารถทำการเปลี่ยนแปลงและปรับปรุงระบบได้ง่ายขึ้น

DevOps ไม่เพียงเป็นกระบวนการและเครื่องมือ แต่ยังเป็นวิธีการทำงานและวัฒนธรรมที่สนับสนุนความร่วมมือและการปรับปรุงอย่างต่อเนื่องในทีมและองค์กร

พื้นฐานจำเป็นสำหรับการทำ DevOps

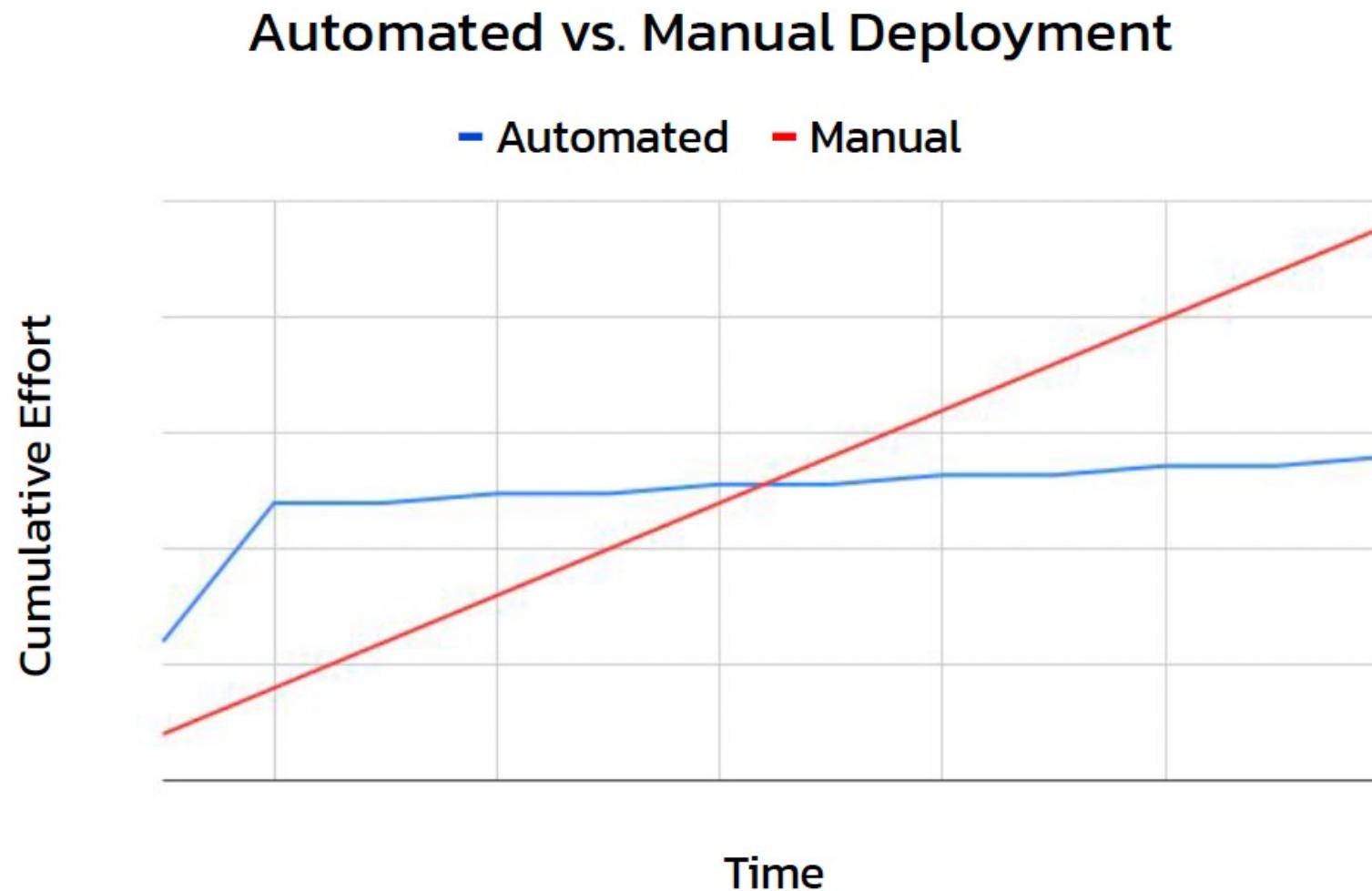


- พื้นฐาน Programming
- พื้นฐาน Operating System (OS) เช่น Windows, Linux, Mac etc.
- พื้นฐานการใช้ Command Line
- พื้นฐาน Software Container เช่น Docker, Kubernetes etc.
- พื้นฐาน Networking เช่น IP Address, การ Setup DNS etc.
- พื้นฐาน Database
- พื้นฐานการทำ Application Server
- พื้นฐานการทำ Cluster, Load Balancing, ..
- พื้นฐานการใช้ Cloud
- พื้นฐาน Security เช่น https etc.
- พื้นฐานความรู้เรื่องการทดสอบระบบ (Testing)
- อื่น ๆ

DevOps ทำไปแล้วได้อะไร

- Rapid Built and Deploy : ทำให้เกิดการ Built และ Deploy อย่างรวดเร็ว
- Stability and Reliability : ทำให้เกิดความเสถียรและน่าเชื่อถือ
- Reduce human error : ลดความผิดพลาดในการทำงานของมนุษย์
- Hybrid Cloud Deployment : ใช้เทคโนโลยีของ cloud แบบ hybrid มาช่วยในการ deploy

DevOps ทำไปแล้วได้อะไร(ต่อ)



ประโยชน์ของ DevOps



- ความเร็ว (Speed)
- การส่งมอบอย่างรวดเร็ว (Rapid Delivery)
- ความน่าเชื่อถือ (Reliability)
- ความสามารถในการขยายตัว (Scale)
- การเพิ่มความร่วมมือ (Improved Collaboration)
- ความปลอดภัย (Security)

ความเร็ว (Speed)

- การทำงานที่เร็วขึ้นเพื่อส่งมอบนวัตกรรมสำหรับลูกค้าได้เร็วขึ้น ปรับตัวต่อการเปลี่ยนแปลงในตลาดได้ดีขึ้น และเติบโตอย่างมีประสิทธิภาพเพื่อการขับเคลื่อนผลลัพธ์ธุรกิจ แบบจำลอง DevOps ช่วยให้ทีมพัฒนาและทีมดำเนินงานสามารถบรรลุผลลัพธ์เหล่านี้ได้ ตัวอย่างเช่น การใช้ microservices และ continuous delivery ทำให้ทีมสามารถเป็นเจ้าของบริการ และ release update ในการให้บริการนั้นได้เร็วขึ้น

การส่งมอบอย่างรวดเร็ว (Rapid Delivery)

- เพิ่มความถี่และความเร็วในการเผยแพร่เพื่อทำให้การสร้างสรรค์และปรับปรุงผลิตภัณฑ์ได้เร็วขึ้น ยิ่งเราสามารถปล่อย features ใหม่และแก้ไขข้อบกพร่องได้เร็วเท่าไหร่ก็ยิ่งสามารถตอบสนองความต้องการของลูกค้าและสร้างความได้เปรียบในการแข่งขันได้เร็วขึ้นเท่านั้น การผสานรวมอย่างต่อเนื่อง(Continuous integration)และการส่งมอบอย่างต่อเนื่อง (Continuous Delivery)เป็นแนวทางปฏิบัติที่ทำให้กระบวนการเผยแพร่ซอฟต์แวร์เป็นไปโดยอัตโนมัติซึ่งแต่การสร้างไปจนถึงการปรับใช้

ความน่าเชื่อถือ (Reliability)

- รับประกันคุณภาพของการอัปเดตแอปพลิเคชันและการเปลี่ยนแปลงโครงสร้างพื้นฐานเพื่อให้เราสามารถส่งมอบซอฟต์แวร์ได้อย่างรวดเร็วขึ้นในขณะที่ยังคงรักษาประสิทธิภาพของระบบ เช่น การใช้แนวทางปฏิบัติ เช่น Continuous Integration และ Continuous Delivery เพื่อทดสอบว่าการเปลี่ยนแปลงแต่ละครั้งนั้นใช้งานได้และมีความปลอดภัย แนวทางปฏิบัติในการตรวจสอบและการบันทึก (Monitoring and logging) ช่วยให้เรา_rับทราบข้อมูลประสิทธิภาพแบบเรียลไทม์

ความสามารถในการขยายตัว (Scale)

- ดำเนินการและจัดการโครงสร้างพื้นฐานและกระบวนการพัฒนาได้ตามขนาดของระบบ ระบบอัตโนมัติและ consistency ช่วยให้การจัดการระบบที่ซับซ้อนหรือเปลี่ยนแปลงเป็นไปอย่างมีประสิทธิภาพและลดความเสี่ยง ตัวอย่างเช่น โครงสร้างพื้นฐานเป็นโคด(Infrastructure as Code:IaC) ช่วยให้เราสามารถจัดการสภาพแวดล้อมในการพัฒนา การทดสอบ และการผลิตในลักษณะที่ทำซ้ำได้และมีประสิทธิภาพมากขึ้น

