INFOPRESS

Practical DEVOPS and Cloud Engineering

Best Practices Workshop
Version Control & Microservices

คู่มือ Upskill & Reskill สำหรับคนที่อยากเป็น Cloud DevOps Engineer หนึ่งในอาชีพดาวรุ่งด้านเทคโนโลยีดิจิตอล

SOURCE CODE : SERAZU.COM

ผู้แต่ง อ.ดร.ณัฐโชติ พรหมฤทธิ์
บรรณาธิการ ภีรพล คชาเจริญ

IDC
PREMIER

มีเพียง “**ความรู้**” เท่านั้นที่มนุษย์ใช้พลิก “**โลก**”
และเปลี่ยนชีวิต เราจึงสร้างสรรค์ และส่งมอบ “**ความรู้**”
ในรูปแบบที่ดีกว่า เพื่อให้คนไทย “**เรียนรู้**” ได้ตลอดชีวิต

Only “**Knowledge**” can help human
change “**The World**” and “**Their Lives**”.
With this truth, it drives us to deliver
“**Knowledge**” for Thai being able to
“**Learn**” better everyday.

Think Beyond
INFOPRESS
Read comics
Little HEART

Practical DevOps and Cloud Engineering

Writer	ดร.ณัฐโชติ พรหมฤทธิ์
Editor	ภีรพล คชาเจริญ
Graphic Designers	ชวนันท์ รัตนะ, กมลชนก ชัยสาครสมุทร
Page Layout	สิริลักษณ์ วาระเลิศ
Proofreaders	สุนทรี บรรลือศักดิ์, เกษรา พรวัฒนมงคล
Publishing Coordinators	วรพล ณธิกุล, สุพัตรา อาจปรุ, วัชรพงศ์ ยงปัญญาสกุล

บริษัท ไอดีซี พรีเมียร์ จำกัด จัดตั้งขึ้นเพื่อเผยแพร่ความรู้ที่มีคุณภาพสู่ผู้อ่านชาวไทย เรายินดีรับงานเขียนของนักวิชาการและนักเขียนทุกท่าน ท่านผู้สนใจกรุณาติดต่อผ่านทางอีเมลที่ infopress@idcpremier.com หรือทางโทรศัพท์หมายเลข 0-2962-1081 (อัตโนมัติ 10 คู่สาย) โทรสาร 0-2962-1084

สร้างสรรค์โดย

INFOPRESS

พิมพ์ครั้งที่ 1 ธันวาคม 2563

ข้อมูลทางบรรณานุกรม
ดร.ณัฐโชติ พรหมฤทธิ์
Practical DevOps
and Cloud Engineering
นนทบุรี : ไอดีซีฯ, 2563
432 หน้า

1. การเขียนโปรแกรมโดยใช้ภาษาโปรแกรมเฉพาะชนิด
I ชื่อเรื่อง
005.262
ISBN 978-616-487-174-8
ราคา 545 บาท

พิมพ์และจัดจำหน่ายโดย

IDC PREMIER

บริษัท ไอดีซี พรีเมียร์ จำกัด
200 หมู่ 4 ชั้น 19 ห้อง 1901
จัสมินอินเตอร์เนชั่นแนลทาวเวอร์
ถ.แจ้งวัฒนะ อ.ปากเกร็ด จ.นนทบุรี 11120
โทรศัพท์ 0-2962-1081 (อัตโนมัติ 10 คู่สาย)
โทรสาร 0-2962-1084

สมาชิกสัมพันธ์
โทรศัพท์ 0-2962-1081-3 ต่อ 121
โทรสาร 0-2962-1084

ร้านค้าและตัวแทนจำหน่าย
โทรศัพท์ 0-2962-1081-3 ต่อ 112-114
โทรสาร 0-2962-1084

PREFACE

การสร้างและดูแลรักษาซอฟต์แวร์ มีความแตกต่างจากการสร้างและดูแลรักษาผลิตภัณฑ์ประเภทอื่น เนื่องจากธรรมชาติของผลิตภัณฑ์ที่เป็นซอฟต์แวร์ มักมีความเปลี่ยนแปลงเกิดขึ้นบ่อย และต้องแข่งขันกับคู่แข่ง รวมทั้งมีการปรับเปลี่ยนไปตามเสียงสะท้อนของผู้ใช้งาน ซึ่งโดยส่วนใหญ่การปรับเปลี่ยนจะเป็นไปในลักษณะทั้งที่มีการแก้ไขปัญหาเก่าและเพิ่มเติมความสามารถใหม่ๆ โดยที่ยังคงให้บริการได้ในขณะที่มีการปรับเปลี่ยนเพื่อไม่ให้ผู้ใช้รู้สึกติดขัด

และเพื่อให้การทำงานของระบบหรือซอฟต์แวร์เป็นไปโดยราบรื่น สามารถรองรับการเปลี่ยนแปลงหรือเพิ่มเติมได้ตลอดเวลา จึงต้องอาศัยทั้งทักษะในแง่ของการพัฒนา (Development) และการดูแล (Operation) รวมทั้งต้องใช้เครื่องมือที่มีความทันสมัยในการบริหารจัดการ จึงทำให้บุคคลที่มีความรู้ความสามารถในเชิง DevOps หรือ DevOps Engineer เป็นกำลังคนที่ตลาดแรงงานมีความต้องการสูงในปัจจุบัน ซึ่งหน้าที่หลักๆ ของ DevOps Engineer คือ เป็นผู้เขียน CI/CD Pipeline Script, Dockerfile, docker-compose.yml วางสภาพแวดล้อมในการพัฒนาโปรแกรม รวมทั้งคอยดูแล Infrastructure และ Config Cluster (เช่น Docker Swarm และ Kubernetes) เป็นต้น

นอกจากนี้ DevOps Engineer ยังเป็นผู้ Monitor ระบบหรือซอฟต์แวร์ที่ Deploy บน Cloud Server ซึ่งเมื่อพบปัญหาจะต้องรีบแจ้ง Developer ให้หาทางแก้ไขอย่างทันท่วงที ดังนั้น หน้าที่ของ DevOps Engineer คือ ให้การสนับสนุนการทำงานของ Developer ในทำนองเดียวกันกับ System Admin แต่มีขอบเขตกว้างกว่าการทำหน้าที่ในงาน System Operation ตามรูปแบบเดิม

จากความต้องการกำลังคนในงานทางด้าน DevOps Engineer ดังกล่าว ผมจึงได้หารือกับ อ.ดร.สัจจาภรณ์ ไวจรรยา อดีต IT Development Department Manager บริษัท TARAD Dot Com Group Co., Ltd. และ Operational Readiness Team Lead (Manager) บริษัท Total Access Communication PLC (DTAC) และคุณชาญศิลป์ ชิ้นประเสริฐ CTO Nipa.cloud ในการออกแบบเนื้อหาของหนังสือเล่มนี้ เพื่อให้ผู้อ่านได้มองเห็นภาพรวมของงานทางด้าน DevOps และหลังจากการฝึกปฏิบัติและทำ Workshop แล้ว หวังว่าผู้อ่านจะมีทักษะขั้นพื้นฐานที่สามารถนำไปใช้กับการทำงานในภาคอุตสาหกรรมดิจิตอลได้จริง

อ.ดร.ณัฐโชติ พรหมฤทธิ์
หัวหน้าภาควิชาคอมพิวเตอร์
คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

EDITOR'S NOTE

เมื่อเทคโนโลยีใหม่ๆ เกิดขึ้นตลอดเวลา ความรู้และทักษะที่ได้รับจากภาคการศึกษา จึงมักจะมีความล่าช้ากว่าในภาคอุตสาหกรรม ที่มีการปรับตัวให้ทันสมัยอยู่เสมอ โดยเฉพาะในภาคอุตสาหกรรมการพัฒนาผลิตภัณฑ์ซอฟต์แวร์ ซึ่งเป็นวงการหนึ่งที่มีการเปลี่ยนแปลงอย่างรวดเร็ว โดยเฉพาะซอฟต์แวร์ที่มีรูปแบบเป็น Platform เช่น E-Commerce Platform, Streaming Service Platform และ Social Network Platform

หนังสือ Practical DevOps and Cloud Engineering เล่มนี้ เป็นคู่มือพัฒนาทักษะทางด้าน DevOps and Cloud Engineering ภาคปฏิบัติ มีจุดประสงค์เพื่อพัฒนาทักษะ 2 ด้านคือ "Reskill" (การเสริมทักษะใหม่เพื่อรองรับอนาคต) และ "Upskill" (การพัฒนาทักษะที่มีอยู่ให้ดีขึ้นตามที่บริษัทต้องการ) โดยเน้นการพัฒนาซอฟต์แวร์ผ่าน Practice & Workshop ภายใต้แนวคิดใหม่อย่าง "Microservices Architecture" (Design-Develop-Deploy) เพื่อให้ผู้อ่านมีแนวทางที่จะนำความรู้และทักษะที่ได้จากหนังสือเล่มนี้ไปประยุกต์ใช้ได้ในการทำงานจริง

เพราะฉะนั้น หนังสือเล่มนี้จึงเหมาะสำหรับนักศึกษา โปรแกรมเมอร์ นักพัฒนาซอฟต์แวร์ และผู้ที่อยากข้ามสายงานมาทำงานทางด้าน "DevOps Engineer/Cloud DevOps Engineer" สายงานที่มีบทบาทสำคัญต่อการพัฒนาผลิตภัณฑ์ที่เป็นซอฟต์แวร์ และเป็นที่ต้องการมากที่สุดอาชีพหนึ่งในปัจจุบัน

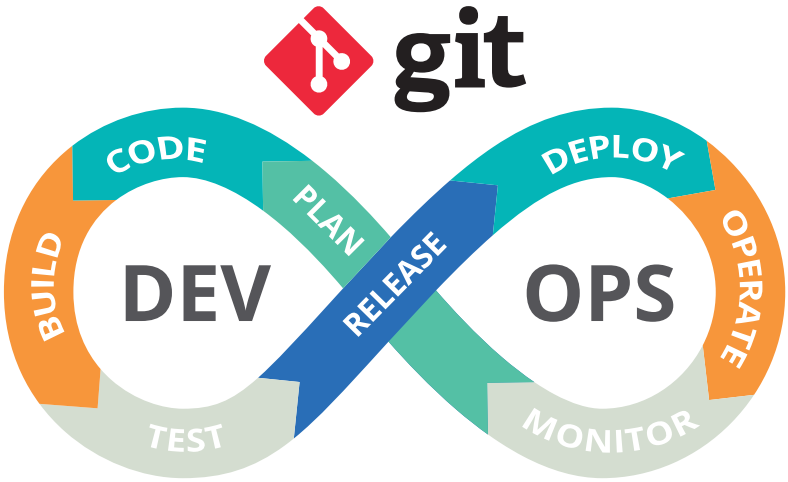
หวังเป็นอย่างยิ่งว่า หนังสือเล่มนี้จะทำหน้าที่เป็นสะพานเชื่อม หรือลดช่องว่างระหว่างภาคการศึกษา ซึ่งเป็นผู้ผลิตบุคลากรป้อนตลาดแรงงาน กับภาคธุรกิจอุตสาหกรรมซอฟต์แวร์ที่ต้องการบุคลากรที่มีทักษะเหมาะกับงาน

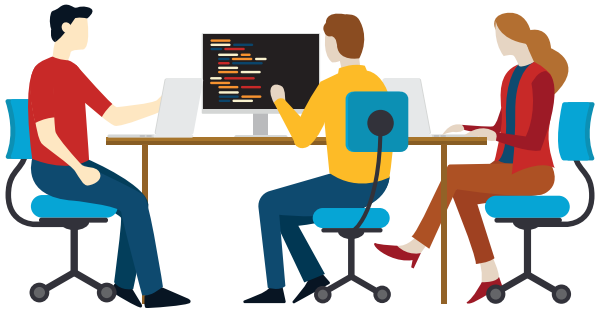
ภีรพล คชาเจริญ
บรรณาธิการ

CONTENTS

CONTENTS

CONTENTS


git
CODE
PLAN
BUILD
DEV
RELEASE
DEPLOY
OPERATE
OPS
TEST
MONITOR



CHAPTER

01

แนวคิดการจัดเก็บเวอร์ชัน

(Version Control Concepts)

ในการพัฒนาซอฟต์แวร์ให้แล้วเสร็จ จนสามารถนำมาใช้งานได้อย่างสมบูรณ์นั้น จำเป็นจะต้องผ่านกระบวนการมากมายหลายขั้นตอน สิ่งที่พบได้ในระหว่างพัฒนาคือ การแก้ไขหรือปรับปรุงซอร์สโค้ดของทีมพัฒนาซอฟต์แวร์ ฉะนั้น การจัดเก็บประวัติเวอร์ชันของซอร์สโค้ดจึงเป็นเรื่องสำคัญ ทั้งนี้ เพื่อให้ทีมพัฒนาฯ สามารถติดตามความเปลี่ยนแปลงที่เกิดขึ้นได้ตลอดการทำงาน นับตั้งแต่เริ่มต้นไปจนถึงได้ซอฟต์แวร์ที่สมบูรณ์คือใช้งานได้จริง มีข้อบกพร่องน้อยที่สุด

การจัดเก็บเวอร์ชันในการพัฒนาซอฟต์แวร์

หากพูดถึง การจัดเก็บเวอร์ชัน (Version Control) โดยเฉพาะ Git หลายคนคงเคยได้ยินกันมาบ้าง และถ้าใครไม่อยากเชยก็น่าจะเรียนรู้ไว้ เพราะ Version Control ถือว่าเป็นสิ่งจำเป็นสำหรับการพัฒนาซอฟต์แวร์ (Software Development) เปรียบได้กับปัจจัย 4 ที่จำเป็นต่อการดำรงชีวิตของมนุษย์เลยทีเดียว

Version Control หรือ Source Control หมายถึง เครื่องมือช่วยติดตามการเปลี่ยนแปลงของซอร์สโค้ด (Source Code) โดยการเก็บประวัติการเปลี่ยนแปลงลงฐานข้อมูลชนิดพิเศษ ซึ่งจุดประสงค์ของการเก็บบันทึกทุกการเปลี่ยนแปลงก็คือ หากมีความผิดพลาดเกิดขึ้น ไม่ว่าจะเป็นความผิดพลาดเพียงเล็กน้อย ไปจนถึงความผิดพลาดขั้นร้ายแรงที่อาจทำให้ซอฟต์แวร์ที่กำลังพัฒนาอยู่นั้นพังทั้งโปรเจ็กต์ ซึ่งเหตุการณ์ทำนองนี้เป็นเรื่องปกติ การบันทึกประวัติเก็บไว้อย่างต่อเนื่อง จะช่วยให้นักพัฒนาสามารถย้อนกลับไปยังช่วงเวลาต่างๆ เพื่อเปรียบเทียบโค้ดใหม่กับโค้ดเวอร์ชันก่อนหน้า และตรวจสอบความผิดพลาดเพื่อแก้ไขให้มีผลกระทบต่อทีมน้อยที่สุด

