**เรียน พฤ.9 ม.ค.68 0900 อ.คิม**

**Trello.com**

**Slack.com**

**GitHub**

**Docker**

**Trello.com** เป็นโปรแกรมหรือแพลตฟอร์มออนไลน์ที่ใช้สำหรับการจัดการงาน (Task Management) และโครงการ (Project Management) ในรูปแบบที่ใช้งานง่าย โดยมีลักษณะเด่นคือ **การใช้กระดาน (Boards)** และ **บัตรงาน (Cards)** เพื่อช่วยให้ผู้ใช้งานสามารถจัดระเบียบและติดตามงานต่าง ๆ ได้อย่างเป็นระบบ

**คุณสมบัติหลักของ Trello:**

1. **Boards (กระดาน):** ใช้สำหรับจัดระเบียบโปรเจกต์หรือหมวดหมู่งานต่าง ๆ
2. **Lists (รายการ):** ใช้แบ่งขั้นตอนการทำงาน เช่น "To Do", "Doing", "Done"
3. **Cards (บัตรงาน):** เป็นหน่วยงานย่อย ๆ ที่สามารถใส่รายละเอียด เช่น รายละเอียดงาน, กำหนดส่ง, เช็คลิสต์, หรือไฟล์แนบ
4. **การทำงานร่วมกัน:** สมาชิกในทีมสามารถทำงานร่วมกันในกระดานเดียวกันได้
5. **ปรับแต่งได้:** เพิ่มป้ายกำกับ, ตั้งกำหนดเวลา, เชื่อมต่อกับเครื่องมืออื่น ๆ เช่น Google Drive, Slack, หรืออื่น ๆ
6. **ใช้ได้หลายอุปกรณ์:** ทั้งบนเว็บเบราว์เซอร์, แอปมือถือ และเดสก์ท็อป

**ประโยชน์ของ Trello:**

* เหมาะสำหรับการวางแผนงานส่วนตัวหรือทีม
* ใช้งานง่าย แม้ไม่มีพื้นฐานด้านการจัดการโปรเจกต์
* ฟรี (สำหรับฟีเจอร์พื้นฐาน) และมีแผนแบบเสียเงินสำหรับฟีเจอร์ขั้นสูง

ตัวอย่างการใช้งาน:

* วางแผนโปรเจกต์ในที่ทำงาน
* วางแผนกิจกรรมส่วนตัว เช่น การจัดงานปาร์ตี้หรือการเดินทาง
* ติดตามงานที่ต้องทำในชีวิตประจำวัน

คุณสามารถเข้าไปที่ [Trello.com](https://trello.com) เพื่อสมัครใช้งานหรือทดลองใช้ได้เลย!

**Slack.com** เป็นแพลตฟอร์มการสื่อสารและการทำงานร่วมกันในองค์กรที่ออกแบบมาเพื่อช่วยให้ทีมสามารถพูดคุยและทำงานร่วมกันได้อย่างมีประสิทธิภาพ โดยแทนที่จะใช้การสื่อสารผ่านอีเมลทั้งหมด Slack ช่วยให้การสื่อสารรวดเร็วและเป็นระบบมากขึ้นผ่าน **ช่องทาง (Channels)** และ **การส่งข้อความ (Direct Messages)**

**คุณสมบัติหลักของ Slack:**

1. **Channels (ช่องทาง):**
   * เป็นพื้นที่สำหรับพูดคุยเกี่ยวกับหัวข้อเฉพาะ เช่น "การตลาด", "โปรเจกต์ A", หรือ "ฝ่ายทรัพยากรบุคคล"
   * สมาชิกในทีมสามารถเข้าร่วมหรือออกจากช่องทางได้ตามความเกี่ยวข้อง
2. **Direct Messages (ข้อความส่วนตัว):**
   * ใช้สำหรับการพูดคุยแบบตัวต่อตัวหรือในกลุ่มเล็ก ๆ
3. **การแชร์ไฟล์:**
   * สามารถแชร์ไฟล์, รูปภาพ, เอกสาร หรือวิดีโอได้โดยตรงในแชท
4. **การค้นหา (Search):**
   * สามารถค้นหาข้อความ, ไฟล์ หรือการพูดคุยเก่า ๆ ได้อย่างรวดเร็ว
5. **การผสานรวม (Integrations):**
   * รองรับการเชื่อมต่อกับเครื่องมืออื่น ๆ เช่น Google Drive, Trello, Zoom, Jira, และอื่น ๆ
6. **การแจ้งเตือน (Notifications):**
   * ปรับแต่งการแจ้งเตือนได้ตามความสำคัญ เช่น รับเฉพาะข้อความจากบางช่องทางหรือปิดเสียงบางการสนทนา