การเพิ่มความร่วมมือ (Improved Collaboration)

- สร้างทีมที่มีประสิทธิภาพมากขึ้นภายใต้รูปแบบวัฒนธรรม DevOps ซึ่งเน้นค่านิยม เช่น ความเป็นเจ้าของและความรับผิดชอบ นักพัฒนาและทีมปฏิบัติการทำงาน ร่วมกันอย่างใกล้ชิดและแบ่งปันความรับผิดชอบและรวมเวิร์กโฟล์วเข้าด้วยกัน สิ่งนี้ จะช่วยลดความไร้ประสิทธิภาพและประหยัดเวลา (เช่นลดระยะเวลาการส่งมอบระหว่างนักพัฒนาและการดำเนินงาน การเขียนโค้ดที่คำนึงถึงสภาพแวดล้อมที่รัน)

ความปลอดภัย (Security)

- เปลี่ยนแปลงอย่างรวดเร็วในขณะที่ยังคงควบคุมและรักษาการปฏิบัติตามข้อกำหนด สามารถใช้โมเดล DevOps ได้โดยไม่ต้องลดความปลอดภัยลงโดยใช้นโยบายการปฏิบัติตามข้อกำหนดอัตโนมัติ การควบคุมแบบละเอียดและเทคนิคการจัดการการกำหนดค่า ตัวอย่างเช่น การใช้โครงสร้างพื้นฐานเป็นโค้ด(IaC) และนโยบายเป็นโค้ด(Policy as Code) ทำให้สามารถกำหนดและติดตามการปฏิบัติตามข้อกำหนดได้กับระบบทุกขนาด

ประโยชน์ในเชิงธุรกิจของ DevOps

- DevOps มีประโยชน์หลายประการในการสามารถช่วยเพิ่มประสิทธิภาพ ลดเวลาตลอดจนเพิ่มความสามารถในการแข่งขันขององค์กร ได้แก่
 1. ลดเวลาในการเข้าสู่ตลาด (Faster Time-to-Market) : DevOps สะท้อนการทำงานแบบอัตโนมัติและการทำงานร่วมกันซึ่งจะช่วยลดเวลาในการพัฒนา ทดสอบ และนำเสนอบอฟต์แวร์ การลดเวลาเหล่านี้ช่วยให้ธุรกิจสามารถเผยแพร่ features และ updates ได้เร็วขึ้น ตอบสนองต่อความต้องการของตลาดได้ดีขึ้น
 2. การทำงานร่วมกันและการสื่อสารที่ดีขึ้น : DevOps ทำลายกำแพงระหว่างทีมพัฒนาและทีมดำเนินงาน สร้างวัฒนธรรมที่สนับสนุนการทำงานร่วมกัน มีการสื่อสารที่ดีและการรับผิดชอบร่วมกันทำให้สามารถแก้ไขปัญหาได้เร็วขึ้นและมีความเข้าใจที่ตรงกันในการทำงานธุรกิจ
 3. เพิ่มประสิทธิภาพและผลิตภัณฑ์ : การทำงานแบบอัตโนมัติและการทำงานร่วมกันทำให้กระบวนการเป็นไปอย่างราบรื่น ลดความผิดพลาดและเน้นการใช้ทรัพยากรสำหรับกิจกรรมที่มีความสำคัญมากขึ้น
 4. คุณภาพและความน่าเชื่อถือที่ดีขึ้น : การทำ CI (Continuous Integration) และ CD (Continuous Deployment) ใน DevOps ช่วยเพิ่มคุณภาพของซอฟต์แวร์ การทดสอบและการนำเสนออัตโนมัติช่วยจับและแก้ไขปัญหาในขั้นตอนการพัฒนาได้เร็วขึ้น

ประโยชน์ในเชิงธุรกิจของ DevOps (ต่อ)

5. **ลดต้นทุน** : การทำงานแบบอัตโนมัติและการปรับปรุงประสิทธิภาพช่วยลดต้นทุนที่เกี่ยวข้องกับกระบวนการทำงานแบบมือ ความผิดพลาดจากมนุษย์ และการหยุดทำงาน การตรวจจับและแก้ไขปัญหาได้เร็วขึ้นย่อมลดค่าใช้จ่ายลง
6. **เพิ่มความพึงพอใจของลูกค้า** : การส่งมอบที่มีคุณภาพสูงและการปรับปรุงที่ถูกต้องช่วยเพิ่มความพึงพอใจของลูกค้า DevOps ช่วยให้ธุรกิจสามารถตอบสนองต่อคำติชมและความต้องการที่เปลี่ยนไปได้อย่างรวดเร็ว.
7. **การพัฒนาและความยืดหยุ่น** : DevOps สอดคล้องกับมาตรฐานการพัฒนาแบบ Agile ช่วยให้องค์กรสามารถปรับตัวได้อย่างรวดเร็wtต่อความเปลี่ยนแปลงที่เกิดขึ้น
8. **นวัตกรรมมากขึ้น** : ด้วยการสนับสนุนวัฒนธรรมในการทำงานร่วมกันและการปรับปรุงอย่างต่อเนื่อง DevOps ส่งเสริมให้เกิดนวัตกรรม ทีมได้รับการสนับสนุนในการทดลอง รับความเสี่ยง และนำเสนอคุณลักษณะใหม่.

ประโยชน์ในเชิงธุรกิจของ DevOps

9. ความยืดหยุ่นและสามารถปรับตัวได้ : การทำ DevOps สนับสนุนความยืดหยุ่นและความสามารถในการปรับตัวของโครงสร้างและแอปพลิเคชัน บริการคลาวด์ และการใช้ containerization ช่วยให้องค์กรสามารถปรับขนาดตามความต้องการและปรับตัวตามความต้องการธุรกิจ.

- การนำ DevOps มาใช้ไม่เพียงแค่เป็นการเปลี่ยนแปลงทางเทคนิค แต่ยังเป็นการเปลี่ยนแปลงทางวัฒนธรรมที่มีผลต่อวิธีที่ทีมทำงานร่วมกันและนำคุณค่าไปสู่ธุรกิจ การรวมห้องการขยายทักษะทางทัศนคติและวิธีการทำงานของทีมทำให้ DevOps มีความสำคัญในการทำให้องค์กรประสบความสำเร็จ

ปรับใช้ DevOps Model ได้อย่างไร

- DevOps Cultural Philosophy : ปรัชญาวัฒนธรรมของ DevOps
- DevOps Practices Explained : แนวทางปฏิบัติของ DevOps

ประชญาวัฒนธรรมของ DevOps

- การเปลี่ยนไปใช้ DevOps จำเป็นต้องมีการเปลี่ยนแปลงทั้งวัฒนธรรมและความคิดสิ่งที่เป็นพื้นฐานที่สุด คือ การขัดอุปสรรคระหว่างสองทีมที่แยกการทำงานกันแบบเดิม คือ ส่วนของการพัฒนาและการดำเนินงาน ในบางองค์กรอาจไม่มีทีมพัฒนาและทีมปฏิบัติการแยกต่างหาก วิศวกรอาจทำได้ทั้งสองอย่าง ด้วย DevOps ทั้งสองทีมจะทำงานร่วมกันเพื่อเพิ่มประสิทธิภาพทั้งผลผลิตของนักพัฒนาและความน่าเชื่อถือของการดำเนินงาน โดยมุ่งมั่นที่จะสื่อสารบ่อยครั้งขึ้นเพื่อเพิ่มประสิทธิภาพและปรับปรุงคุณภาพของบริการที่ส่งมอบให้กับลูกค้า ทั้งสองทีมเป็นเจ้าของบริการอย่างเต็มที่ซึ่งมักจะอยู่นอกเหนือบทบาทที่ระบุไว้โดยคำนึงถึงความต้องการของลูกค้าปลายทางและทำให้เกิดวิธีการที่ทีมสามารถมีส่วนร่วมในการแก้ปัญหาความต้องการเหล่านั้น ทีมประกันคุณภาพและความปลอดภัยอาจรวมเข้ากับทีมเหล่านี้ด้วยก็ได้

แนวทางปฏิบัติของ DevOps

- มีแนวทางปฏิบัติที่สำคัญบางประการที่ช่วยให้องค์กรสามารถสร้างสรรค์สิ่งใหม่ ๆ ได้เร็วขึ้นผ่านระบบอัตโนมัติและปรับปรุงกระบวนการพัฒนาซอฟต์แวร์และการจัดการโครงสร้างพื้นฐาน แนวทางปฏิบัติเหล่านี้ส่วนใหญ่ทำได้ด้วยเครื่องมือที่เหมาะสม
- วิธีปฏิบัติพื้นฐานอย่างหนึ่งคือการอัปเดตบ่อยครั้ง แต่เล็กน้อย ซึ่งเป็นวิธีการที่องค์กรสร้างสรรค์สิ่งใหม่ ๆ ให้กับลูกค้าได้เร็วขึ้น การอัปเดตเหล่านี้มักจะมีลักษณะเพิ่มขึ้นมากกว่าการอัปเดตเป็นครั้งคราวที่ดำเนินการภายใต้แนวทางปฏิบัติในการเผยแพร่แบบเดิม การอัปเดตบ่อยครั้งแต่มีขนาดเล็กทำให้การปรับใช้แต่ละครั้งมีความเสี่ยงน้อยลง ช่วยให้ทีมแก้ไขข้อบกพร่องได้เร็วขึ้นเนื่องจากทีมสามารถระบุการปรับใช้ล่าสุดที่ทำให้เกิดข้อผิดพลาดได้ เมื่อเวลาจังหวะและขนาดของการอัปเดตจะแตกต่างกันไป แต่องค์กรที่ใช้โมเดล DevOps จะปรับใช้การอัปเดตบ่อยกว่าองค์กรที่ใช้แนวทางการพัฒนาซอฟต์แวร์แบบเดิม