นอกจากนี้ Version Control ยังเป็นเครื่องมือหลักของทีมพัฒนาแบบ DevOps ที่ทำให้เกิดความต่อเนื่องในการพัฒนาซอฟต์แวร์ ช่วยให้นักพัฒนาสามารถ Deploy Software วันละหลายครั้งได้จริงในทางปฏิบัติ เพียงการ Push ซอร์สโค้ดกลับไปยัง Git เซิร์ฟเวอร์

ถ้าจะเทียบให้เห็นภาพจากเรื่องใกล้ตัว Version Control ก็คงคล้ายกับคำสั่ง Undo/Redo ในโปรแกรม Microsoft Word ที่ช่วยเรียกคืนเวอร์ชันของเอกสารได้ แต่หากผู้ใช้สั่งบันทึกแล้วปิดโปรแกรม Microsoft Word เมื่อเรียกไฟล์ขึ้นมาแก้ไขอีกครั้ง ผู้ใช้จะไม่สามารถย้อนกลับไปยังเวอร์ชันก่อนหน้าได้อีก

แนวคิดการจัดเก็บเวอร์ชัน

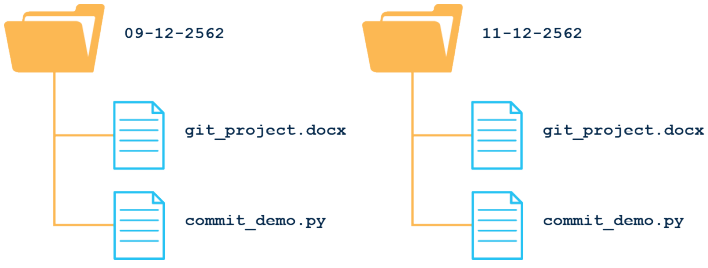
ในบทนี้ผู้อ่านจะได้เข้าใจแนวคิดและความเป็นมาของ Version Control ในยุคแรกๆ จนมาถึง Git ซึ่งเป็น Version Control ยอดนิยมในยุคนี้

การจัดเก็บเวอร์ชันด้วยวิธีก็อปปี้ (Copy File & Folder)

การก็อปปี้ไฟล์และโฟลเดอร์คือ วิธีการพื้นฐานแรกเริ่มที่เราสามารถจัดการได้ เป็นลักษณะของการบันทึกเอกสารแยกออกเป็นหลายๆ ไฟล์ แล้วตั้งชื่อไฟล์เพื่อระบุเวอร์ชันตามลำดับ เช่น

```
git_project.docx
git_project_v2.docx
git_project_back.docx
git_project_final.docx
git_project_lastest.docx
```

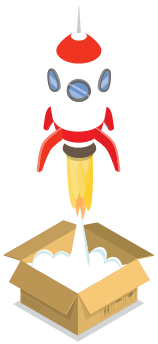
แต่จะเห็นว่า การตั้งชื่อไฟล์อาจทำให้ผู้ใช้สับสนในภายหลังว่า ไฟล์ไหนใหม่กว่ากัน รวมทั้งไม่สามารถบอกได้ว่า ไฟล์ไหนถูกสร้างต่อยอดมาจากไฟล์ไหนบ้าง เพราะฉะนั้น เพื่อไม่ให้เกิดความสับสนเรื่องเวอร์ชันของเอกสาร จึงมีการแก้ปัญหานี้โดยการจัดเก็บไฟล์แยกเป็นโฟลเดอร์ตามวันที่ เช่นตัวอย่างในรูปที่ 1.1



รูปที่ 1.1 ตัวอย่างโครงสร้างโฟลเดอร์ตามวันที่เพื่อช่วยจัดการเวอร์ชัน (Control Version)

แต่ข้อเสียของวิธีการนี้ก็คือ จะทำให้สิ้นเปลืองเนื้อที่การจัดเก็บข้อมูลตามจำนวนเวอร์ชันของเอกสารที่เกิดขึ้น แม้ว่าจะมีการเปลี่ยนแปลงในเอกสารเพียงไม่กี่บรรทัด ก็ต้องคัดลอกไฟล์ทั้งโฟลเดอร์เพื่อสร้างเอกสารเวอร์ชันใหม่ขึ้นมา และบอกไม่ได้ว่าเวอร์ชันนั้นๆ มีความสำคัญขนาดไหน


git



CHAPTER

02

หลักการพื้นฐานของ Git

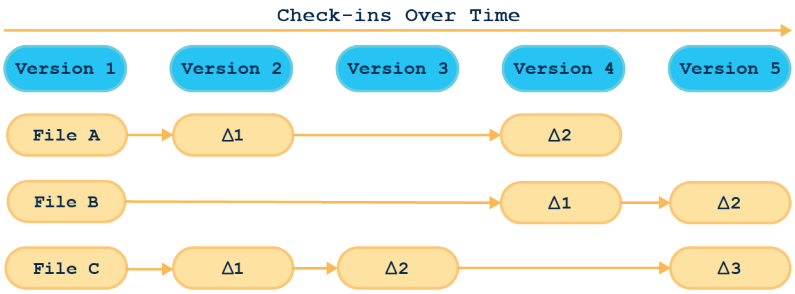
(Basic Principles of Git)

นอกจาก Linus Torvalds จะเปิดให้ใช้งาน Git แบบ Open Source และ Free Software แล้ว สิ่งที่ทำให้ Git เป็น Version Control ยอดนิยมในยุคนี้ก็คือ เรื่องของความเร็ว บทนี้ผู้อ่านจะได้เข้าใจวิธีการจัดเก็บ Version ของ Git ซึ่งเป็นปัจจัยหนึ่งที่ทำให้เราสามารถย้อนกลับไปยังเวอร์ชันก่อนหน้าได้รวดเร็ว เปรียบเหมือนกับการทำ Undo/Redo ใน Microsoft Word นอกจากนี้ เพื่อให้สามารถใช้งาน Git แบบ Command Line ได้อย่างคล่องแคล่ว ผมจะแนะนำขั้นตอนการทำงานของ Git (Git Workflow) ให้ผู้อ่านได้เข้าใจกันก่อน

เปรียบเทียบ Git กับ Version Control System อื่นๆ

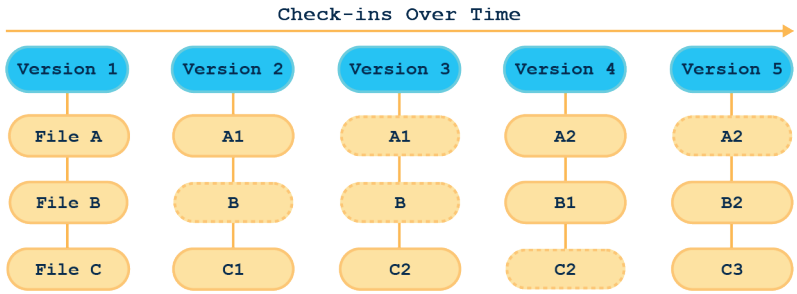
Git เป็น Distributed Version Control System (DVCS) ซึ่งต้องมีการก็อปปี้เวอร์ชันของซอร์สโค้ดมาเก็บไว้ที่เครื่องของเราก่อน (Local Host) ทำให้ผู้ใช้สามารถแก้ไขโปรเจ็กต์ได้ทุกที่แบบ Offline และ Check-in เพื่อจัดเก็บความเปลี่ยนแปลงของซอร์สโค้ดลงในฐานข้อมูลภายในเครื่องของเรา (Local Repository) โดยไม่จำเป็นต้องติดต่อกับ Git Repository บนเซิร์ฟเวอร์ และพอแก้ไขเสร็จแล้วจึงค่อยสั่ง Sync เพื่อให้ Version ฝั่ง Local และ Server อัพเดตเหมือนกันในภายหลัง (ด้วยการ Pull/Merge/Push)

ดังนั้น เมื่อเราสามารถทำทุกอย่างบน Local Host ได้ มันจึงเร็วกว่าระบบแบบรวมศูนย์ เช่น Subversion (SVN) ที่เป็น Centralized Version Control System อย่างมาก นอกจากนี้ Git ยังมีความแตกต่างกับ Version Control System (VCS) อื่นๆ ในเรื่องของวิธีการจัดเก็บข้อมูลเวอร์ชันโดย Version Control System ก่อนหน้านี้จะเก็บเป็นลิงค์ (Link) ของความเปลี่ยนแปลง หรือส่วนต่างระหว่างเวอร์ชัน ดังรูปที่ 2.1



รูปที่ 2.1 Delta-based Version Control
(ที่มา : https://git-scm.com)

แทนที่จะเก็บส่วนต่างระหว่างเวอร์ชัน แต่ **Git จะเก็บเวอร์ชันของซอร์สโค้ดเป็น Snapshot ซึ่งเป็นเหมือนการถ่ายรูปไฟล์** และอ้างถึงไฟล์ที่ไม่มีการเปลี่ยนแปลงในเวอร์ชันก่อนหน้า เมื่อมีการ Check-in มันจะถ่ายรูปไฟล์ของเราว่ามีหน้าตาเป็นอย่างไร โดย Git จะมองข้อมูลที่จัดเก็บเป็นเหมือนกับระบบแฟ้ม (File System) เล็กๆ ระบบหนึ่ง ดังรูปที่ 2.2



รูปที่ 2.2 Snapshot ซึ่งเป็น Version Control ในแบบของ Git
(ที่มา : https://git-scm.com)

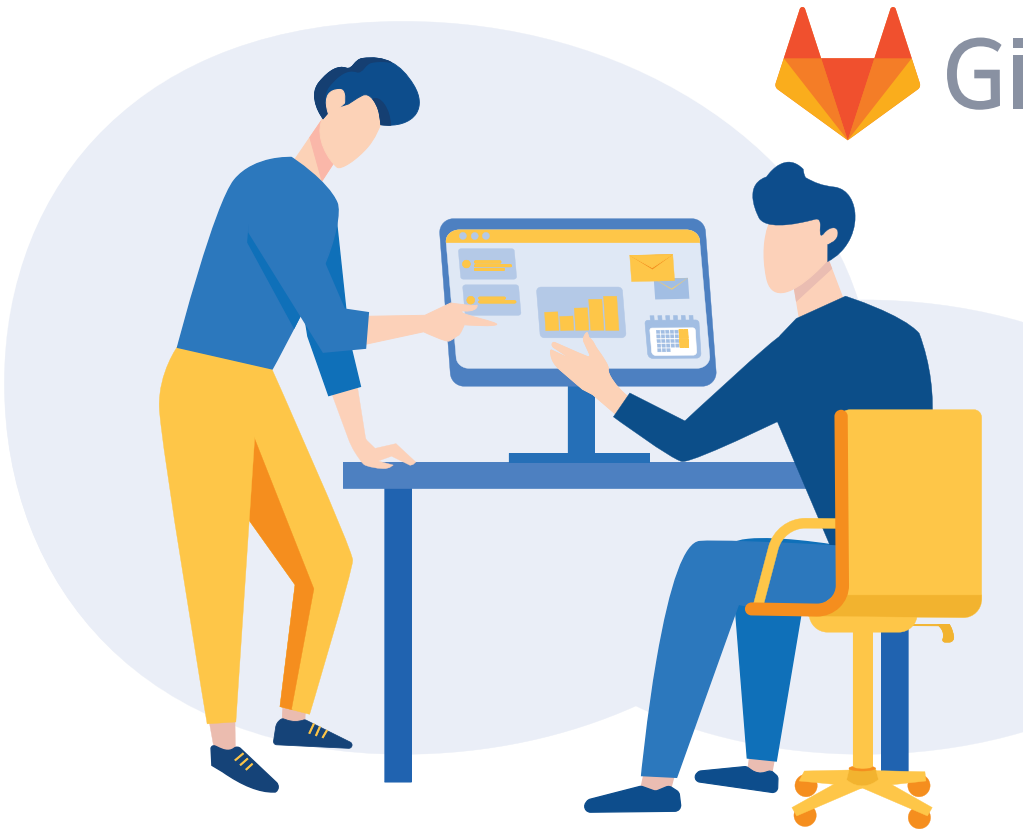
ก่อนที่จะมีการบันทึก ทุกอย่างในเวอร์ชันจะถูกนำไปหา Checksum ซึ่งเป็นการตรวจสอบความถูกต้องของข้อมูล โดยใช้ SHA-1 Hash ดังนั้น จึงไม่มีทางที่เราจะเปลี่ยนแปลงข้อมูลในไฟล์และโฟลเดอร์โดย Git จะไม่รู้ เราจะใช้ค่า Checksum ในการอ้างถึงเวอร์ชันของซอร์สโค้ดในหลายๆ งาน ซึ่ง SHA-1 Hash มีหน้าตาดังตัวอย่าง

ตัวอย่างของ Checksum

```
e70cdec636912065839fed45238db9d4f898f199
```

SHA-1 Hash คืออะไร

เป็นอัลกอริทึมที่อาศัยการคำนวณทางคณิตศาสตร์ในการเข้ารหัสทางเดียว โดยผลจากการเข้ารหัสเวอร์ชันแบบ SHA-1 จะได้เลขฐาน 16 ขนาด 40 ตัวอักษร


Gi

GitLab

CHAPTER

03

ฝึกการใช้งาน Git ขั้นพื้นฐาน

(Practicing Git Basics)

เพื่อให้ผู้อ่านได้เห็นขั้นตอนการทำงานของ Git (Git Workflow) จากการปฏิบัติจริง บทนี้เราจะเริ่มต้นด้วยการสร้าง Project บน GitLab Server แล้วเชื่อมโยงกับ Local Repository เนื่องจาก Git นั้นเป็น Distributed Version Control System เราจึงแก้ไข Project แบบ Offline แล้ว Check-in ซอร์สโค้ดบน Local Repository หลังจากนั้นจึง Sync ข้อมูลกับ GitLab Server อีกที


jupyter
GitLab

CHAPTER

04

การใช้งาน Git ร่วมกับ Jupyter Notebook

(How to use Git with Jupyter Notebook)

Jupyter Notebook เป็นเครื่องมือที่เหมาะสำหรับการทดสอบไอเดียด้วยการรันงานทางด้านวิทยาการข้อมูล (Data Science) และงานด้านปัญญาประดิษฐ์ (Artificial Intelligence : AI) อย่างมาก เนื่องจากผู้ใช้สามารถลองผิดลองถูกเขียน Code สลับไปมาระหว่าง Cell โดยไม่จำเป็นต้องเขียนโปรแกรมให้เสร็จสมบูรณ์ก่อน เราสามารถกดปุ่ม Ctrl + Enter เพื่อทดลองรัน Code ดูผลลัพธ์ทีละส่วน และไล่หา Error/Bug ไปเรื่อยๆ แล้วนำ Code ที่ใช้งานได้ไปใส่ใน Cell ที่เตรียมไว้ ด้วยความสะดวกดังกล่าว Jupyter Notebook จึงกลายเป็นเครื่องมือที่ได้รับความนิยมเป็นอันดับต้นๆ

การทำ Version Control กับ Jupyter Notebook

อย่างไรก็ตาม การทำ Version Control กับ Jupyter Notebook นั้นไม่ใช่เรื่องง่าย เพราะการที่มันเก็บซอร์สโค้ด และผลลัพธ์จากการรันเป็น JSON และ Binary Values Encoded อาจทำให้เกิดความสับสนในการเปรียบเทียบความเปลี่ยนแปลงของซอร์สโค้ด โดยเฉพาะเมื่อมีการแชร์ Code ให้คนอื่นๆ ภายในทีม เพื่อแก้ปัญหาดังกล่าว เราจึงต้องมีการปรับค่า Jupyter Notebook ให้จัดเก็บซอร์สโค้ดในรูปแบบที่เรียบง่ายขึ้น

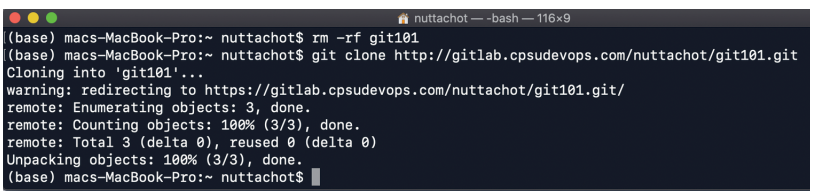
ทดลองแก้ไขและจัดเก็บซอร์สโค้ด

ในหัวข้อนี้ เราจะใช้คำสั่ง git diff เปรียบเทียบ Code ในสองเวอร์ชันที่เขียนโดยซอฟต์แวร์ Text Editor สำหรับการเขียนโค้ดยอดนิยม ได้แก่ Visual Studio Code และ Jupyter Notebook รวมทั้งอธิบายการแก้ปัญหาการทำงานร่วมกันของ Jupyter Notebook กับ Git โดยเราจะ Clone Project จาก URL ของ Remote Repository เดิมที่ได้สร้างไว้ในบทที่ 3 มาแก้ไข

การโคลนโปรเจ็กต์

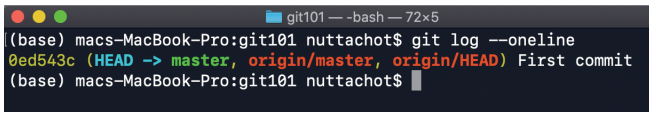
ก่อนจะ Clone ให้ลบโฟลเดอร์เดิม (git101) ออก แล้วใช้คำสั่ง git clone จะได้ผลลัพธ์ ดังรูปที่ 4.1

```
git clone http://gitlab.cpsudevops.com/nuttachot/git101.git
```



รูปที่ 4.1 ผลลัพธ์ของคำสั่ง git clone

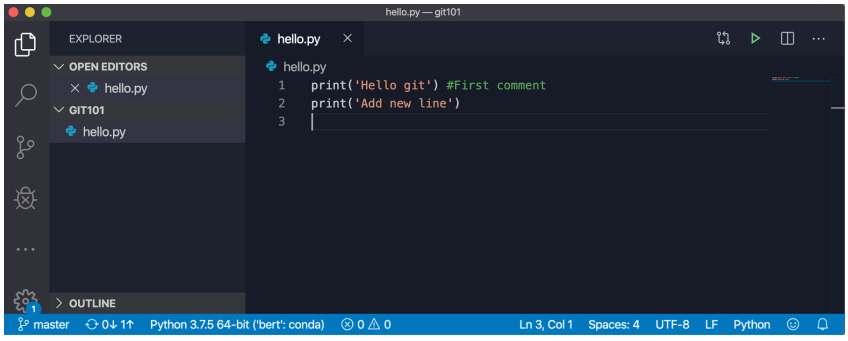
จากรูปที่ 4.1 Git จะก็อปปี้ซอร์สโค้ดทุกเวอร์ชันจาก Remote Project และสร้างโฟลเดอร์ใหม่ที่ชื่อ "git101" บนเครื่องของเรา ซึ่งเราสามารถใช้คำสั่ง git log --oneline เพื่อเรียกดู History ของ Branch ปัจจุบัน (Master Branch) ที่กำลัง Check-out ได้ผลลัพธ์ ดังรูปที่ 4.2



รูปที่ 4.2 ผลลัพธ์ของคำสั่ง git log --online

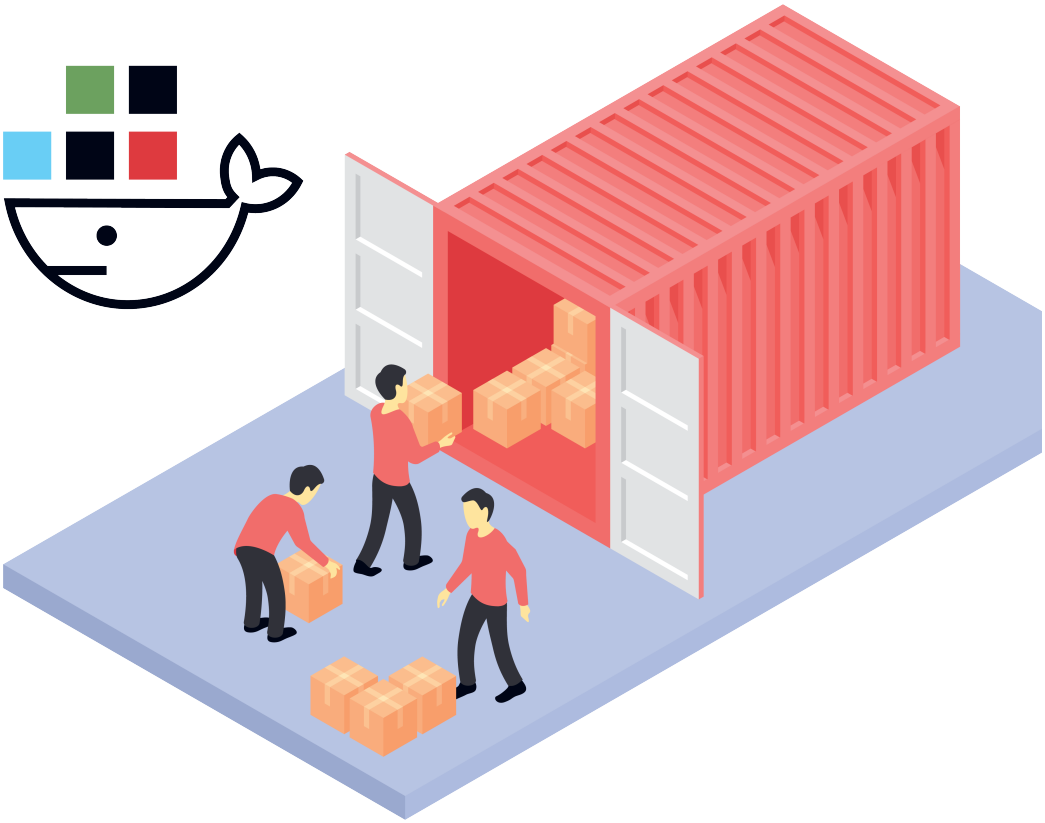
การเปิดและแก้ไขโค้ด

จากนั้นให้เปิดไฟล์ hello.py ที่อยู่ในโฟลเดอร์ git101 โดยใช้ Visual Studio Code แล้วเพิ่ม #First comment ที่บรรทัดแรก และคำสั่ง print('Add new line') ในบรรทัดที่ 2 แล้วคลิกปุ่ม Save เพื่อบันทึก ดังรูปที่ 4.3



รูปที่ 4.3 hello.py ที่เปิดเพื่อแก้ไข

ขณะนี้เรามีไฟล์ hello.py อยู่ 2 เวอร์ชันแล้ว โดยเวอร์ชันแรกถูก Check-in ด้วย Commit ID 0ed543c ส่วนอีกเวอร์ชันเป็น Code ที่มีการแก้ไขจาก Commit เดิม ซึ่งทำให้ไฟล์ hello.py มีสถานะเป็น Modified ดังรูปที่ 4.4 ที่มีการแสดงสถานะของ hello.py ด้วยคำสั่ง git status



CHAPTER

08

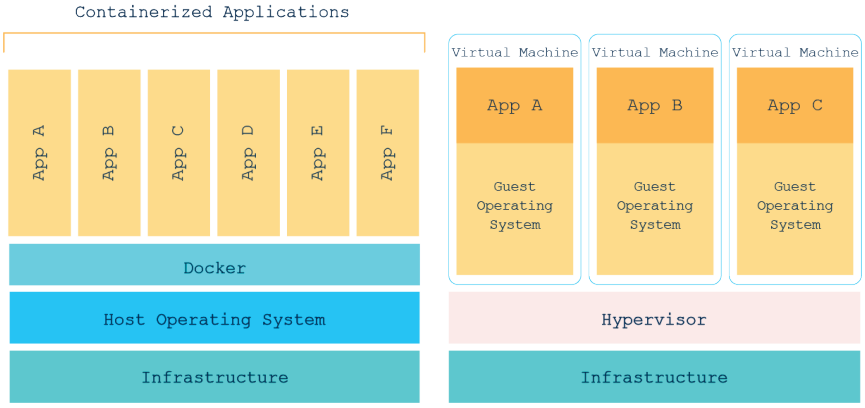
แนวคิดของ Docker Container

(Docker Container Concept)

หลายคนคงเคยได้ยินคำว่า Container ที่เป็นเครื่องมือสำหรับการบรรจุ จัดส่ง และ Deploy Software กันมาบ้าง ซึ่งถ้าคุณเป็น Developer ที่ทำงานอยู่ในบริษัทพัฒนาซอฟต์แวร์ในปัจจุบัน คุณมีโอกาสสูงที่จะได้เขียนโปรแกรมแล้วบันทึกเวอร์ชัน โดยใช้ Git และ Deploy บน Container ดังนั้น การเข้าใจแนวคิดของ Container จึงเป็นสิ่งจำเป็นขั้นพื้นฐานของ Software Developer รวมทั้ง DevOps Engineer ซึ่งมีหน้าที่สำคัญในการดูแล Container ให้ทำงานได้อย่างราบรื่น

เปรียบเทียบ Container vs Virtual Machine

Container และ Virtual Machine นั้นมีเป้าหมายเดียวกันคือ การแยกแอพพลิเคชัน (Application) และไลบรารี (Library) รวมทั้งไฟล์ต่างๆ ที่เกี่ยวข้องเก็บไว้ในกล่องที่สามารถนำไปรันบน Host ได้ทุกๆ ที่ แต่ทั้ง Container และ Virtual Machine มีสถาปัตยกรรมที่แตกต่างกัน ดังรูปที่ 8.1

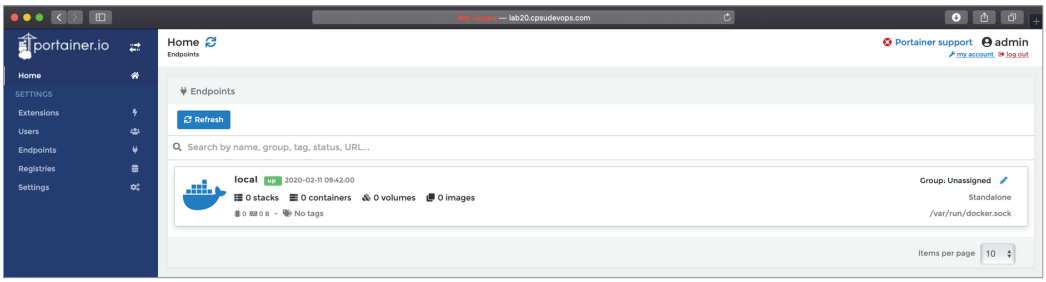


รูปที่ 8.1 Container และ Virtual Machine

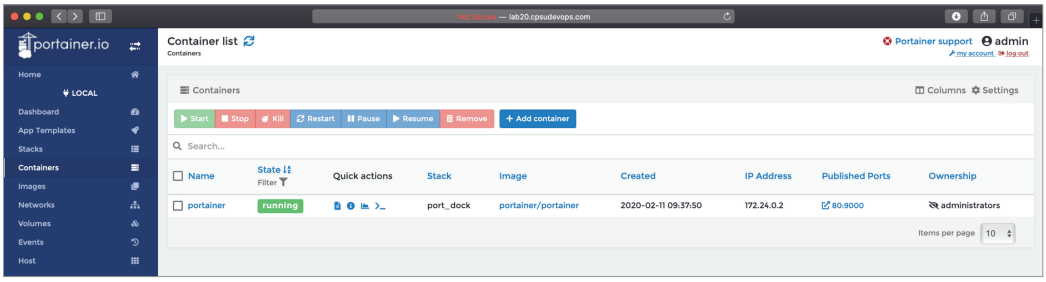
Virtual Machine นั้นจะจำลองคอมพิวเตอร์ทั้งเครื่อง และรันโปรแกรมผ่าน Guest OS ได้เหมือนกับคอมพิวเตอร์จริงๆ ดังรูปที่ 8.1 ขวามือ โดย Virtual Machine จะใช้ Hypervisor ในการรัน ซึ่ง Hypervisor สามารถทำงานบน Hardware/Infrastructure โดยตรง (Bare-metal) หรือจะทำงานบนระบบปฏิบัติการ (OS) ก็ได้

- **Hypervisor ที่รันบนระบบปฏิบัติการ (Hosted Hypervisor)** ได้แก่ VMware Workstation, Microsoft Virtual PC และ Oracle Virtual Box เป็นต้น
- **Hypervisor ที่รันบนฮาร์ดแวร์ (Bare-metal Hypervisor)** ได้แก่ VMware ESX, Citrix Xen Server และ Microsoft Hyper-V เป็นต้น