แนวทางปฏิบัติของ DevOps(ต่อ)

- องค์กรอาจใช้สถาปัตยกรรมไมโครเซอร์วิสเพื่อทำให้แอปพลิเคชันมีความยืดหยุ่นมากขึ้นและเปิดใช้งานนวัตกรรมที่รวดเร็วขึ้น สถาปัตยกรรมไมโครเซอร์วิสแยกระบบขนาดใหญ่และซับซ้อนออกเป็นโครงการอิสระและมีความซับซ้อนลดลง และ พลิเคชันถูกแบ่งออกเป็นส่วนประกอบ (บริการ) จำนวนมากโดยแต่ละบริการมีขอบเขตตามวัตถุประสงค์หรือฟังก์ชันเดียวและดำเนินการอย่างอิสระจากบริการอื่น ๆ และแอปพลิเคชันโดยรวม สถาปัตยกรรมนี้ช่วยลดค่าใช้จ่ายในการประสานงานของการอัปเดตแอปพลิเคชันและเมื่อแต่ละบริการเป็นความรับผิดชอบของทีมขนาดเล็กทำให้การทำงานคล่องตัวและทำงานได้เร็วขึ้น

แนวทางปฏิบัติของ DevOps (ต่อ)

- การรวมกันของไมโครเซอร์วิสและความถี่ในการเผยแพร่ที่เพิ่มขึ้นนำไปสู่การปรับใช้ที่มากขึ้นอย่างมีนัยสำคัญ ซึ่งอาจทำให้เกิดความท้าทายในการปฏิบัติงาน ดังนั้นแนวทางปฏิบัติของ DevOps เช่น การบูรณาการอย่างต่อเนื่องและการส่งมอบอย่างต่อเนื่องช่วยแก้ปัญหาเหล่านี้และช่วยให้องค์กรส่งมอบได้อย่างรวดเร็วในลักษณะที่ปลอดภัยและเชื่อถือได้ แนวทางปฏิบัติด้านระบบอัตโนมัติของโครงสร้างพื้นฐาน เช่น โครงสร้างพื้นฐานจัดการโค้ดและการกำหนดค่า ช่วยให้ทรัพยากรการประมวลผลมีความยืดหยุ่นและตอบสนองต่อการเปลี่ยนแปลงบ่อยครั้ง นอกจากนี้ การใช้การตรวจสอบและการบันทึกยังช่วยให้วิศวกรติดตามประสิทธิภาพของแอปพลิเคชันและโครงสร้างพื้นฐานเพื่อให้สามารถตอบสนองต่อปัญหาได้อย่างรวดเร็ว
- แนวทางปฏิบัติเหล่านี้ช่วยให้องค์กรส่งมอบการอัปเดตที่รวดเร็วและเชื่อถือได้มากขึ้นให้กับลูกค้าของตน

ความท้าทายของ DevOps ในกระบวนการพัฒนา software ในรูปแบบเดิม

การนำ DevOps เข้าสู่กระบวนการพัฒนาซอฟต์แวร์และดำเนินงานมีความท้าทายบางประการเมื่อเปรียบเทียบกับการทำงานในรูปแบบเดิม:

- **ขั้นตอนและกระบวนการที่ทันสมัย :** การนำ DevOps เข้ามาใช้ในองค์กรที่มีขั้นตอนและกระบวนการทางซอฟต์แวร์ในรูปแบบเดิมที่ล้าสมัยเป็นสิ่งที่ท้าทาย การปรับปรุงและปรับเปลี่ยนกระบวนการเพื่อให้สอดคล้องกับหลักการของ DevOps อาจต้องพึ่งกับความหยุดชะงักและความไม่เข้าใจ
- **การแบ่งแยกทีมในการพัฒนาและปฏิบัติการซอฟต์แวร์ :** ถ้าหากมีการแบ่งแยกทีมในการพัฒนาและปฏิบัติการซอฟต์แวร์ในรูปแบบเดิม การนำ DevOps เข้าสู่องค์กรอาจต้องพึ่งปัญหาเนื่องจากทีมทำงานไม่สอดคล้องกัน
- **ขาดทรัพยากรที่มีความเชี่ยวชาญ :** การทำ DevOps อาจต้องการทรัพยากรที่มีความเชี่ยวชาญในด้านต่าง ๆ เช่น Automation CI/CD Cloud ซึ่งอาจเป็นเรื่องท้าทายในการหาคนที่มีทักษะเหล่านี้
- **การเปลี่ยนแปลงทางวัฒนธรรม :** DevOps ไม่เพียงแค่เปลี่ยนแปลงทางเทคนิค แต่ยังเป็นการเปลี่ยนแปลงทางวัฒนธรรมที่มีผลต่อวิธีการทำงานของทีม ความยึดหยุ่นและการทำงานร่วมกันต้องเป็นส่วนหนึ่งของวัฒนธรรม

ความท้าทายของ DevOps ในกระบวนการพัฒนา software ในรูปแบบเดิม(ต่อ)

- **การทดสอบและการรักษาคุณภาพ :** การทดสอบแบบต่อเนื่องและการรักษาคุณภาพเป็นส่วนสำคัญของ DevOps แต่การนำเข้ากระบวนการนี้อาจต้องพบร่วมกับการที่ทีมทดสอบและทีมพัฒนามีความสามารถทำงานร่วมกันอย่างเต็มที่
- **ความมั่นคงทางปฏิบัติ :** การบำรุงรักษาระบบในสภาพแวดล้อมการทำงานจริงและการดำเนินงานที่มีความมั่นคงเป็นอีกความท้าทายที่ต้องเผชิญหน้า ความมั่นคงทางปฏิบัติเป็นสิ่งสำคัญในการให้บริการที่มีคุณภาพ
- **การจัดการระบบอินฟราสตรัคเจอร์ (Legacy Systems) :** ถ้าหากระบบมีประวัติการทำงานมาอย่างยาวนาน การนำ DevOps เข้าสู่ระบบเหล่านี้อาจต้องพบร่วมกับทีมทำงานที่มีประสิทธิภาพต่ำ
- **การจัดการความมั่นคงปลอดภัย :** ความมั่นคงปลอดภัยต้องเป็นส่วนหนึ่งของ DevOps (DevSecOps) แต่การจัดการความมั่นคงปลอดภัยในกระบวนการที่กำลังเปลี่ยนแปลงอาจเป็นเรื่องที่ท้าทาย
- **การบริหารจัดการเปลี่ยนแปลง :** การทำ DevOps ส่งผลให้มีการเปลี่ยนแปลงบ่อย ๆ ทั้งในโครงสร้างและกระบวนการ การบริหารจัดการการเปลี่ยนแปลงที่มีประสิทธิภาพเป็นสิ่งสำคัญ การนำ DevOps เข้าสู่องค์กรที่มีการดำเนินงานแบบทางเดิมอาจต้องมีการแก้ไขและปรับปรุงในหลายด้าน เพื่อให้เกิดการทำงานร่วมกันระหว่างทีม การใช้เทคโนโลยีที่ทันสมัย และการเปลี่ยนแปลงทางวัฒนธรรม เป็นไปอย่างมีประสิทธิภาพ

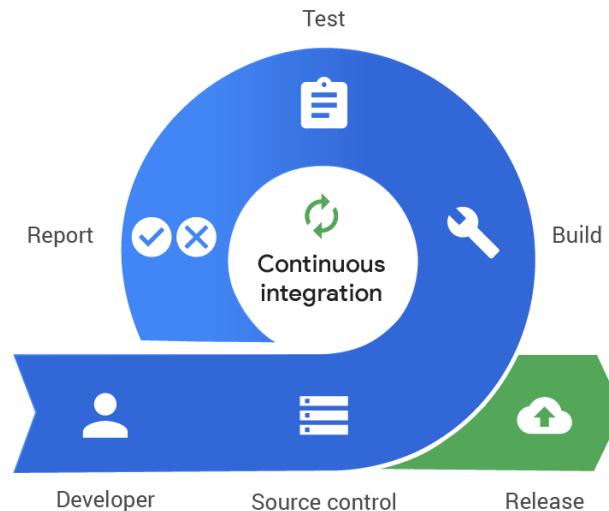
DevOps Practices



- DevOps best practices ประกอบด้วย :
 - Continuous Integration
 - Continuous Delivery
 - Microservices
 - Infrastructure as Code
 - Monitoring and Logging
 - Communication and Collaboration

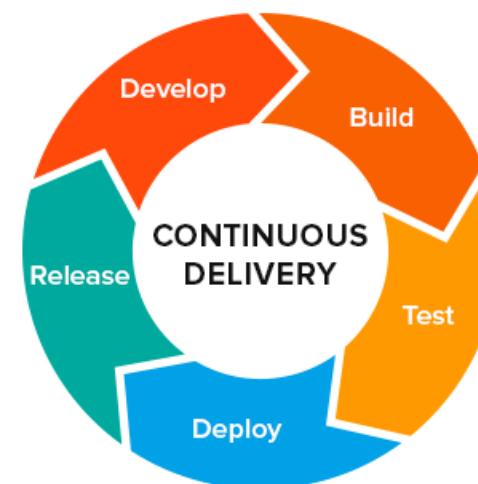
Continuous Integration

- Continuous integration เป็นวิธีปฏิบัติในการพัฒนาซอฟต์แวร์ที่นักพัฒนา รวบรวมการเปลี่ยนแปลงโค้ดลงในที่เก็บส่วนกลางอย่างสม่ำเสมอ หลังจากนั้นจะมี การสร้างและการทดสอบอัตโนมัติ เป้าหมายหลักของ Continuous integration คือ การค้นหาและแก้ไขข้อบกพร่องได้เร็วขึ้น ปรับปรุงคุณภาพซอฟต์แวร์และลด เวลาที่ใช้ในการตรวจสอบและเผยแพร่การอัปเดตซอฟต์แวร์ใหม่



Continuous Delivery

- Continuous delivery เป็นวิธีปฏิบัติในการพัฒนาซอฟต์แวร์ที่การเปลี่ยนแปลงโค้ดที่ถูกสร้างขึ้นถูกทดสอบและเตรียมพร้อมสำหรับการเผยแพร่โดยอัตโนมัติ มีการขยายและรวมอย่างต่อเนื่องโดยการปรับใช้การเปลี่ยนแปลงโค้ดทั้งหมดไปยังสภาพแวดล้อมการทดสอบและ / หรือสภาพแวดล้อมการผลิตหลังจากขั้นตอนการพัฒนา เมื่อดำเนินการจัดส่งอย่างต่อเนื่องอย่างถูกต้องนักพัฒนาจะมีซอฟต์แวร์ที่พร้อมสำหรับการปรับใช้ซึ่งผ่านกระบวนการทดสอบที่ได้มาตรฐานเสมอ



CI/CD คืออะไร?

- CI/CD เป็นวิธีการที่ช่วยให้เราสามารถสร้าง Application ให้ลูกค้าได้ใช้งานด้วย การเอาระบบอัตโนมัติไปใส่ไว้ในขั้นตอนของการพัฒนา Application ซึ่งเป็น แนวคิดที่ช่วยลดปัญหาในการ Merge Code ใหม่ ๆ ของ Developer และปัญหา ระหว่างทีม Development และทีม Operation ก่อนที่ Deploy ไปยัง Production

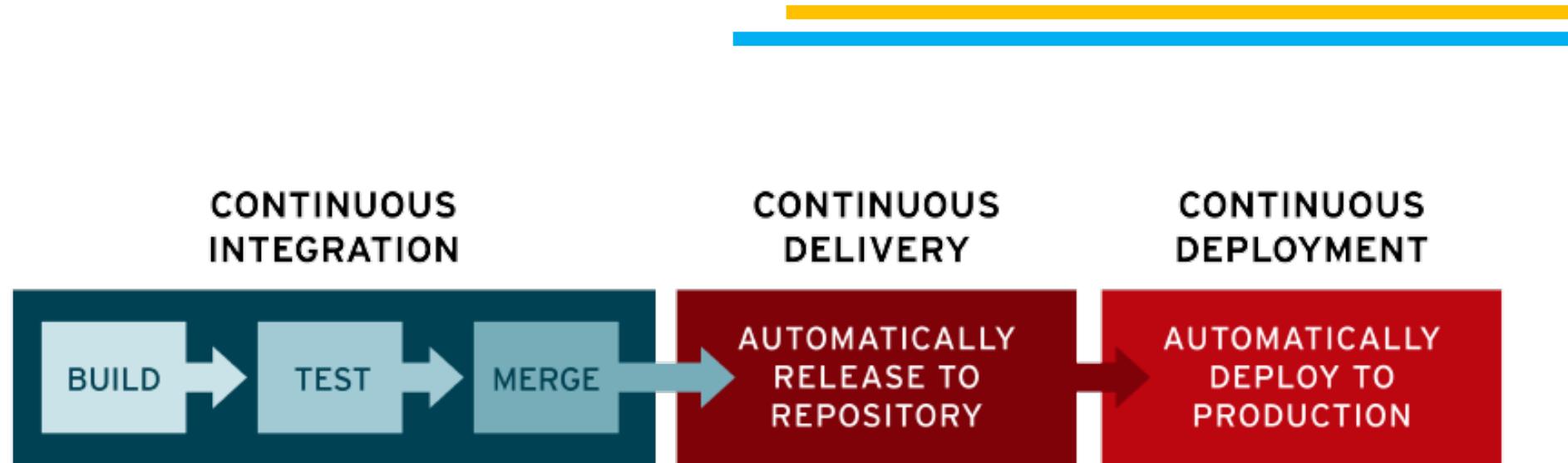
CI/CD ช่วยทำงานได้อย่างไร?

- CI/CD จะใช้ระบบอัตโนมัติและช่วยมอนิเตอร์ตลอดช่วงเวลาพัฒนา Application ตั้งแต่การนำ Code ของ Developer รวมกัน ไปจนถึงการทดสอบการใช้งาน เพื่อที่จะ Deploy ลง Production ต่อไป เรียกขั้นตอนทั้งหมดว่า CI/CD Pipeline ซึ่งช่วยให้ทีม Development และทีม Operation ทำงานด้วยกันได้อย่างต่อเนื่องแบบ Agile โดยเรียกคนกลางที่จะมาทำ CI/CD นี้ว่า DevOps หรือ Site reliability Engineer (SRE)

CI ย่อมาจาก Continuous Integration
CD มี 2 ความหมาย ได้แก่

- Continuous Delivery หรือ
- Continuous Deployment

CI/CD ช่วยทำงานได้อย่างไร?(ต่อ)



CI/CD ช่วยทำงานได้อย่างไร?(ต่อ)

- การสร้างและพัฒนา Application ในปัจจุบันจะใช้ Developer หลาย ๆ คน พร้อม ๆ กันในการพัฒนา Feature ต่าง ๆ ใน App
- แต่ละวันจะต้องเอา Code ของ Developer ที่พัฒนา Feature ต่าง ๆ แยกกัน เอามารวมกัน เรียกวันนี้ว่า Merge Day ซึ่งจะต้องใช้เวลา กำลังคน หากต้องมาแก้ไข Source Code ให้สามารถใช้ร่วมกันได้
- เหตุการณ์นี้เกิดขึ้น เพราะการทำงานของ Developer จะเป็นแบบ Isolate ที่นำ Source code ของ Application มาปรับปรุง หรือพัฒนา Feature ทำให้มีโอกาส ที่จะเกิดการ Conflict กันกับ Developer คนอื่น ๆ
- ปัญหานี้เกิดจากการที่ Developer แต่ละใช้ Integrated Development Environment (IDE) ไม่เหมือนกัน เช่น Source code editor, Debugger, Build

Continuous Integration (CI) คืออะไร มีหน้าที่อย่างไร?

- CI มีบทบาทในการช่วย Developer ให้สามารถ Merge code ที่แก้ไขส่งกลับไปที่ Pool กลาง เช่น Git Repository ซึ่งอาจจะต้องค่าให้ทำวันละครั้ง ดังนั้นเมื่อ Developer มีการเปลี่ยนแปลงเกี่ยวกับ Application ก็จะมี CI ช่วย Merge code รวมไปถึงการทำ Automate test, Unit test และ Integration test เพื่อให้มั่นใจว่าสิ่งที่ Developer เปลี่ยนแปลงไปจะไม่กระทบกับการทำงานของ Application ทำให้เกิดการทดสอบทุกอย่างตั้งแต่ Class ไปจนถึง Function ใน Module ที่แตกต่างกันแล้วนำมาประกอบเป็น Application ทำการทำ Automate test แล้วเจอปัญหารือ Conflict ระหว่าง Code เดิมกับของใหม่ CI จะช่วยให้ง่ายต่อการหา และแก้ไข Bug

Continuous Delivery (CD) คืออะไร มีหน้าที่อย่างไร?

- หลังจากที่มีการทำ Automation ในเรื่องของการ Build test, Unit test และ Integration test ในฝั่ง CI กันไปแล้ว CD ที่เป็น Continuous Delivery จะทำหน้าที่ส่ง Code ที่มีการตรวจสอบแล้วไปยัง Repository ดังนั้นถ้าต้องการให้ระบบ Continuous Delivery มีประสิทธิภาพมากที่สุด จะต้องทำระบบ CI ให้อยู่ให้ Pipeline เดียวกันกับ CD เป้าหมายของการทำ CD คือการที่พร้อมที่จะนำ Code ไป Deploy ที่ Production ได้ตลอดเวลานั้นเอง แต่ Continuous Delivery จะยังเป็น Step Manual อยู่ ไม่ได้เป็น Automation ทั้งหมด

Continuous Deployment (CD) คืออะไร มี หน้าที่อย่างไร?