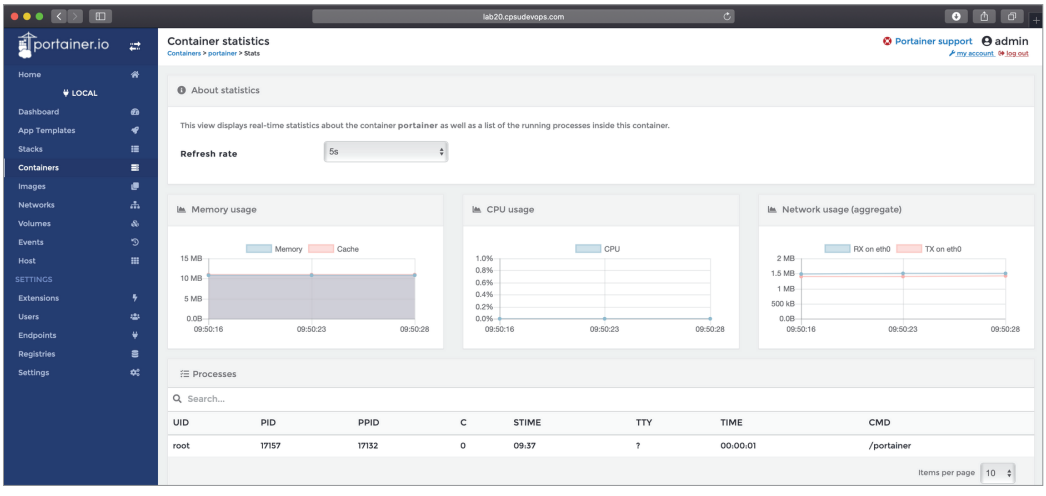
8. จะปรากฏหน้า Home ของ Portainer

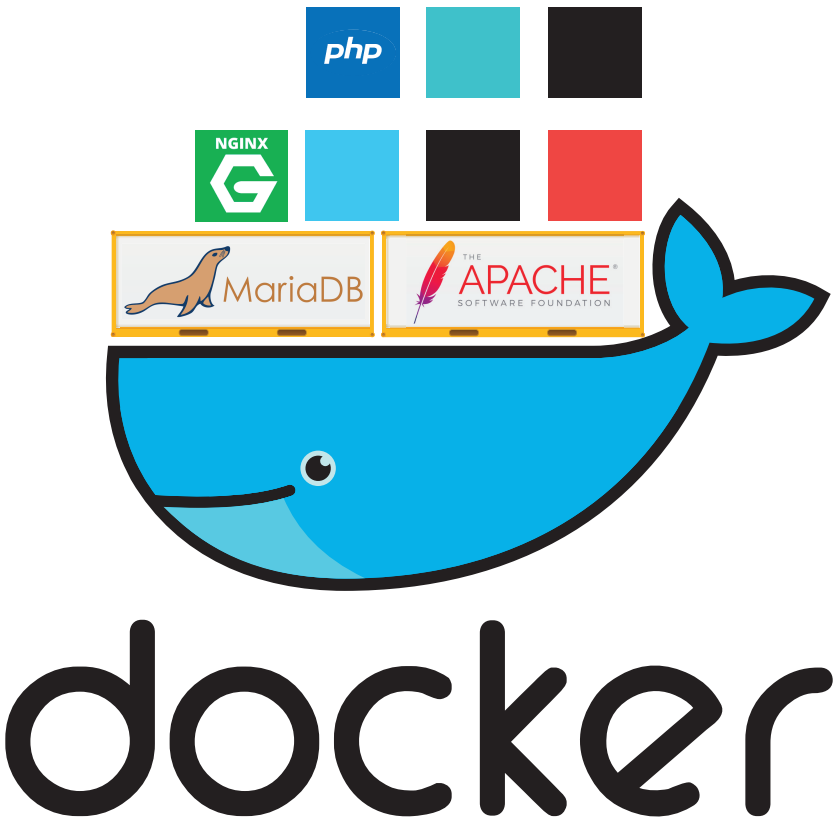


9. คลิกเมนู Containers เพื่อดู Container ทั้งหมด



10. คลิกปุ่ม Stats เพื่อดูสถานะการใช้ทรัพยากรของ Container




php
NGINX
MariaDB
THE APACHE SOFTWARE FOUNDATION
docker

CHAPTER

10

วิธีติดตั้ง LEMP Stack ด้วย Docker

(How to Setup LEMP Stack with Docker)

LEMP Stack เป็นกลุ่มของ Open Source Software สำหรับการสร้างเว็บไซต์ด้วยภาษา PHP ซึ่งในสมัยก่อน LEMP จะประกอบด้วย Linux OS, Apache, MySQL และ PHP แต่ปัจจุบัน Apache มักจะถูกแทนที่โดย Nginx และ MySQL จะถูกแทนที่โดย MariaDB คือ L = Linux OS, E = (E)Nginx Web Server, M = MariaDB และ P = PHP ในบทนี้เราจะติดตั้ง LEMP Stack บน Docker Container อย่างง่ายๆ ในเวลาไม่กี่นาที โดยใช้ Docker-compose

เปรียบเทียบ Apache vs Nginx

ปัจจุบันมี Web Server อยู่ 2 ค่ายที่ได้รับความนิยมอย่างแพร่หลาย คือ Apache ซึ่งครองตลาดมานาน โดย Apache Web Server มีการพัฒนามาตั้งแต่ปี ค.ศ. 1995 ขณะที่ Nginx ได้เปิดให้ใช้งานเมื่อปี ค.ศ. 2004 โดยนักพัฒนาชาวรัสเซีย ซึ่ง Web Server ทั้ง 2 ตัว สามารถทำงานได้บน Docker Container และใช้เวลาติดตั้งเพียงไม่ถึง 1 นาที

ขั้นตอนติดตั้ง Apache Web Server ด้วยคำสั่ง docker run

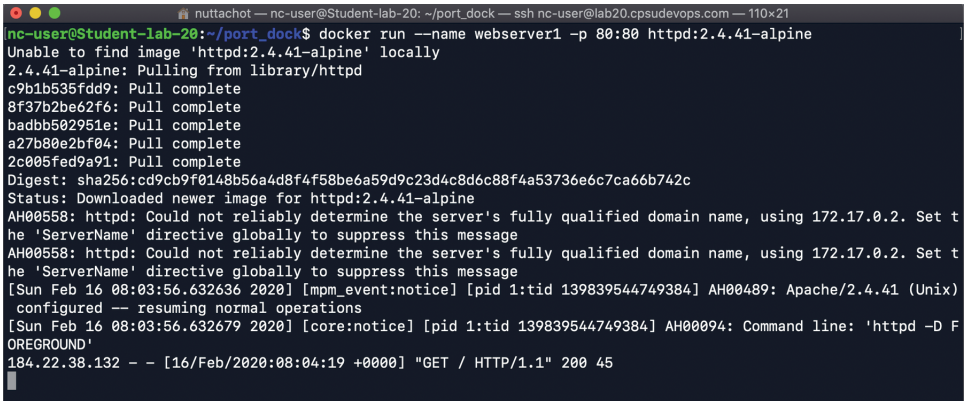
ทดลองติดตั้ง Apache Web Server บน Docker ตามขั้นตอน ดังต่อไปนี้

1. **ให้ Remote Login ไปยัง Cloud Server** โดยใช้คำสั่ง ssh แล้วรันคำสั่งด้านล่างหรือเปิด Ubuntu Command Line บน Windows หรือเปิด Command Line บน macOS

```
ssh nc-user@lab20.cpsudevops.com
```

2. **ติดตั้ง Apache Web Server โดยใช้ httpd:2.4.41-alpine Image ด้วยคำสั่ง** docker run

```
docker run --name webserver1 -p 80:80 httpd:2.4.41-alpine
```



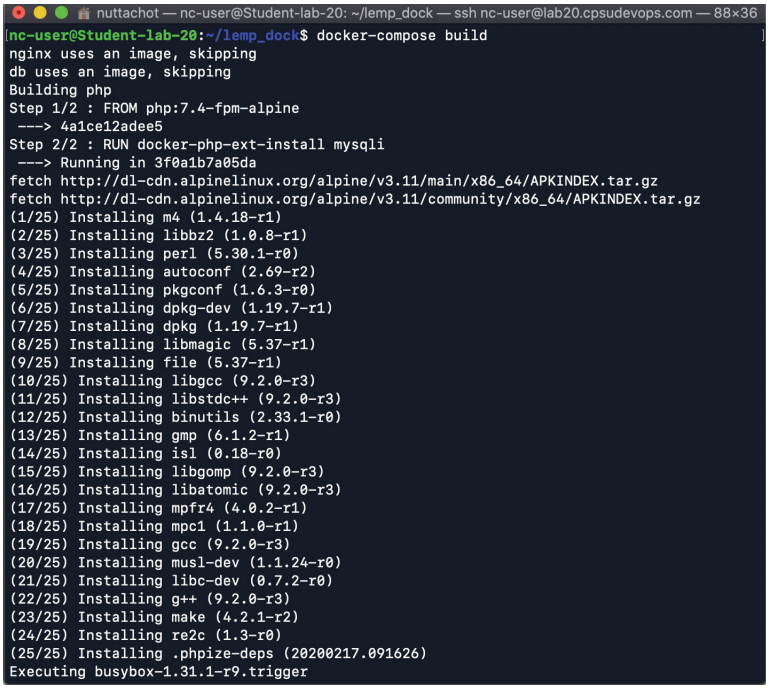
5. แก้ไขไฟล์ index.php ตามตัวอย่าง

index.php

```
<?php
   $servername = “db”;
   $username = “devops”;
   $password = “devops101”;

   $dbhandle = mysqli_connect($servername, $username, $password);
   $selected = mysqli_select_db($dbhandle, “titanic”);

   echo “Connected database server<br>”;
   echo “Selected database”;
?>
```

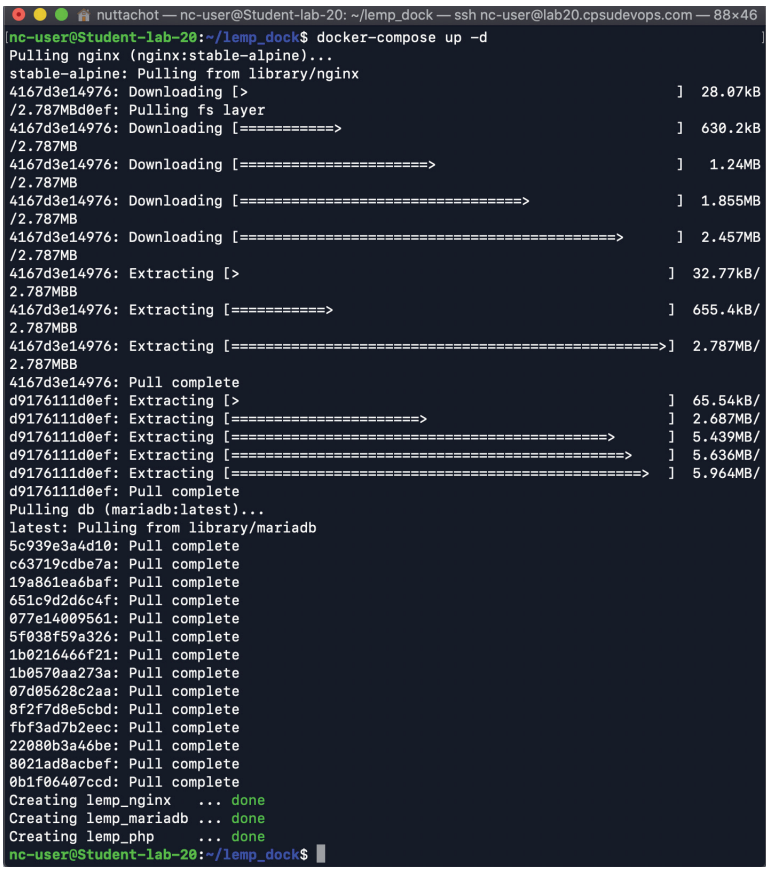
6. สร้าง php Image ด้วยคำสั่ง docker-compose build จะได้ผลลัพธ์ ดังรูป

```
docker-compose build
```

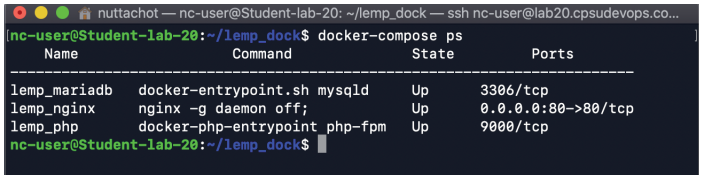


7. **รัน Container ด้วยคำสั่ง** docker-compose up และกลับไปที่ Command Line โดยใช้พารามิเตอร์ -d

```
docker-compose up -d
```



8. **เรียกดู Container ที่รันทั้งหมด** ที่ docker-compose.yml ดูแล



9. **เปิดดูเว็บไซต์ที่รันบน Container โดยระบุ URL เป็นชื่อ Server ของเราเอง** เช่น http://lab20.cpsudevops.com หรือ localhost

สรุปท้ายบท

- LEMP Stack สมัยใหม่จะประกอบด้วย Linux OS, (E)Nginx Web Server, MariaDB และ PHP
- เราสามารถ Remote ไปยัง Container ด้วยคำสั่ง docker exec -it
- Michael Widenius หนึ่งในผู้ก่อตั้งหลักของ MySQL ได้ "Fork" ซอร์สโค้ดของ MySQL เพื่อนำไปพัฒนาต่อในชื่อ "MariaDB"
- Mariadb จะ Import ไฟล์ "*.sql" ในโฟลเดอร์ docker-entrypoint-initdb.d ลง Database ให้อัตโนมัติถ้ามันพบว่าภายใน /var/lib/mysql/ ของ Container ยังไม่มีการสร้าง Database
- Nginx สามารถรองรับ Client Request ได้พร้อมกันจำนวนมาก ส่วนหนึ่งเป็นเพราะมันมีการตัดฟีเจอร์ อย่างเช่นการประมวลผลภาษา Script ออก แต่ในการนำมันมาใช้งานเป็น Web Server เรามักจะต้องมีการประมวลผลแบบ Dynamic Content โดยใช้ภาษา Script เช่น PHP ทำให้ต้องมีการส่งต่อการรันไฟล์ PHP ไปยังอีก Container