- CD อีกตัวที่ต่างจาก Continuous Delivery คือ Continuous Deployment ขั้นตอนนี้เป็น ขั้นตอนสุดท้ายของ CI/CD Pipeline ซึ่งเป็นส่วนที่ขยายมาจาก Continuous Delivery ซึ่งจะ ทำหน้าที่จัดการโดยอัตโนมัติในส่วนของการนำ Code จาก Repository ไป Deploy บน Production ซึ่ง Continuous Deployment จะเหมาๆกับการใช้งานที่มีการ Deploy เป็น จำนวนมาก และมีขั้นตอน Test automation ที่ออกแบบมาเป็นอย่างดี
- โดยในทางปฏิบัติแล้ว Continuous Deployment หมายถึงการที่ Developer สามารถ Dev app ขึ้นมาแล้วไป Go Live บน Cloud ได้โดยใช้เวลาเพียงไม่กี่นาที (สมมุติว่าทดสอบด้วยระบบ Automate ผ่าน) การที่สามารถทำแบบนี้ได้จะช่วยให้ได้รับ Feedback ต่าง ๆ กลับมาไว ไม่ว่า จะเป็นจากผู้ User หรือว่า QA เมื่อ CI กับ CD มารวมกันเป็น CI/CD ก็จะช่วยให้สามารถ Deploy app โดยลดความเสี่ยงและปัญหาที่จะเกิดขึ้นได้อย่างไรก็ตาม CI/CD จะเหมาๆกับการ แก้ไขหรือเปลี่ยน Feature เล็ก ๆ น้อย ๆ มากกว่าแก้ไขทั้งหมด แต่การที่จะทำ CI/CD Pipeline ขึ้นมาจะต้องลงแรง และใช้เวลาเนื่องจากต้องวางแผนเรื่อง Automated test ซึ่งมีการเขียนที่ หลากหลายรูปแบบตามสถานการณ์ในการทดสอบ

ความแตกต่างของ CI และ CD

- CI จะเป็นระบบ Automate test สำหรับ Developer ถ้าแก้ Code แล้วผ่านระบบ CI ไปได้หมายถึง Code จะ Merge ไปยัง Repository สำเร็จ CI เข้ามาตอบโจทย์ในกรณีที่ Developer มีการ Dev แยกส่วนกันแล้วต้องทำ Code มารวมกัน ทำให้อาจเกิดปัญหาเรื่อง Conflict กันได้
- CD (Continuous Delivery) ส่วนใหญ่จะหมายถึงการที่ Developer สามารถ Upload Code เพื่อไปทดสอบหา Bug ที่ Repository เช่น GitHub ได้โดยอัตโนมัติ ซึ่งทำให้ทีม Operation สามารถนำไป Deploy ได้เลย ช่วยให้ลดปัญหาเรื่องของการสื่อสารระหว่าง Developer และทีม Business สุดท้ายแล้ว จุดประสงค์ของการทำ Continuous Delivery คือการทำให้มั่นใจว่าเราจะใช้แรงหรือความพยายามที่น้อยที่สุดในการ Deploy Code ใหม่ ๆ

ความแตกต่างของ CI และ CD(ต่อ)

- CD (Continuous Deployment) หมายถึง การปล่อย Code จาก Repository ไปยัง Production โดยอัตโนมัติ ซึ่งจะถูกใช้งานโดยลูกค้า การทำแบบนี้ช่วยลดการทำงานของทีม Operation ในการทำระบบแบบ Manual ซึ่งอาจทำให้การ Deploy ล่าช้าได้

ประโยชน์ของ CI/CD

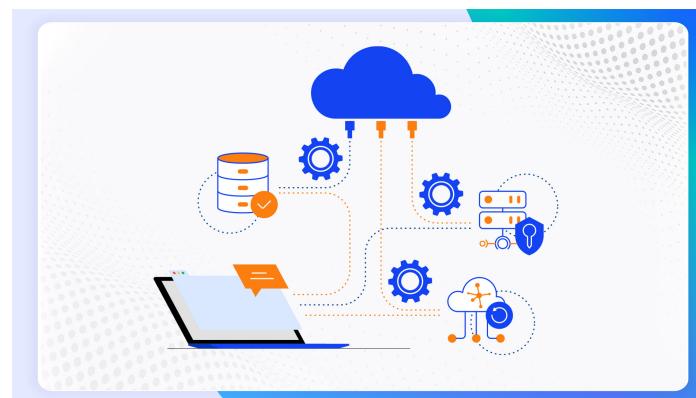
- **ลดปัญหาเรื่องการแก้ไข Code** – เนื่องจากมีระบบที่คอยนำ Code มารวมกันแล้วทดสอบให้
- **ลดปัญหาจากการทำงานแยกส่วนกัน** – จากเหตุผลเดียวกับด้านบน
- **ลดระยะเวลาในการส่งมอบงาน** – ระบบเป็นผู้ทดสอบให้ ทำให้ไม่ต้องเสียเวลาให้คนมาตรวจสอบแบบ Manual
- **เพิ่มความน่าเชื่อถือในการทดสอบ** – เนื่องเวลาแก้ไข Feature จะแก้ไขแบบเล็ก ๆ ทำให้เวลาเปลี่ยนแปลงระบบ จะทดสอบด้วยเวลาไม่นาน และมีความแม่นยำมากกว่าการเริ่มทำใหม่ตั้งแต่ต้น
- **ลดค่าใช้จ่าย** – เนื่องจากเป็นระบบอัตโนมัติทำให้ Flow การทำงานบางส่วนไม่ต้องใช้จำนวนคนในการ Monitor
- **ดูแลง่าย Update ง่าย** – การทำ CI/CD ช่วยทำให้สามารถตรวจสอบหาปัญหาได้ในทุก ๆ ขั้นตอน ไม่ว่าจะเป็นในช่วงแรกของการ Build หรือการ Update จาก App เดิมที่มีอยู่ โดย Bug ที่เกิดขึ้นจะถูกตรวจสอบได้ง่ายและตรงจุด ทำให้สามารถ Maintenance และ Update ได้ง่าย

เครื่องมือสำหรับการทำ CI/CD

- CI/CD tool ที่ช่วยให้ทีมงานสามารถทำ Automate ในเรื่องของการ Develop, Deploy และ Test มีอยู่จำนวนมาก บางตัวเหมาะสมกับการทำ CI บางตัวเหมาะสมกับ CD เป็นพิเศษ สำหรับเครื่องมือสำหรับทำ CI/CD ยอดนิยม คือ Jenkins แต่ก็มีเครื่องมืออื่น ๆ ให้ใช้งานได้จากหลายช่องทาง ไม่ว่าเป็นจาก Public Cloud เจ้าต่าง ๆ หรือจะเป็น GitLab, CircleCI, Travis CI, Atlassian Bamboo หรืออื่น ๆ นอกจากนี้ก็ยังมีบางส่วนที่เป็นเครื่องมือที่ไม่ได้ครอบคลุมระบบ CI/CD ทั้งหมด แต่เป็นเครื่องมือสำหรับ Config การทำ Automation เช่น Ansible, Chef และ Puppet หรือจะเป็นเครื่องมือที่ใช้สำหรับทำ Container เช่น Docker และ rkt และตัวจัดการ Container เช่น Kubernetes

Microservices

- สถาปัตยกรรมไมโครเซอร์วิสเป็นวิธีการออกแบบเพื่อสร้างแอปพลิเคชันเดียวเป็นชุดของบริการขนาดเล็ก แต่ละบริการทำงานในกระบวนการของตนเองและสื่อสารกับบริการอื่น ๆ ผ่านอินเทอร์เฟซที่กำหนดไว้อย่างดีโดยใช้กลไกที่มีลักษณะ lightweight ซึ่งโดยทั่วไปจะเป็นอินเทอร์เฟซการเขียนโปรแกรมแอปพลิเคชัน (API) ที่ใช้ HTTP ไมโครเซอร์วิสระบุรุษจากความสามารถทางธุรกิจ แต่ละบริการมีขอบเขตเพื่อวัตถุประสงค์เดียวโดยสามารถใช้เฟรมเวิร์กหรือภาษาการเขียนโปรแกรมที่แตกต่างกันเพื่อเขียนไมโครเซอร์วิสและปรับใช้อย่างอิสระเป็นบริการเดียวหรือเป็นกลุ่มบริการ



ความหมายและความสำคัญของ Microservices

- Microservices คือแนวคิดการสร้างแอปพลิเคชันที่ใช้หลายบริการหรือส่วนประกอบเล็ก ๆ แยกจากกันเพื่อเพิ่มประสิทธิภาพในการพัฒนาและการปรับปรุง แต่ละบริการมีการทำงานอิสระกัน โดยมีการเชื่อมต่อกันผ่านระบบ API หรือโปรโตคอลต่าง ๆ

ข้อดีของการใช้งาน Microservices

- สามารถพัฒนาระบบได้เป็นส่วนๆ โดยทีมพัฒนาแต่ละทีมไม่จำเป็นต้องรู้ถึงรายละเอียดของระบบทั้งหมด
- การทดสอบและปรับปรุงระบบจะสะดวกและรวดเร็ว
- ระบบที่มีการแบ่งเป็นส่วนๆ จะเป็นไปตามหลักการ SOA (Service Oriented Architecture) ทำให้ง่ายต่อการมีความยืดหยุ่นในการเพิ่มหรือลดบริการในอนาคต
- สามารถใช้เทคโนโลยีและภาษาโปรแกรมต่าง ๆ ที่เหมาะสมกับแต่ละบริการได้
- Microservices เป็นแนวคิดที่สำคัญสำหรับการพัฒนาระบบซอฟต์แวร์ในปัจจุบัน ซึ่งจะช่วยเพิ่มประสิทธิภาพ ความยืดหยุ่น และความสามารถในการจัดการระบบให้มีประสิทธิภาพมากยิ่งขึ้น

องค์ประกอบสำคัญในการออกแบบสถาปัตยกรรม Microservices

1. **การแบ่งความรับผิดชอบ (Responsibility separation)** – การแบ่งบริการเป็นส่วนย่อย ๆ ที่มีความรับผิดชอบเฉพาะ เพื่อให้ง่ายต่อการพัฒนาและปรับปรุงแต่ละบริการ
2. **การสื่อสารระหว่างบริการ (Inter-service communication)** – การใช้การสื่อสารระหว่างบริการในรูปแบบ API หรือโปรโตคอลต่าง ๆ เพื่อเชื่อมต่อระบบของแต่ละบริการ
3. **การรวมเข้าด้วยกันและการยืดหยุ่น (Integration and scalability)** – การรวมบริการแต่ละตัวเข้าด้วยกันให้เป็นระบบโดยใช้เทคโนโลยีที่เหมาะสม และสามารถขยายขนาดได้ตามความต้องการ
4. **กระบวนการพัฒนาและการทดสอบ Microservices แบบแยกตัว (Independent development and testing)** – การพัฒนาแต่ละบริการแยกจากกันเพื่อเพิ่มประสิทธิภาพในการพัฒนาและทดสอบ แต่ยังคงใช้เครื่องมือที่เหมือนกันในการพัฒนาและทดสอบ
5. **การใช้งาน CI/CD (Continuous Integration/Continuous Delivery)** – การใช้กระบวนการพัฒนาแบบ CI/CD เพื่อให้การปรับปรุงและการอัปเดตระบบ Microservices เป็นไปอย่างรวดเร็วและปลอดภัย

เครื่องมือและเทคโนโลยีที่ใช้ในการพัฒนาระบบ Microservices

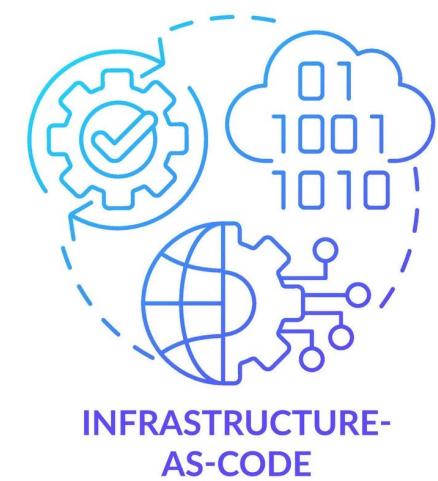
1. ภาษาโปรแกรม (Programming languages): ภาษาที่มักจะใช้กันในการพัฒนาระบบ Microservices ได้แก่ Java, Python, Node.js, Go, Ruby, C#, Kotlin, Scala, PHP, Swift ฯลฯ

2. เฟรมเวิร์คและไลบรารี (Frameworks and libraries): เฟรมเวิร์คและไลบรารีที่มักจะใช้กันในการพัฒนาระบบ Microservices ได้แก่ Spring Boot, Micronaut, Quarkus, Flask, Django, Express.js, Nest.js, Ruby on Rails, ASP.NET Core, Vapor, ฯลฯ

3. แพลตฟอร์มและเทคโนโลยีคลาวด์ (Cloud platforms and technologies): แพลตฟอร์มและเทคโนโลยีคลาวด์ที่มักจะใช้กันในการพัฒนาระบบ Microservices ได้แก่ Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud, Heroku, Docker, Kubernetes, Apache Kafka, RabbitMQ, ฯลฯ

Infrastructure as Code

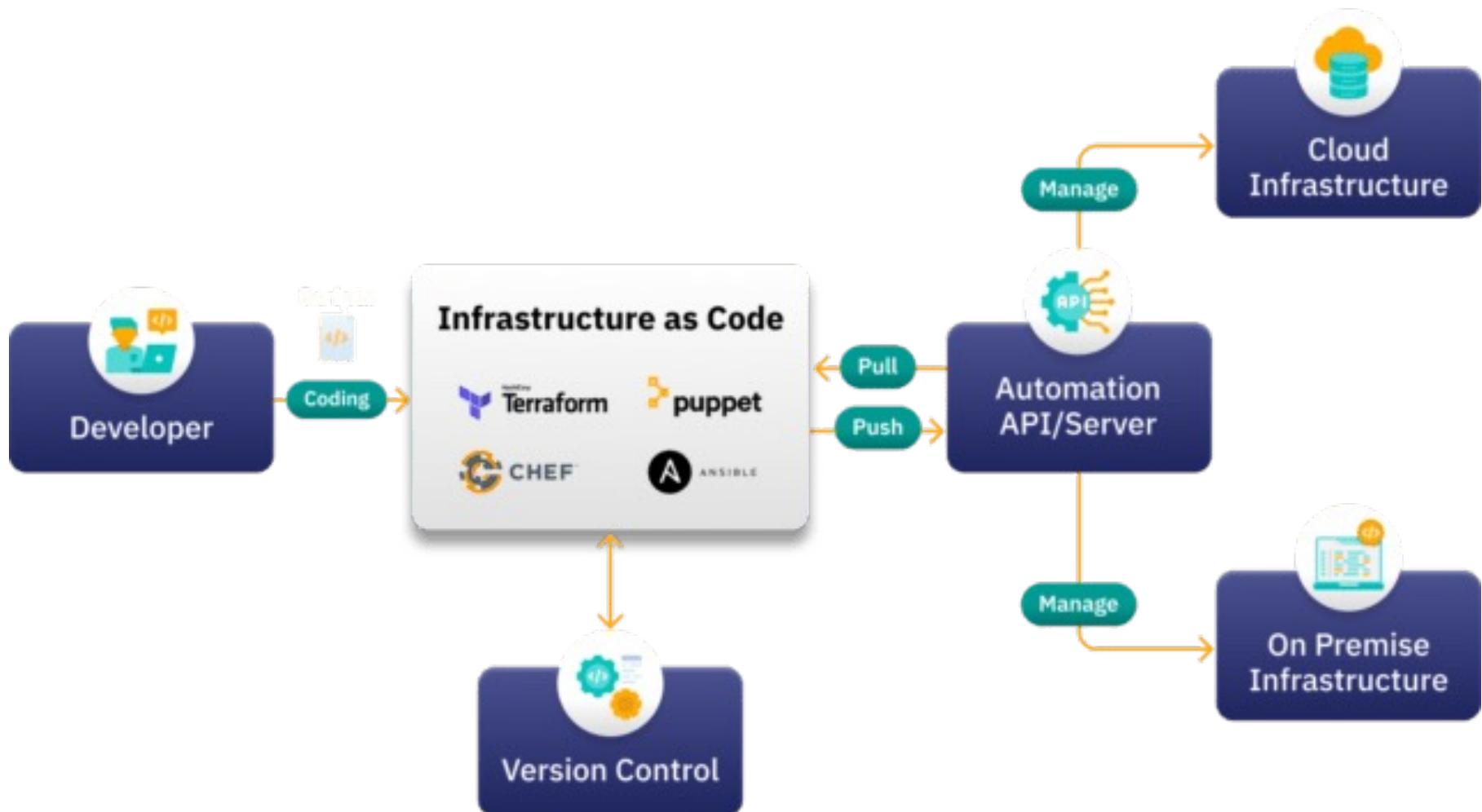
- โครงสร้างพื้นฐานเป็นรหัสเป็นแนวทางปฏิบัติที่โครงสร้างพื้นฐานได้รับการจัดเตรียมและจัดการโดยใช้รหัสและเทคนิคการพัฒนาซอฟต์แวร์ เช่น การควบคุมเวอร์ชันและการรวมอย่างต่อเนื่องโมเดลที่ขับเคลื่อนด้วย API ของระบบคลาวด์ช่วยให้นักพัฒนาและผู้ดูแลระบบสามารถโต้ตอบกับโครงสร้างพื้นฐานโดยทางโปรแกรม และในวงกว้างแทนที่จะต้องตั้งค่าและกำหนดค่าทรัพยากรด้วยตนเอง ดังนั้นวิศวกรสามารถเชื่อมต่อกับโครงสร้างพื้นฐานโดยใช้เครื่องมือที่ใช้รหัสและปฏิบัติต่อโครงสร้างพื้นฐานในลักษณะที่คล้ายกับวิธีที่พวกลเข้าทำกับรหัสแอปพลิเคชัน เนื่องจากถูกกำหนดโดยโค้ดโครงสร้างพื้นฐานและเซิร์ฟเวอร์จึงสามารถใช้งานได้อย่างรวดเร็วโดยใช้รูปแบบมาตรฐานอัปเดตด้วยแพตช์และเวอร์ชันล่าสุดหรือทำซ้ำได้ด้วยวิธีทำซ้ำได้



Infrastructure as Code คืออะไร?