CHAPTER

11

วิธีติดตั้ง VPS และ Let's Encrypt ด้วย Docker Container

(How to setup VPS and Let's Encrypt with Docker)

หากใครเคยคอนฟิก Virtual Private Server (VPS) และติดตั้ง SSL Certificate เพื่อทำเว็บไซต์มาบ้าง คงทราบดีว่าการคอนฟิกโดยไม่ใช้ Docker ไม่ใช่เรื่องง่ายนัก แต่ในปัจจุบันการคอนฟิก VPS และติดตั้ง SSL Certificate (โดยเฉพาะ Let's Encrypt Certificate) บน Docker Container นั้นทำได้อย่างง่ายๆ โดยใช้เวลาไม่นาน ซึ่งหลักๆ เราจะใช้ฟีเจอร์ Reverse Proxy ของ Nginx Container เพื่อรับ Request จาก Client และ Forward ไปให้ Web Server Container ตามที่เรากำหนดไว้ โดยตั้งแต่บทนี้เป็นต้นไป ผู้อ่านจะได้ทำ Workshop โดยใช้ Linux Cloud Server ซึ่งสามารถนำไปต่อยอดเพื่อใช้งานได้จริง

เทคโนโลยี Cloud และ SSL Certificate

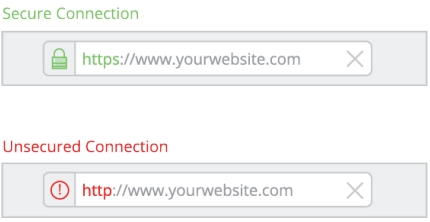
แนวคิดแบบ VPS และ Cloud Server

VPS (Virtual Private Server) คือ การจำลองเซิร์ฟเวอร์ขึ้นมาให้ทำงานได้เหมือนกับเซิร์ฟเวอร์จริง จึงถูกเรียกว่า "เซิร์ฟเวอร์เสมือน" แต่ปัญหาที่พบได้บ่อยของ VPS ก็คือ เซิร์ฟเวอร์ล่ม (Server Down) หรือมีปัญหาทำงานช้าเนื่องจากรับภาระโหลดการทำงานไม่ไหว

แต่พอมีเทคโนโลยี Cloud Computing ซึ่งเป็นระบบคอมพิวเตอร์ที่ใหญ่มากๆ โดยสามารถรองรับการใช้งานผ่านระบบเครือข่าย รวมทั้งสามารถประมวลผล ตลอดจนจัดเก็บข้อมูลได้อย่างมหาศาล การใช้งานระบบคอมพิวเตอร์จึงเปลี่ยนจากการที่องค์กรจะต้องลงทุนกับระบบเครือข่ายภายในสำนักงาน ไปใช้งานระบบคอมพิวเตอร์ของผู้ให้บริการ Cloud Computing หรือ Cloud Server แทน

ใบรับรองอิเล็กทรอนิกส์ SSL Certificate

เนื่องจากการใช้งานอินเทอร์เน็ตมีความเสี่ยงสูง โดยเฉพาะจาก Hacker ที่พยายามดักจับข้อมูล เราจึงต้องหาวิธีป้องกันการโจรกรรมข้อมูล ซึ่งวิธีหนึ่งที่นิยมใช้กันคือ การเข้ารหัสข้อมูลที่ถูกส่งผ่านโปรโตคอล HTTPS และ "SSL (Secure Socket Layer)"

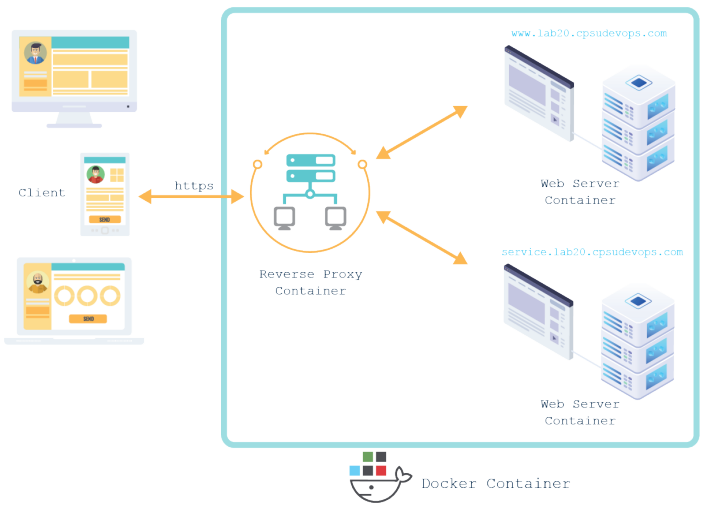


SSL Certificate คือ ใบรับรองอิเล็กทรอนิกส์ ที่ช่วยให้การสื่อสารข้อมูลในเครือข่ายอินเทอร์เน็ตมีความปลอดภัยมากขึ้น โดยมี Internet Security Research Group (ISRG) ทำหน้าที่เป็นหน่วยงานออกใบรับรองที่สามารถใช้งานได้ฟรี (Let's Encrypt) และใช้ระบบอัตโนมัติในการจัดการใบรับรอง

สรุปว่า Let's Encrypt คือ การทำเว็บให้เป็น HTTPS ซึ่งจะมีการเข้ารหัสข้อมูลบนเว็บไซต์ของเรา เพื่อเพิ่มความปลอดภัยในการรับ-ส่งข้อมูลบนเครือข่ายอินเทอร์เน็ต เราจะสังเกตเห็นรูปแม่กุญแจบนบราวเซอร์เมื่อเว็บไซต์นั้นมีการเข้ารหัสข้อมูล

สร้าง VPS และ Let's Encrypt ด้วย Docker Container

การคอนฟิก Virtual Private Server (VPS) และติดตั้ง SSL แบบ Let's Encrypt Certificate บน Docker Container นั้นทำได้ง่ายๆ ใช้เวลาไม่นาน ซึ่งหลักๆ เราจะใช้ฟีเจอร์ Reverse Proxy ของ Nginx Container เพื่อรับ Request จาก Client และ Forward ไปให้ Web Server Container ดังรูปที่ 11.1



รูปที่ 11.1 VPS และ Let's Encrypt บน Docker Container

3. **แก้ไข docker-compose.yml** โดยกำหนด VIRTUAL_HOST เป็นชื่อเซิร์ฟเวอร์ของตัวเอง ในที่นี้ผู้เขียนยกตัวอย่างเป็น www.lab20.cpsudevops.com และกำหนด Network ตามตัวอย่าง

docker-compose.yml

```
version: '3'
services:
  php:
    container_name: lemp_php
    build: php/
    restart: unless-stopped
    volumes:
      - ./html/:/var/www/html
    expose:
      - "9000"
    depends_on:
      - db

  nginx:
    container_name: lemp_nginx
    image: nginx:stable-alpine
    restart: unless-stopped

    networks:
      - webproxy
      - default

    environment:
      VIRTUAL_HOST: www.lab20.cpsudevops.com

    volumes:
      - ./html/:/var/www/html
      - ./nginx/conf/nginx.conf:/etc/nginx/conf/nginx.conf:ro
      - ./nginx/conf.d:/etc/nginx/conf.d:ro
    expose:
      - "80"

  db:
    container_name: lemp_mariadb
```

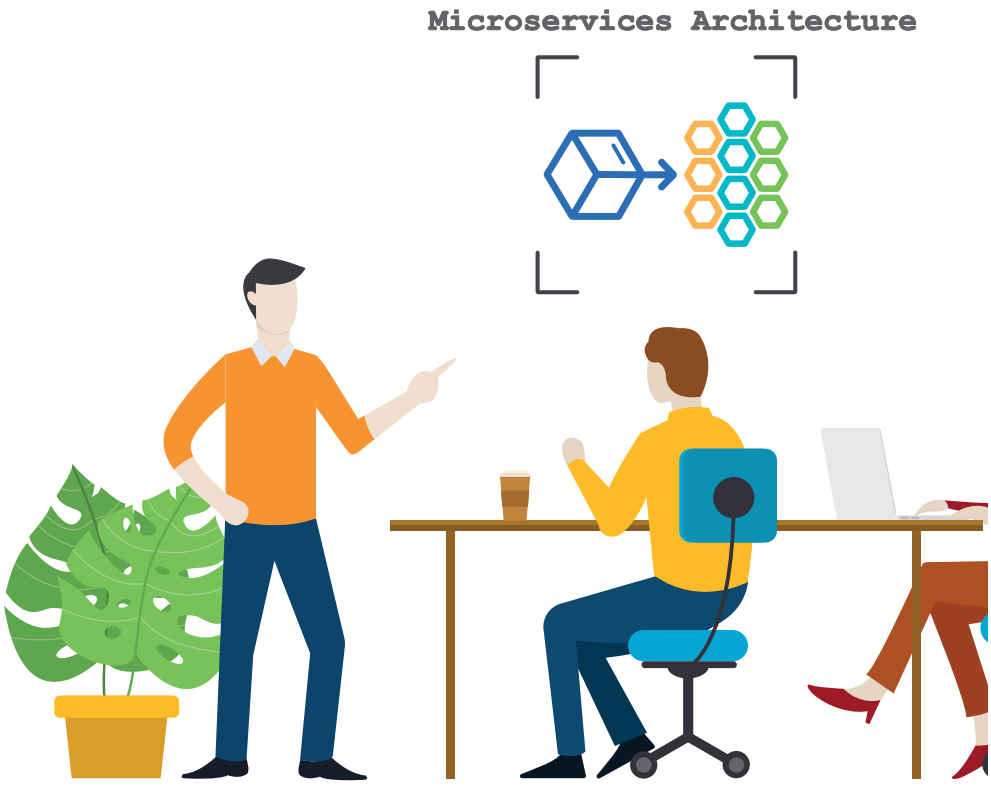
docker-compose.yml (ต่อ)

```
    image: mariadb:latest
    restart: unless-stopped
    volumes:
      - ./mariadb/initdb/:/docker-entrypoint-initdb.d
      - ./mariadb/data/:/var/lib/mysql/

    environment:
      - MYSQL_ROOT_PASSWORD=devops101
      - MYSQL_DATABASE=devops_db
      - MYSQL_USER=devops
      - MYSQL_PASSWORD=devops101

networks:
  default:
    external:
      name:
        web_network
  webproxy:
    external:
      name: webproxy
```

4. แก้ไขไฟล์ index.php ตามตัวอย่าง

index.php

```
<?php
   $servername = “db”;
   $username = “devops”;
   $password = “devops101”;
   $dbhandle = mysqli_connect($servername,$username,$password);
   $selected = mysqli_select_db($dbhandle, “titanic”);
   echo “Hello : from www.lab20.cpsudevops.com<br>”;
   echo “Connected database server<br>”;
   echo “Selected database”;
?>
```


Microservices Architecture

CHAPTER

12

การพัฒนา Microservices ด้วย Docker Container

(Microservices Architecture Development with Docker Container)

บทนี้เราจะทำความเข้าใจแนวคิดของ Microservices ที่ทำให้นักพัฒนาสามารถปล่อยซอฟต์แวร์ (Release Software) ได้ดีขึ้น และสามารถปรับขนาดเพื่อรองรับ Load ง่ายขึ้น ซึ่งในปัจจุบัน บริษัทชั้นนำหลายแห่ง เช่น Amazon, Netflix และ Wongnai ฯลฯ ต่างก็นำสถาปัตยกรรมแบบนี้ไปใช้งานจริงเป็น Production

การปรับเปลี่ยนจาก Monolith สู่ Microservices

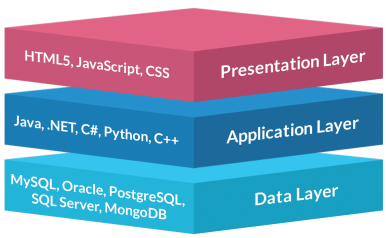
โครงสร้างของสถาปัตยกรรมแบบ "Microservices" จะมีผลต่อแนวทางการพัฒนาซอฟต์แวร์อย่างมาก ดังนั้น ตั้งแต่ Section 3 ของบทนี้ เราจะได้ทดลอง Workshop สร้าง Microservices จำลองระบบงานทะเบียนนักศึกษา เพื่อให้ผู้อ่านได้เข้าใจกระบวนการพัฒนาซอฟต์แวร์แบบ Microservices มากยิ่งขึ้น ซึ่งจะมีการ Deploy ระบบด้วย Docker Container และเรียกใช้ Service แบบ RESTful API และ RPC (Remote Procedure Call)

- สำหรับผู้อ่านใหม่ที่ต้องการจะ Implement ตาม สามารถอ่านเนื้อหาก่อนหน้านี้ จากบทที่ 11 วิธีติดตั้ง VPS และ Let's Encrypt ด้วย Docker Container
- ผู้อ่านสามารถ Stop/Delete Container และ Delete Image ด้วยคำสั่ง docker-compose down --rmi all เพื่อเคลียร์ Container และ Docker Image ที่ไม่ได้ใช้งานแล้วจาก Workshop ก่อนหน้า

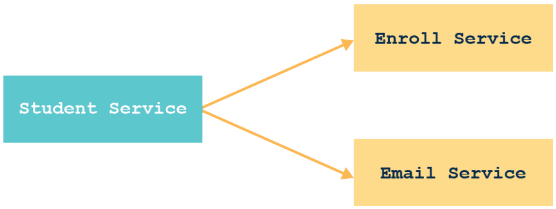
แต่ก่อนจะไปถึง Workshop ใน Section 1 และ 2 เราจะมาเปรียบเทียบสถาปัตยกรรมซอฟต์แวร์ (Software Architecture) ทั้งสองตัวคือ "Monolith vs Microservices" เพื่อให้เห็นความแตกต่างก่อนที่จะนำไปสู่การตัดสินใจปรับเปลี่ยนการทำงานจาก Monolith สู่ Microservices

Section 1 : Monolithic Architecture

หลายคนน่าจะรู้จักสถาปัตยกรรมแบบ Three-tier Application Architecture ซึ่งจะแบ่งระบบออกเป็น 3 ระดับ ได้แก่ Presentation Layer, Application Layer และ Data Layer ตามรายละเอียดดังต่อไปนี้