- Infrastructure as Code หรือ IaC (ไอ-เอ-ซี) เป็นเทคนิคที่ใช้ “การเขียนโค้ด” เพื่อสร้างและจัดการตั้งค่าพื้นฐานของระบบในรูปแบบ Automation ซึ่งผู้ติดตั้งสามารถนำ Script ที่เขียนกลับมาทำซ้ำได้ หรือมีการปรับปรุงเปลี่ยนแปลง Config ใหม่ก็ไม่ใช่เรื่องยุ่งยาก ทำให้ช่วยลดขั้นตอน ลดเวลาและลดข้อผิดพลาดที่เกิดจาก การตั้งค่าตกรหล่นในการติดตั้งแบบ Manual รูปแบบเดิม ๆ
- IaC ถูกพูดถึงมากขึ้นอย่างต่อเนื่องในกลุ่มนักพัฒนาซอฟต์แวร์ และ IaC จะไม่ใช่แค่ เรื่องของทีม Infra เท่านั้น แต่ทีม Software Developer ก็ต้องรู้และต้องเร่งศึกษา ก่อนที่จะคุยกับองค์กร Tech ชั้นนำอีก ไม่รู้เรื่อง

Infrastructure as Code คืออะไร?(ต่อ)



ประโยชน์ของ Infrastructure as Code

1. ช่วยลดเวลา Provisioning Infra ในรูปแบบ Automation : ขั้นตอนในการ Setting ที่ยุ่งยากมายสามารถแปลงเป็นรูปแบบ Code เก็บเป็น Script ทำให้สามารถทำ Automate provisioning ได้ตลอดเวลา ไม่ว่าจะเป็น Dev, UAT หรือ Production ก็สามารถทำได้จะ Scale ทีหลังก็ง่าย

2. จบปัญหา Config ซ้ำซ้อน สร้างความเป็นมาตรฐาน : จากเดิมที่ทีมติดตั้งต้องคอย Config แบบ Manual ความเสี่ยงที่จะตั้งค่าไม่ตรงกัน หรือ เกิดโอกาสผิดพลาดสูง อีกทั้งหากทำนานๆ แล้ว อาจทำให้เกิดการลืม หากไม่มีการบันทึกไว้ดีพอ การทำ Infrastructure as Code จึงตอบโจทย์เรื่องนี้มาก จะ Provisioning Infra กี่ครั้ง ระบบก็ Config ได้เหมือนเดิม

3. Review และทำ Automated Testing ได้เหมือน Code : เมื่อแปลง Infrastructure ให้มาอยู่ในรูปแบบของ Code ได้ ย่อมต้องสามารถ Review Version ของ Infrastructure ได้เหมือนการ Review code ทำให้สามารถ track ปัญหาย้อนหลังได้ หรือจะเก็บเป็น Document ไว้ Audit ก็เป็นสิ่งที่ควรทำ ว่าทีมเซ็ตอัประบบอย่างไร หากใครจะลาออกไปก็ไม่ต้องกลัว เพราะข้อมูลทั้งหมดอยู่ในไฟล์แล้ว

ประโยชน์ของ Infrastructure as Code(ต่อ)

4. จัดการ Server หรือ Cloud จำนวนมากได้ใน “คลิกเดียว” : คำว่า “คลิกเดียว” หมายถึง ถ้าหาก Config นั้นเสถียรและนิ่งแล้ว เมื่อต้องการ Scale Server หรือ ติดตั้งเพิ่ม ก็สามารถทำได้ง่าย เพราะแค่ Run ไฟล์ Script ครั้งเดียว ก็ได้ระบบหน้าตา Config เดิม ใช้งานได้ทันที ลดเวลา ลดขั้นตอนไปได้

5. เป็น Global Standard ที่องค์กร Tech ชั้นนำใช้พัฒนาระบบ : ปัจจุบันการทำ Infrastructure as Code ได้เป็นมาตรฐานที่ต้องทำ หากองค์กรนั้นเป็น Tech ที่มีระบบซับซ้อน โดยหลายองค์กรในไทยก็ได้ดำเนินการกันมาแล้วระยะหนึ่ง อย่างเช่น ธนาคารทุกวันนี้ก็ได้กำหนดไว้ว่า ทีมจาก Outsource ไหนเข้ามาต้องทำ Infrastructure as Code กันทุกที่

Infrastructure as Code จะเข้ามาช่วยให้ชีวิต การทำงานของทีมนักพัฒนาดีขึ้นได้อย่างไร?

- ในกรณีที่ทีม Infra ต้องเช็คระบบและตั้งค่าระบบต่าง ๆ ไม่ว่าจะเป็น OS, Software, Database และอื่น ๆ อีกมากmany แต่เดิมขั้นตอนเหล่านี้จะต้องทำในรูปแบบ Manual ตาม Checklist และ Policy ที่กำหนดไว้ หากเพียงแค่ 1-2 เครื่อง ก็อาจจะไม่ใช่เรื่องใหญ่ แต่ถ้าเป็นหลัก 10 หรือ หลัก 100 เครื่อง ทีมติดตั้งจะต้องใช้เวลาในการทำงานนาน
- ปัญหาที่เกิดตามมาถ้าต้องติดตั้งหลายเครื่อง อาจมีโอกาสเกิดความผิดพลาดจาก การติดตั้ง หรือ ตกหล่นจากการทำด้วยวิธี Manual ได้ อีกทั้งเวลาจะปรับปรุง Version ก็ง่าย ไม่ว่าต้องรีเซ็ตใหม่กี่ครั้ง ก็ช่วยให้ทีมพัฒนาสามารถทำได้ทันที เพียงแค่ปรับโค้ดเล็กน้อย และก็ Run Script ให้ทำงานแบบ Automation

Monitoring and Logging

- องค์กรจะตรวจสอบเมตริกและบันทึกเพื่อดูว่าประสิทธิภาพของแอปพลิเคชันและโครงสร้างพื้นฐานส่งผลต่อประสบการณ์ของผู้ใช้ผลิตภัณฑ์อย่างไร ด้วยการจัดหมวดหมู่และวิเคราะห์ข้อมูล และบันทึกที่สร้างโดยแอปพลิเคชันและโครงสร้างพื้นฐาน องค์กรจะเข้าใจว่าการเปลี่ยนแปลงหรือการอัปเดตส่งผลกระทบต่อผู้ใช้อย่างไรทำให้ข้อมูลเชิงลึกเกี่ยวกับสาเหตุของปัญหาหรือการเปลี่ยนแปลงที่ไม่คาดคิด การตรวจสอบที่ใช้งานอยู่มีความสำคัญมากขึ้นเนื่องจากบริการต้องพร้อมใช้งานตลอด 24 ชั่วโมงทุกวันและเมื่อความถี่ในการอัปเดตแอปพลิเคชันและโครงสร้างพื้นฐานเพิ่มขึ้น การสร้างการแจ้งเตือนหรือการวิเคราะห์ข้อมูลนี้แบบเรียลไทม์ยังช่วยให้องค์กรตรวจสอบบริการของตนในเชิงรุกมากขึ้น



Communication and Collaboration

- การสื่อสารและการทำงานร่วมกันที่เพิ่มขึ้นในองค์กรเป็นหนึ่งในแรงมุ่งทางวัฒนธรรมที่สำคัญของ DevOps การใช้เครื่องมือ DevOps และระบบอัตโนมัติของกระบวนการจัดส่งซอฟต์แวร์สร้างการทำงานร่วมกันโดยการรวมรวมเวิร์กโฟล์ว และความรับผิดชอบของการพัฒนาและการดำเนินงานเข้าด้วยกัน ยิ่งไปกว่านั้นทีมเหล่านี้ยังสร้างบรรทัดฐานทางวัฒนธรรมที่แข็งแกร่งเกี่ยวกับการแบ่งปันข้อมูลและอำนวยความสะดวกในการสื่อสารผ่านการใช้อุปกรณ์เช่นแชทระบบติดตามปัญหา หรือโครงการและวิกิ สิ่งนี้ช่วยเร่งการสื่อสารระหว่างนักพัฒนาการดำเนินงานและแม้แต่ทีมอื่น ๆ เช่นการตลาดหรือการขายทำให้ทุกส่วนขององค์กรสามารถปรับเปลี่ยนอย่างรวดเร็วและมีประสิทธิภาพยิ่งขึ้น



DevOps Tools

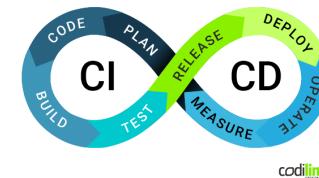
Git

- Gitlab
- Bitbucket
- Github



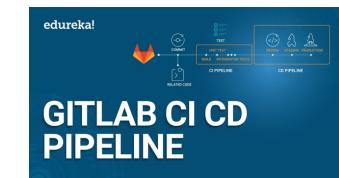
CI/CD Tools

- Jenkins
- Github Action
- Circle CI/CD
- Azure DevOps
- Gitlab Pipeline



Platform

- Kubernetes
- Docker



DevOps Tools(ព័ត៌មាន)

Monitoring

- Prometheus
- Grafana



Logs

- ELK Stack
- Promtail
- Loki

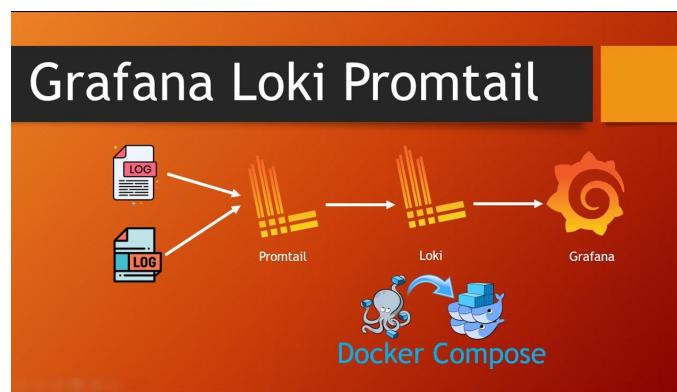


Cloud

- Google Cloud
- Huawei Cloud
- AWS
- Azure



Google Cloud



Developer คือใคร ทำหน้าที่อะไร พร้อมรู้จักกับ Developer ทั้ง 4 แบบ

- Developer หรือที่หลายคนเรียกว่า Software Developer คือ นักพัฒนาซอฟต์แวร์ ซึ่งในการพัฒนาสินค้า หรือบริการขึ้นมาหนึ่งอย่าง มีหลายส่วนที่ต้องทำ อาชีพ Developer จึงถูกแบ่งออกเป็นหลายตำแหน่งย่อย ๆ เช่น Front End Developer, Back End Developer, Full Stack Developer หรือ DevOps Engineer จนหลายคนมักสงสัยว่าแต่ละตำแหน่งมีหน้าที่แตกต่างกันอย่างไร

Developer คืออะไร

- Developer หรือ Software Developer คือ นักพัฒนาซอฟต์แวร์ ผู้ที่ทำการพัฒนา Digital Product ต่าง ๆ ด้วยการ Coding หรือการเขียนโปรแกรม ไม่ว่าจะเป็นการพัฒนาเว็บไซต์ แอปพลิเคชัน หรืออื่น ๆ ที่ต้องอาศัยซอฟต์แวร์ในการทำงาน

หน้าที่ของ Developer

- Developer มีหน้าที่พัฒนาสินค้า หรือบริการหนึ่ง ๆ ผ่านการเขียนโค้ด แต่การเขียนโค้ดก็ไม่ใช่หน้าที่ทั้งหมด สิ่งที่ต้องทำนั้นต้องคำนึงถึง คือ การสามารถพัฒนาสินค้า หรือบริการได้ตามที่กำหนดไว้ จึงมีหน้าที่ช่วยคิด วางแผน ออกแบบ ตรวจสอบ และปรับปรุงร่วมกับทีมอื่น ๆ เพื่อให้ได้ของที่ตรงตามความต้องการในเวลาที่เหมาะสม ดังนั้น นอกจากทักษะด้าน Coding แล้ว ยังควรมีทักษะการสื่อสาร และความรู้ในเรื่อง Product และ Business ด้วยเช่นกัน
- อย่างไรก็ตาม ตำแหน่งเหล่านี้มักจะเป็นตำแหน่งที่ถูกทดแทนได้ เช่น ตำแหน่ง Programmer ก็สามารถพัฒนาสินค้า หรือบริการได้ ดังนั้นบางองค์กรอาจมีหน้าที่ที่แตกต่างกันออกไป

ประเภทของ Developer

- โดยทั่วไปแล้ว Developer จะแบ่งได้เป็น 4 ตำแหน่งหลัก ๆ ได้แก่ Front End Developer, Back End Developer, Full Stack Developer และ DevOps Engineer เนื่องจากการพัฒนา Software มีความซับซ้อนเป็นอย่างมาก โดยแต่ละตำแหน่งที่แยกย่อยลงมาใน ก็จะมีหน้าที่เฉพาะเจาะจงมากยิ่งขึ้น
 - Front End Developer
 - Back End Developer
 - Full Stack Developer
 - DevOps Engineer

Front End Developer

- คือ ผู้พัฒนาเว็บไซต์ หรือแอปพลิเคชันที่รับผิดชอบในการพัฒนาส่วนที่ติดต่อกับผู้ใช้งาน ซึ่งเป็นส่วนที่ผู้ใช้งานสามารถมองเห็น และสื่อสารกับระบบได้โดยตรง ทำหน้าที่ค่อยควบคุมดูแล และสร้างเว็บไซต์ให้มีหน้าตาและใช้งานได้ถูกต้องตามที่ออกแบบไว้
- โดยทักษะที่ต้องมี เช่น
 - การใช้ภาษา Front End Language โดยมักนิยมใช้ภาษา HTML, CSS และ JavaScript
 - การใช้งาน Front End Libraries and Frameworks เช่น React, Flutter, Vue.js เป็นต้น
 - การใช้งาน Version Control/ Git
 - การทำ Responsive Design
 - การทำ Progressive Web Apps (PWA)
 - การทำ Testing and Debugging
 - การตรวจสอบ และปรับปรุง Web Performance
 - การใช้งาน Command Line

Back End Developer

- คือ ผู้พัฒนาเว็บไซต์ หรือแอปพลิเคชันที่รับผิดชอบในการพัฒนาส่วนระบบหลังบ้าน โดยสร้างและควบคุมดูแลระบบที่ทำงานภายใต้เว็บไซต์ เช่น การเชื่อมต่อกับฐานข้อมูล, การจัดการกับข้อมูลผู้ใช้, การสร้างฟังก์ชันการทำงานต่างๆ เพื่อให้ผู้ใช้งานสามารถใช้งานเว็บไซต์ได้อย่างมีประสิทธิภาพตรงตามที่ออกแบบไว้
- โดยทักษะที่ตำแหน่งนี้ควรมี เช่น
 - การใช้ภาษา Back End Language เช่น C#, Go, Java, PHP, Python เป็นต้น
 - การใช้งาน Back End Libraries and Frameworks เช่น .NET, Node.js, ROR ขึ้นอยู่กับภาษาที่เลือกใช้
 - การทำ Database
 - การทำ API
 - การใช้ Version Control/ Git
 - การทำ Testing and Debugging
 - การทำ Cyber Security
 - การใช้งาน Command Line

Full Stack Developer



- គឺជាអ្នកដំឡើងទាំងអស់ដែលមានភាពជាបន្ទាល់ពីការរចនាថាមប្រព័ន្ធនិងការអនុវត្តន៍យកពីការងារ។ គឺជាអ្នកដំឡើងទាំងអស់ដែលមានភាពជាបន្ទាល់ពីការរចនាថាមប្រព័ន្ធនិងការអនុវត្តន៍យកពីការងារ។
- គឺជាអ្នកដំឡើងទាំងអស់ដែលមានភាពជាបន្ទាល់ពីការរចនាថាមប្រព័ន្ធនិងការអនុវត្តន៍យកពីការងារ។ គឺជាអ្នកដំឡើងទាំងអស់ដែលមានភាពជាបន្ទាល់ពីការរចនាថាមប្រព័ន្ធនិងការអនុវត្តន៍យកពីការងារ។

DevOps Engineer

- DevOps Engineer คือ คนที่ทำหน้าที่เป็นตัวเชื่อมระหว่างสองฝ่าย คือฝ่าย Development และฝ่าย Operation เป็นคนที่ค่อยสนับสนุนกระบวนการพัฒนาซอฟต์แวร์ให้สามารถทำงานได้โดยอัตโนมัติเท่าที่จะสามารถทำได้ เช่น การวางแผน Infrastructure, การทำ CI/CD ซึ่งช่วยให้การทำงานของทั้งสองฝ่ายมีความเข้ากัน และรวดเร็วมากขึ้น
- โดยทักษะที่สำคัญมี เช่น
 - การใช้งาน Programming Language เพื่อให้การทำงานร่วมกับ Developer อื่น ๆ ราบรื่นมากขึ้น
 - การจัดการกับ Operating System (OS)
 - การใช้งาน Command Line
 - การทำ Networking Protocols
 - การใช้งาน Container
 - การเตรียม Infrastructure Provisioning
 - การทำ CI/CD
 - การทำ Logs Management
 - การทำ Infrastructure Monitoring
 - การทำ Application Monitoring
 - การใช้งาน Cloud