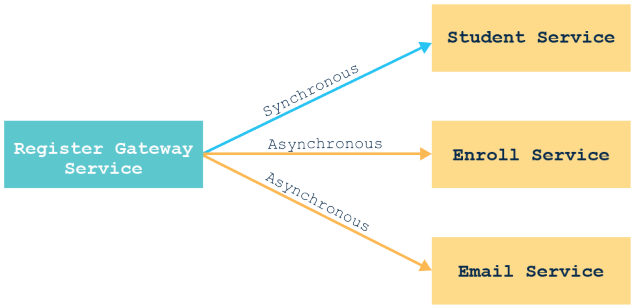


รูปที่ 12.1 ระดับชั้นของสถาปัตยกรรมแบบ Three-tier



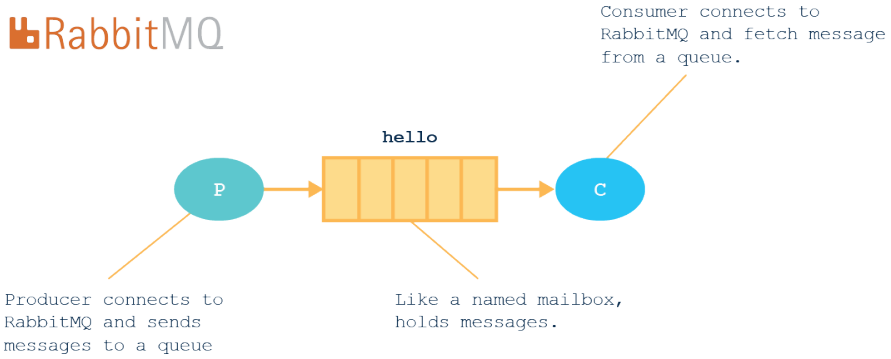
รูปที่ 12.6 จำลอง Service ที่เกี่ยวข้องกัน 3 Services

จากเดิมที่ Student Service จะต้องทำงานหลายหน้าที่ เราจะสร้าง Register Gateway Service เพื่อควบคุม Flow ของงาน โดยจะยอมให้เกิด High Coupling ที่ Register Gateway Service แทน นอกจากนี้ เราจะใช้ RPC ในการสื่อสารกับ Student Service แบบ Synchronous (ต้องรอ Student ID จาก Student Service) และใช้ RPC ในการสื่อสารกับ Enroll Service และ Email Service แบบ Asynchronous เพราะสามารถทำทั้ง 2 งาน สามารถทำขนานกันไปได้ ซึ่ง Register Gateway Service ไม่จำเป็นต้องรอให้ Service ทั้ง 2 ตัวทำงานเสร็จก่อนจึงจะ Reply ผลลัพธ์กลับไปยังผู้ใช้งาน



รูปที่ 12.7 การออกแบบการสื่อสารของทั้ง 3 Services

Section 3 : RabbitMQ



รูปที่ 12.8 แนวคิดของ RabbitMQ

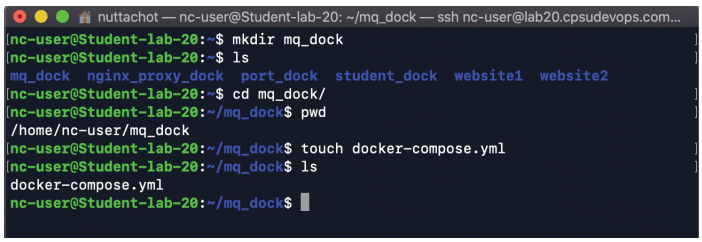
แต่ก่อนจะสร้าง Register Gateway Service เพื่อเป็นหน้าด่านรับ Request และประสานงานกับ Service ต่างๆ เราจะต้องติดตั้ง RabbitMQ เพื่อเป็นท่อส่งข้อมูลให้ Service ได้สื่อสารกันผ่าน RPC โดยจะมีการคอนฟิกตามขั้นตอน ดังนี้

1. Remote Login ไปยัง Cloud Server โดยใช้คำสั่ง ssh

```
ssh nc-user@lab20.cpsudevops.com
```

2. สร้าง Project ชื่อ "mq_dock" สำหรับจัดเก็บไฟล์ docker-compose.yml

```
.
|__docker-compose.yml
```



3. **แก้ไข docker-compose.yml** เพื่อติดตั้ง RabbitMQ Container ตามตัวอย่างด้านล่าง

docker-compose.yml

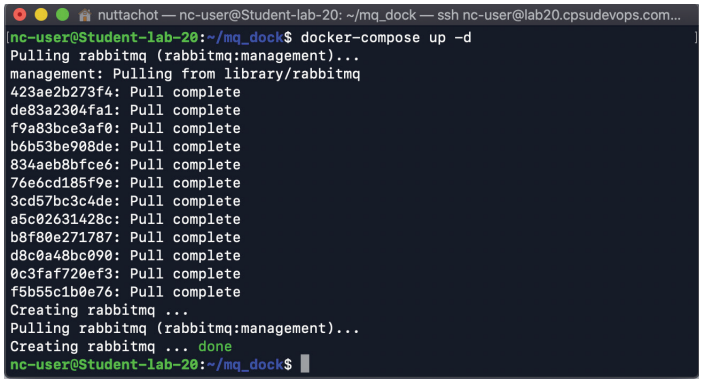
```
version: "2"
services:
  rabbitmq:
    image: "rabbitmq:management"
    container_name: rabbitmq
    restart: always

networks:
  default:
    external:
      name: microservice_network
```

4. **สร้าง Bridge Network ชื่อ "microservice_network"**

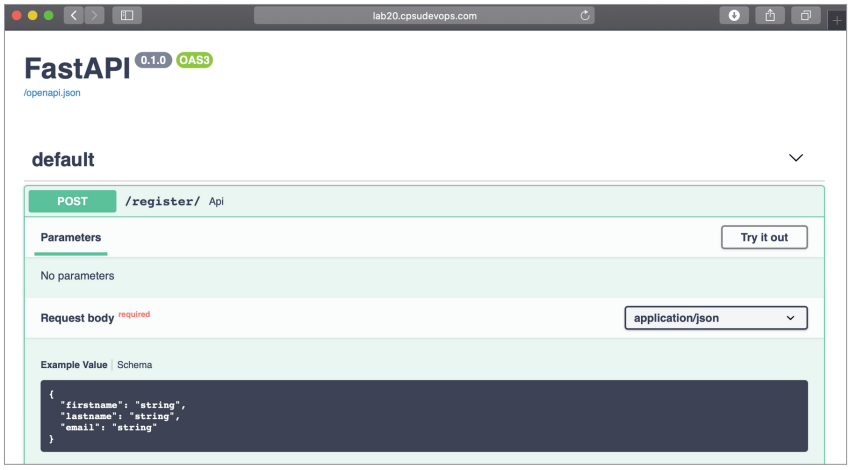
```
docker network create microservice_network
```

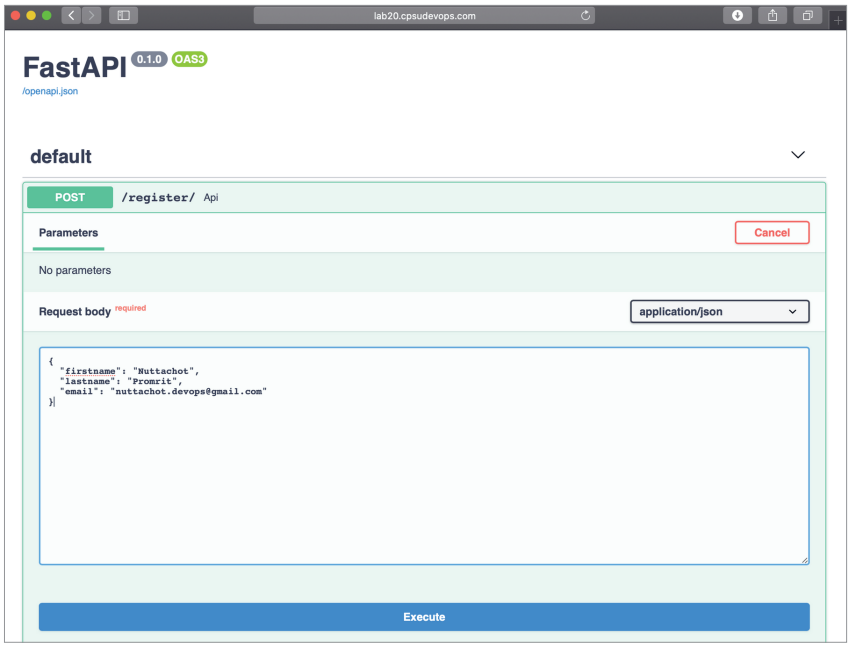
5. **รัน Container ด้วยคำสั่ง** docker-compose up และกลับไปที่ Command Line โดยใช้พารามิเตอร์ -d

```
docker-compose up -d
```



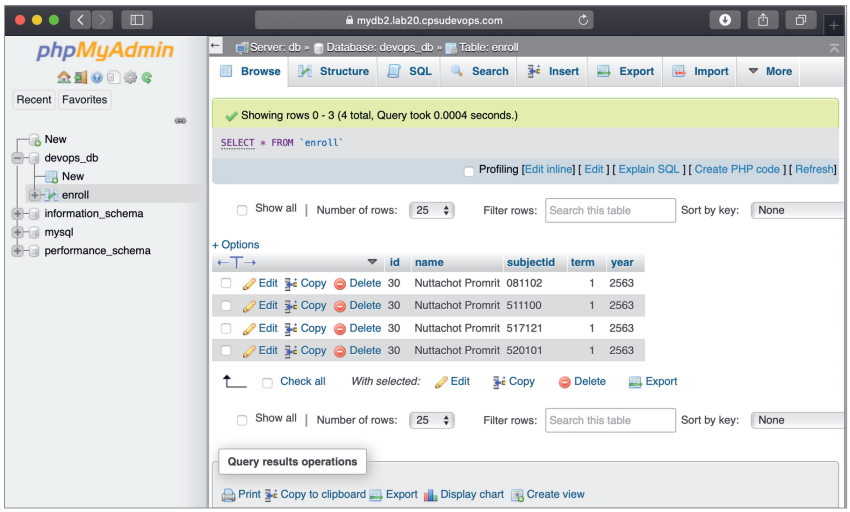
11. คลิกปุ่ม Try it out



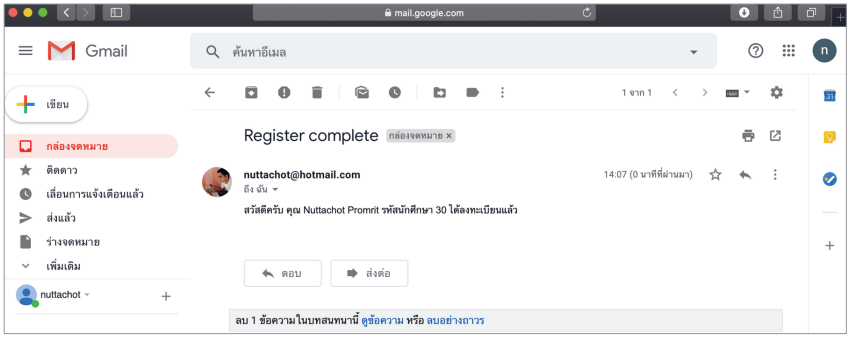


14. ดู Record ที่บันทึกโดย Enroll Service จากตัวอย่าง

URL: https://mydb2.lab20.cpsudevops.com



15. เปิดดู Email ใน Inbox Email จะพบข้อความยืนยันการลงทะเบียน



CHAPTER

13

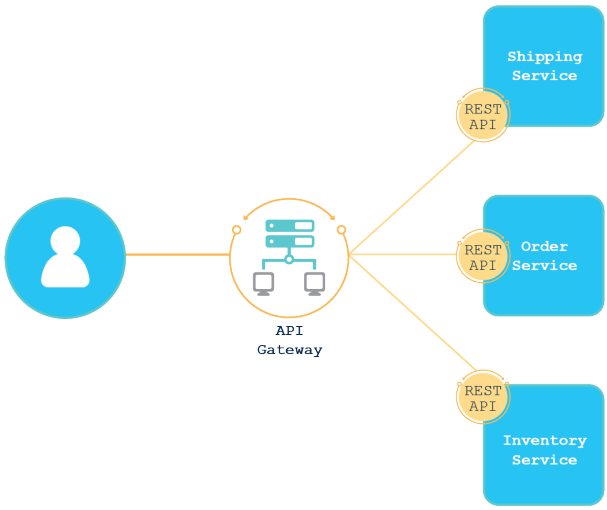
การติดตั้ง API Gateway และระบบ Monitoring ด้วย Kong + Prometheus + Grafana

(How to set up API Gateway and Monitoring System with Kong + Prometheus + Grafana)

ในบทนี้จะเป็น Workshop การติดตั้ง API Gateway และระบบ Monitoring งานทะเบียนนักศึกษาซึ่งถูกพัฒนาแบบ Microservices Architecture โดยใช้ Prometheus จัดเก็บข้อมูลต่างๆ ของ Cloud Server และ Service ในแบบ Time Series และใช้ Grafana ดึงข้อมูลมาแสดงผลบนหน้าจอแดชบอร์ด (Dashboard) แบบเรียลไทม์

หน้าที่ของ API Gateway

ในองค์กรที่ใช้แนวทาง DevOps นั้น Developer จะพัฒนาซอฟต์แวร์ให้อยู่ในรูปแบบที่สามารถส่งมอบ (Deploy) ได้อย่างรวดเร็วและถี่ขึ้น โดยมักมี API จะเป็นช่องทางการสื่อสารหลักระหว่าง Backend Services ต่างๆ รวมทั้งยังเป็นช่องทางในการเรียกใช้งานจาก Client อีกด้วย ดังนั้น ในระบบที่ใช้งานจริงเป็น Production จึงมักมีการติดตั้ง API Gateway และระบบ Monitoring เพื่อจะจัดการและดูแลการเข้าถึง API จากภายนอก



รูปที่ 13.1 API Gateway
(ที่มา : https://www.express-gateway.io/eg-vs-amazon-aws-api-gateway/)

บทนี้จะเป็น Workshop การติดตั้ง API Gateway และระบบ Monitoring งานทะเบียนนักศึกษา ซึ่งถูกพัฒนาภายใต้สถาปัตยกรรมแบบ Microservices ที่กำลังได้รับความนิยมในปัจจุบัน

สำหรับผู้อ่านใหม่ที่ต้องการจะ Implement ตาม ควรอ่านเนื้อหา 2 บทก่อนหน้านี้มาก่อน ได้แก่

- บทที่ 11 วิธีติดตั้ง VPS และ Let's Encrypt ด้วย Docker Container
- บทที่ 12 การพัฒนา Microservices ด้วย Docker Container

Kong API Gateway

API Gateway เป็นหนึ่งในเครื่องมือที่อยู่ระหว่าง Client และ Backend Services ซึ่งมันจะทำหน้าที่คล้ายกับ Reverse Proxy ในการคอยรับการเรียกใช้งานและส่งต่อ Request ไปยัง Service ที่กำหนดไว้ หรือกล่าวได้ว่า API Gateway นั้นเป็นจุดศูนย์รวมของการควบคุมการเข้าถึง Backend Services ที่เราเปิด Public ให้ Client ภายนอกเรียกใช้งาน ซึ่งอาจมีการทำ Authentication เพื่อยืนยันตัวตน การทำ Rate Limiting เพื่อควบคุม Traffic และการทำ Monitoring Service ร่วมกับเครื่องมืออย่างเช่น Prometheus เป็นต้น

รูปที่ 13.2 โลโก้ของ Kong และ Prometheus

Kong เป็น API Gateway แบบ Open Source ที่ได้รับความนิยม โดยเฉพาะการนำมาใช้กับซอฟต์แวร์แบบ Microservices ซึ่งเราสามารถคอนฟิก Kong ได้ใน 4 ส่วนหลักๆ ได้แก่

- **Service** คือ Service ปลายทางที่ต้องการให้ Kong ส่งต่อ Request ไปหา
- **Route** คือ Path ที่ Client เรียกมาที่ Kong โดย Route จะต้องมีการผูกกับ Service
- **Consumer** คือ ผู้ที่จะเรียกใช้ Kong
- **Plugin** คือ ตัวที่ทำให้ Kong สามารถทำ Authentication, Rate Limiting หรือ Monitoring Service ฯลฯ

ซึ่งมีการกำหนด Default Port ในการรับ-ส่งข้อมูลไว้ ดังนี้

- **8000** สำหรับ **HTTP Request** ที่จะถูกส่งต่อไปยัง Service ตามที่กำหนด
- **8443** สำหรับ **HTTPS Request** ที่จะถูกส่งต่อไปยัง Service ตามที่กำหนด
- **8001** สำหรับการคอนฟิก Kong ผ่าน **HTTP Request**
- **8444** สำหรับการคอนฟิก Kong ผ่าน **HTTPS Request**


redis
Flask
web development,
one drop at a time

CHAPTER

14

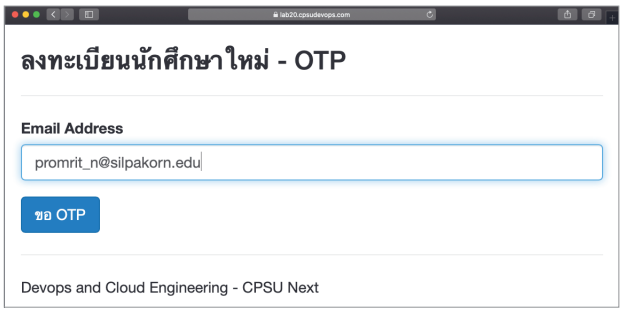
การพัฒนาระบบ OTP Service และ Session Server ด้วย Redis และ Flask

(OTP Service and Session Server Development with Redis and Flask)

ในบทนี้จะมีการสร้าง Microservices จำลองระบบงานทะเบียนนักศึกษาเพิ่มเติมอีก 4 Services ได้แก่ 1) OTP Service ซึ่งเป็น Service ที่เก็บรายการ Email และ OTP สำหรับผู้ผ่านการสอบสัมภาษณ์ ในรูปของ Key/Value บน Redis Database, 2) Send Mail OTP Service เป็น Service ที่ส่ง OTP ไปยัง Email ของผู้ผ่านการสอบสัมภาษณ์, 3) OTP Gateway Service จะเป็น Service ที่ควบคุม Flow ของงาน, 4) Register UI Service จะเป็นหน้า UI สำหรับกรอกข้อมูลการลงทะเบียน

OTP Service และ Session Server คืออะไร

OTP เป็นคำย่อของ "One-Time Password" ซึ่งก็คือ รหัสผ่านสำหรับยืนยันตัวตนชั่วคราว ที่ใช้งานได้เพียงครั้งเดียว โดยอาจจะมีการส่งมายังผู้ใช้งานผ่าน SMS หรือ Email ซึ่ง OTP จะมีอายุจำกัด เช่น ไม่เกิน 3 นาที หากเลยเวลาที่กำหนดแล้วจะไม่สามารถนำมาใช้งานได้อีก ฉะนั้น OTP จึงเป็นส่วนหนึ่งของระบบที่จะช่วยเพิ่มความปลอดภัยในการทำธุรกรรมทางอินเทอร์เน็ต



รูปที่ 14.1 หน้าจอการขอ OTP การลงทะเบียนนักศึกษาใหม่

Session นั้นจะคล้ายกับ Cookies ที่ถูกสร้างขึ้นเมื่อมีการเปิดเว็บบราวเซอร์ แต่ Session จะถูกเก็บอยู่ทางฝั่ง Server แต่ Cookies จะอยู่ทางฝั่ง Client ผู้ดูแลเว็บไซต์อาจใช้ข้อมูลใน Session ในการตรวจสอบ และติดตามผู้ใช้งานที่เข้ามาเยี่ยมชมเว็บไซต์ หรือใช้เพื่อจำกัดการเข้าถึงหน้าเว็บ หรืออาจใช้เก็บข้อมูลการใช้งานของ User ในเว็บขายออนไลน์ เป็นต้น

บทนี้ผู้อ่านจะได้คลายข้อสงสัยเกี่ยวกับการพัฒนาระบบ OTP ที่มีการส่งไปยังผู้ใช้ทาง Email รวมทั้งการทำให้ Session ไม่หลุดในตอนทำ Load Balance รวมทั้งเมื่อมีการเพิ่มจำนวน Service เพื่อรองรับโหลดที่มากขึ้นแบบไม่มี Downtime โดยการพัฒนาจะมีการจัดเก็บข้อมูล OTP และ Session ลงใน In-memory Database แบบ Open Source ที่ชื่อว่า "Redis"

Redis เป็นระบบเก็บข้อมูลแบบ Key/Value บนหน่วยความจำ หรืออาจมองว่ามันเป็น NoSQL Database ตัวหนึ่งที่ทำงานบน RAM ซึ่งทำให้มันทำงานได้เร็วมากๆ

นอกจากนี้ Redis ยังมีความสามารถในการ Set Expire ให้กับข้อมูลที่จัดเก็บ เราจึงนำมันมา Implement ระบบ OTP (One Time Password) สำหรับการยืนยันตัวตนในงานทะเบียนนักศึกษา โดยสมมติว่า เมื่อมีการประกาศรายชื่อผู้ผ่านการสอบสัมภาษณ์ เพื่อเข้าศึกษาในระดับปริญญาตรีแล้ว ว่าที่นักศึกษาจะต้องเข้าระบบกรอก Email ของตนเองเพื่อขอ OTP ซึ่งมีการกำหนดอายุ OTP ไว้ 3 นาที หากหมดอายุ ผู้ใช้จะต้องขอ OTP ใหม่

สำหรับผู้อ่านใหม่ที่ต้องการจะ Implement ตาม สามารถอ่านเนื้อหา 3 บทก่อนหน้านี้ ได้แก่

- บทที่ 11 วิธีติดตั้ง VPS และ Let's Encrypt ด้วย Docker Container
- บทที่ 12 การพัฒนา Microservices ด้วย Docker Container
- บทที่ 13 การติดตั้ง API Gateway และระบบ Monitoring ด้วย Kong + Prometheus + Grafana

Microservices Architecture และ Design Workshop

เพื่อให้เห็นภาพมากขึ้น เราจะสร้าง Microservices จำลองระบบงานทะเบียนนักศึกษาเพิ่มเติมอีก 4 Services ได้แก่

1) **OTP Service** จะเป็น Service ที่เก็บรายการ Email และ OTP สำหรับผู้ผ่านการสอบสัมภาษณ์ ในรูปของ Key/Value บน Redis Database โดยจะถูกเรียกใช้ผ่าน RPC
2) **Send Mail OTP Service** จะเป็น Service ที่ส่ง OTP ไปยัง Email ของผู้ผ่านการสอบสัมภาษณ์ โดยจะถูกเรียกใช้ผ่าน RPC เช่นเดียวกัน
3) **OTP Gateway Service** จะเป็น Service ที่ควบคุม Flow ของงาน ซึ่งจะถูกเรียกใช้ผ่าน RESTful API
4) **Register UI Service** จะเป็น "ส่วนติดต่อกับผู้ใช้ (User Interface : UI)" สำหรับกรอกข้อมูลการลงทะเบียน

ซึ่งเมื่อผู้ผ่านการสอบสัมภาษณ์เข้าระบบ และกรอก Email ของตนเองผ่าน Register UI Service จะมี Flow เกิดขึ้น ดังนี้

1. จะมีการสร้าง OTP ใหม่ อายุ 3 นาที ที่ OTP Service
2. ส่ง OTP ไปยัง Email ของผู้ผ่านการสอบสัมภาษณ์โดย Send Mail OTP Service
3. เมื่อผู้ผ่านการสอบสัมภาษณ์กรอก Email และ OTP ของตนเองได้ถูกต้อง จะมีการสร้างและจัดเก็บ Session ที่ Session Server
4. เมื่อผู้ผ่านการสอบสัมภาษณ์กรอก ชื่อ-นามสกุล ที่ Register UI จะมีการเรียกใช้ Register Gateway Service เพื่อขึ้นทะเบียนเป็นนักศึกษา

รูปที่ 14.2 แสดง Flow ของการลงทะเบียน

ขั้นตอนการสร้าง OTP Service

เราจะสร้าง OTP Service ที่เก็บรายการ Email ของผู้ผ่านการสอบสัมภาษณ์ รวมทั้ง OTP สำหรับการพิสูจน์ตัวตน ตามขั้นตอนดังนี้

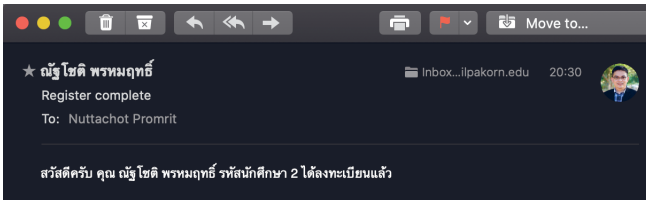
1. **Remote Login ไปยัง Cloud Server ของตัวเอง โดยใช้คำสั่ง** ssh

```
ssh nc-user@lab20.cpsudevops.com
```

2. **สร้าง Project ชื่อ "otp_dock"** ซึ่งภายในโฟลเดอร์จะประกอบด้วยไฟล์และโฟลเดอร์ ดังต่อไปนี้

```
.
|__ docker-compose.yml
|__ python/
   |__ Dockerfile
   |__ requirements.txt
   |__ rpc.py
```

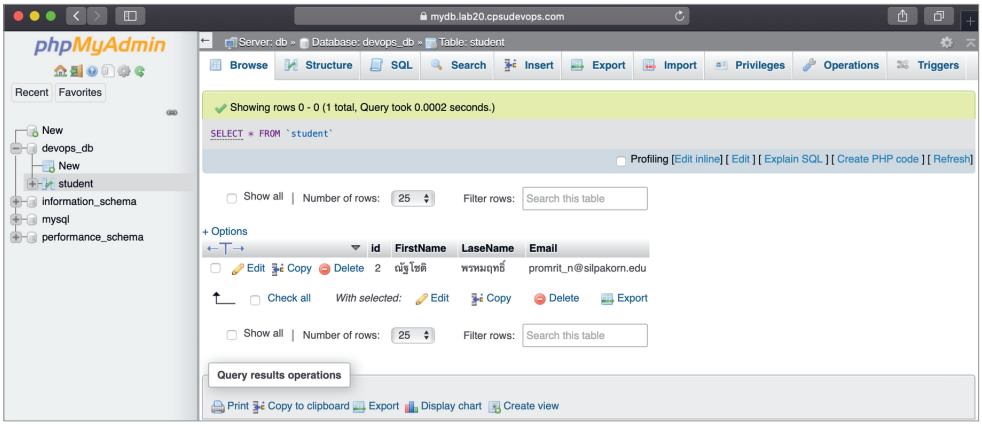
5. Email ยืนยันการลงทะเบียน



วิธีการตรวจสอบข้อมูลใน Database

ตรวจสอบข้อมูลนักศึกษาใหม่ใน Database จาก URL ดังตัวอย่างด้านล่าง

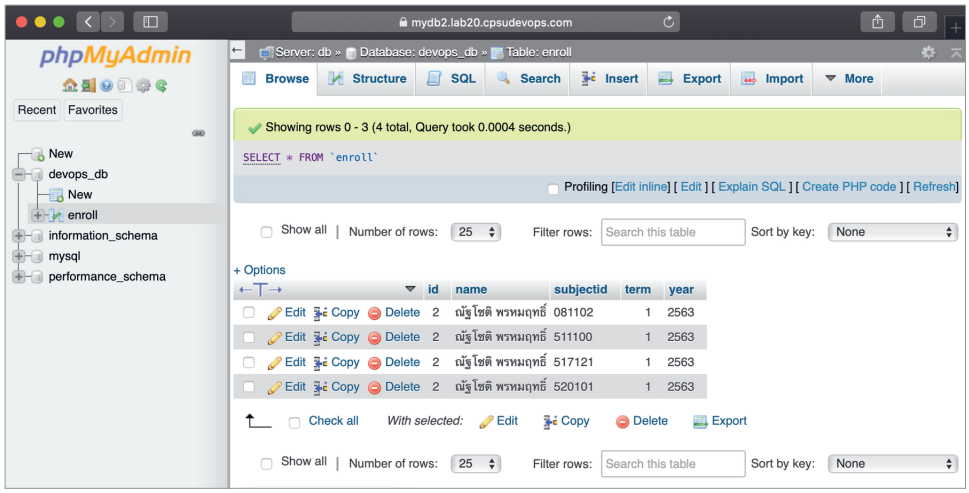
```
https://mydb.lab20.cpsudevops.com
```



รูปที่ 14.17 ข้อมูลนักศึกษาใหม่ใน Database

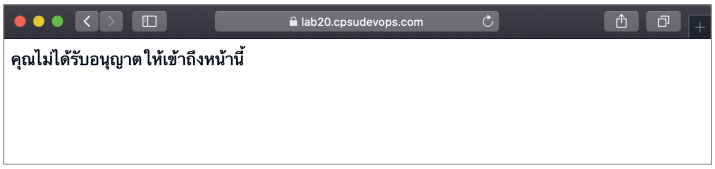
ตรวจสอบข้อมูลการลงทะเบียนของนักศึกษาใน Database จาก URL ดังตัวอย่างด้านล่าง

```
https://mydb2.lab20.cpsudevops.com
```



รูปที่ 14.18 ข้อมูลการลงทะเบียนของนักศึกษาใน Database

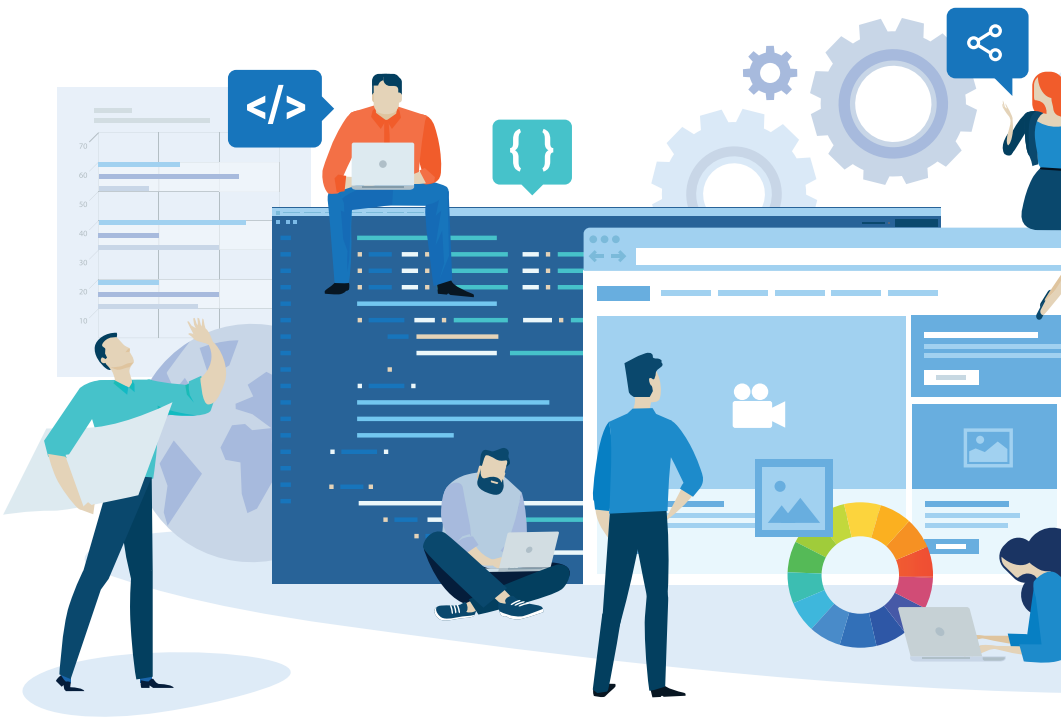
ทดลองเข้าหน้าลงทะเบียนอีกครั้ง จะพบข้อความ "คุณไม่ได้รับอนุญาตให้เข้าถึงหน้านี้"



รูปที่ 14.19 หน้าลงทะเบียนที่ไม่ได้รับอนุญาต

สรุปท้ายบท

- OTP หรือ One Time Password เป็นรหัสผ่านที่ใช้ได้ครั้งเดียว โดยระบบอาจจะมีการส่งมายังผู้ใช้งานผ่าน SMS หรือ Email เพื่อการพิสูจน์ตัวตน ซึ่ง OTP จะมีอายุจำกัด
- จากตัวอย่างบทนี้ มีการจัดเก็บข้อมูล OTP และ Session ลง Database แบบ In Memory ที่ชื่อว่า Redis
- Redis เป็นระบบเก็บข้อมูลแบบ Key/Value บนหน่วยความจำ หรืออาจมองว่ามันเป็น NoSQL Database ตัวหนึ่งที่ทำงานบน RAM ซึ่งทำให้มันทำงานได้เร็วมากๆ นอกจากนี้ Redis ยังมีความสามารถในการ Set Expire ให้กับข้อมูลที่จัดเก็บด้วย
- ในบทนี้มีการสร้าง Microservices จำลองระบบงานทะเบียนนักศึกษาเพิ่มเติมอีก 4 Services คือ
    1) OTP Service เป็น Service ที่เก็บรายการ Email และ OTP สำหรับผู้ผ่านการสอบสัมภาษณ์ในรูปของ Key/Value บน Redis Database
    2) Send Mail OTP Service เป็น Service ที่ส่ง OTP ไปยัง Email ของผู้ผ่านการสอบสัมภาษณ์
    3) OTP Gateway Service จะเป็น Service ที่ควบคุม Flow ของงาน
    4) Register UI Service จะเป็นหน้า UI สำหรับกรอกข้อมูลการลงทะเบียน



CHAPTER

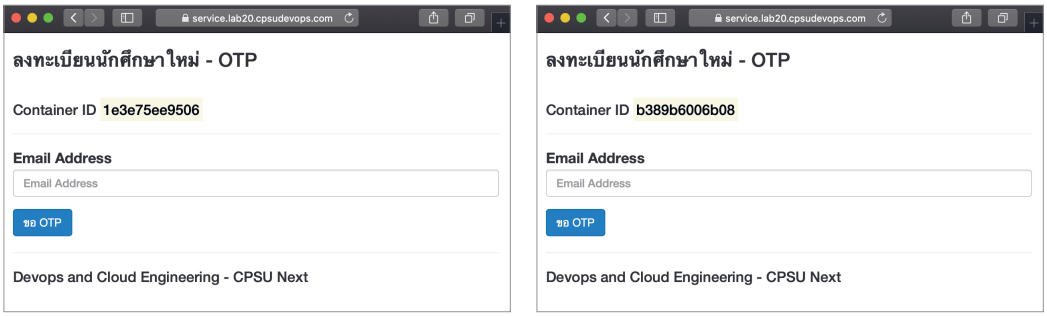
15

การพัฒนา Web Application แบบ (เกือบจะ) Zero Downtime ด้วย Swarm Cluster

(Zero Downtime Web Application Deployment with Docker Swarm)

ในบทนี้เราจะพัฒนา Web Application ซึ่งจะต้องสามารถ Online ได้ตลอดเวลา โดยใช้ความสามารถของ Docker Swarm ซึ่งมีชื่อเสียงในเรื่องของการนำเอา Cloud Server หลายๆ เครื่องมาช่วยกันทำงาน รวมทั้งการนำมาทำ Load Balance ที่ทำให้ผู้พัฒนาสามารถ Scale-out Microservices ได้ ซึ่งในที่สุดแล้วเราจะสามารถเพิ่มประสิทธิภาพให้ระบบสามารถรองรับ User ได้เป็นจำนวนเท่าไรนั้น โปรดติดตามกันได้

8. **เปิด Browser ไปที่ URL** เช่น https://service.lab20.cpsudevops.com/getotpui **แล้ว Refresh หน้าจอ** จะเห็นการสลับไปมาระหว่าง 2 Containers



ผลลัพธ์ที่ได้คือ จะเห็นการสลับไปมาระหว่าง Container ที่ 1 และ Container ที่ 2

Microservices Migration

เราจะนำ Microservices ที่เหลืออีก 5 Services รวมทั้ง RabbitMQ ไปรันบน Swarm Cluster (โดยจะยังไม่นำ Student และ Enroll RPC Service ที่มีการเรียกใช้งานฐานข้อมูลขึ้น Swarm Cluster) ซึ่งแต่ละส่วนจะมีการใช้ Network ดังนี้

Docker Swarm	
1	Register Gateway (register_gateway_swarm)
2	OTP Gateway (otp_gateway_swarm)
3	Email (email_swarm)
4	OTP (otp_swarm)
5	Send Email OTP (email_otp_swarm)
6	RabbitMQ (rabbitmq_swarm)
Docker	
7	Student (webproxy, student_network)
8	Enroll RPC (webproxy, enroll_network)

ขั้นตอน Migration : Register Gateway Service

1. สร้าง Network แบบ Overlay ชื่อ "register_gateway_swarm"

```
docker network create -d overlay register_gateway_swarm
```

2. ไปที่ Project "register_gateway_dock" แล้ว Stop/Delete Container และ Delete Image ด้วยคำสั่ง docker-compose down --rmi all

```
docker-compose down --rmi all
```

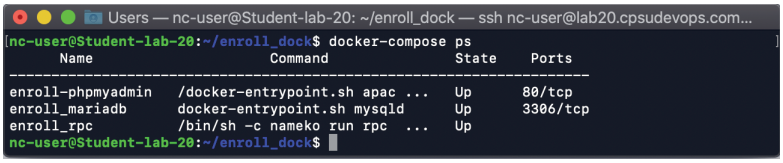
3. แก้ไขไฟล์ docker-compose.yml ตามตัวอย่าง

docker-compose.yml

```
version: '3'

services:
  register_gateway:
    build: python/
    image: localhost:5000/register_gateway
    ports:
      - "7001:80"
    deploy:
      restart_policy:
        condition: on-failure
        delay: 10s
      replicas: 1

networks:
  default:
    external:
      name:  register_gateway_swarm
```

5. เรียกดู Container ที่กำลังรันทั้งหมด ที่ docker-compose.yml ดูแล

```
docker-compose ps
```



ขั้นตอน Migration : Email Service

1. **สร้าง Network แบบ Overlay ชื่อ "email_swarm"** โดยใช้คำสั่งดังนี้

```
docker network create -d overlay email_swarm
```

2. **ไปที่ Project "email_dock" แล้ว Stop/Delete Container และ Delete Image ด้วยคำสั่ง** docker-compose down --rmi all

```
docker-compose down --rmi all
```

3. **แก้ไขไฟล์ docker-compose.yml** (เปลี่ยน RELAY_USERNAME และ RELAY_PASSWORD เป็นของตัวเอง)

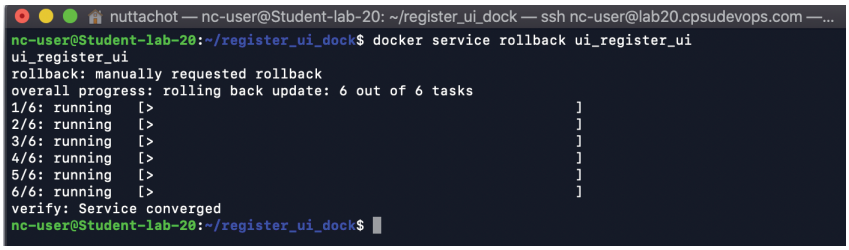
docker-compose.yml

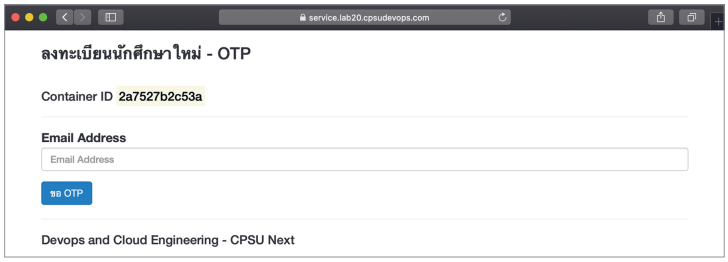
```
version: '3'

services:
  email_rpc:
    build: python/
    image: localhost:5000/email_rpc

    deploy:
      restart_policy:
        condition: on-failure
```

ขั้นตอนการ Rollback Service

การเปลี่ยนสีฟอนต์หน้า **ขอ OTP** กลับคืนมาเหมือนเดิมด้วยการ Rollback Service

```
docker service rollback ui_register_ui
```





รูปที่ 15.25 หน้าจอลงทะเบียนที่สีของข้อความ "ลงทะเบียนนักศึกษาใหม่ - OTP" กลับมาตามเดิม

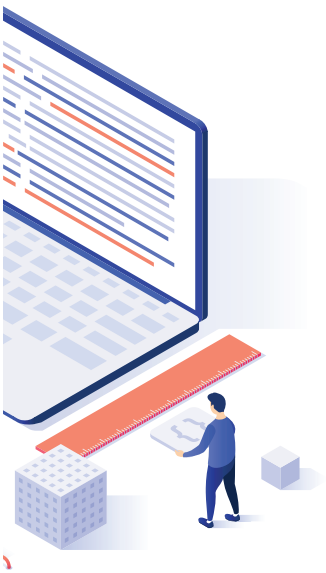
สรุปท้ายบท

- Apache JMeter เป็นเครื่องมือสำหรับการทำ Performance Test แบบ Open Source Software ที่พัฒนาด้วยภาษา Java
- ปัจจุบันมีซอฟต์แวร์ที่ใช้จัดการและควบคุม Container แบบ Cluster หรือที่เราเรียกว่า Container Orchestration หลักๆ อยู่ 2 ราย ได้แก่ Docker Swarm จาก Mirantis และ Kubernetes จาก Google
- ด้วยความที่ Docker Swarm นั้นใช้ง่าย และสามารถเรียนรู้ได้เร็วกว่า Kubernetes หลายเท่า จึงเป็นตัวเลือกที่ถูกนำมาศึกษาในบทนี้ แต่เมื่อผู้อ่านได้ฝึกปฏิบัติและเข้าใจแนวคิดต่างๆ ดีแล้ว การจะศึกษา Kubernetes จึงเป็นเรื่องที่ไม่ยากจนเกินไป

DevOps Engineering

CHAPTER

16



การทำ CI/CD Pipeline สำหรับ DevOps Team

(How to setup a CI/CD Pipeline for DevOps Team)

เพื่อให้สามารถส่งมอบซอฟต์แวร์ที่มีคุณภาพแก่ลูกค้าได้อย่างรวดเร็ว และมีการนำ Feedback กลับไปปรับปรุงงานให้ดียิ่งขึ้น เราจึงต้องมีเครื่องมือที่ช่วยให้สมาชิกในทีมนำงานของตัวเองมา Integrate, Delivery และ Deploy แบบอัตโนมัติ (CI/CD) ซึ่งเนื้อหาในบทนี้ ผู้อ่านจะได้ทดลองทำ CI/CD กับระบบลงทะเบียนนักศึกษาที่มีแนวทางการพัฒนาระบบตามแบบ Microservices Architecture โดยจะมีการ Build Image, ทำ Unit Test และ Deploy Software แบบอัตโนมัติบน Swarm Cluster โดยใช้ GitLab Server