**ประโยชน์ของ Slack:**

* ลดความยุ่งยากของการใช้อีเมล
* เพิ่มความคล่องตัวในการทำงานร่วมกันในทีม
* สามารถพูดคุยกันแบบเรียลไทม์ หรือทิ้งข้อความไว้ให้สมาชิกในทีมที่อยู่คนละเวลาได้
* รองรับทั้งการใช้งานบนเว็บเบราว์เซอร์, แอปมือถือ และเดสก์ท็อป

**ตัวอย่างการใช้งาน:**

* การวางแผนและติดตามโปรเจกต์ในทีม
* การประชุมหรือพูดคุยเกี่ยวกับงานแบบเรียลไทม์
* การสื่อสารระหว่างทีมในองค์กรขนาดใหญ่หรือทีมที่ทำงานจากระยะไกล (Remote Work)

Slack มีแผนใช้งานฟรีที่มีฟีเจอร์พื้นฐาน และมีแผนเสียเงินสำหรับฟีเจอร์เพิ่มเติม เช่น การเก็บประวัติการสนทนาย้อนหลังทั้งหมดและการใช้งานร่วมกับแอปเสริมเพิ่มเติม

สามารถเรียนรู้เพิ่มเติมและสมัครใช้งานได้ที่ [Slack.com](https://slack.com) ค่ะ! 😊

**GitHub** เป็นแพลตฟอร์มออนไลน์ที่ใช้สำหรับการจัดเก็บและจัดการโค้ดโปรแกรม โดยรองรับการทำงานร่วมกันในโครงการต่าง ๆ ไม่ว่าจะเป็นโครงการส่วนตัวหรือโครงการแบบเปิดเผย (Open Source) นอกจากนี้ยังใช้ **Git** ซึ่งเป็นระบบควบคุมเวอร์ชัน (Version Control System) เพื่อช่วยติดตามการเปลี่ยนแปลงในไฟล์และซอร์สโค้ด

**คุณสมบัติหลักของ GitHub:**

1. **การจัดเก็บโค้ด (Repositories):**
   * โค้ดจะถูกเก็บไว้ในที่เรียกว่า **Repository (Repo)** ซึ่งทำหน้าที่เหมือนที่เก็บโครงการ
   * สามารถเป็นแบบ **สาธารณะ (Public)** หรือ **ส่วนตัว (Private)** ได้
2. **Version Control (ควบคุมเวอร์ชัน):**
   * ติดตามการเปลี่ยนแปลงของไฟล์โค้ด ทำให้สามารถย้อนกลับไปยังเวอร์ชันก่อนหน้าได้
3. **การทำงานร่วมกัน (Collaboration):**
   * นักพัฒนาหลายคนสามารถทำงานในโครงการเดียวกันได้อย่างมีประสิทธิภาพ
   * ใช้ **Pull Requests** เพื่อแนะนำการเปลี่ยนแปลงและ **Code Reviews** เพื่อตรวจสอบโค้ด
4. **Branching และ Merging:**
   * สามารถสร้าง **Branches** เพื่อพัฒนาโค้ดในฟีเจอร์ใหม่หรือแก้ไขปัญหา โดยไม่กระทบโค้ดหลัก
   * รวมโค้ดจาก Branch ต่าง ๆ เข้าด้วยกันผ่านการ **Merge**
5. **Issue Tracking:**
   * ใช้ติดตามข้อผิดพลาด (Bugs) หรือฟีเจอร์ใหม่ที่ต้องพัฒนา
6. **Actions (CI/CD):**
   * รองรับการตั้งค่าการทำงานอัตโนมัติ เช่น การทดสอบและการปรับใช้งาน (Deployment)
7. **Open Source Community:**
   * GitHub เป็นศูนย์กลางสำหรับโครงการ Open Source ทั่วโลก นักพัฒนาสามารถค้นหาและมีส่วนร่วมในโครงการต่าง ๆ ได้

**ประโยชน์ของ GitHub:**

* ช่วยให้ทีมพัฒนาซอฟต์แวร์ทำงานร่วมกันได้อย่างเป็นระบบ
* ลดปัญหาการสูญเสียโค้ดหรือการเขียนทับกัน
* ส่งเสริมการทำงานในโครงการ Open Source
* ใช้เป็นที่เก็บผลงาน (Portfolio) สำหรับนักพัฒนา

**การใช้งาน GitHub:**

* สร้างโปรเจกต์และอัปโหลดโค้ด
* ติดตามความคืบหน้าของการพัฒนา
* ร่วมมือในโครงการแบบทีม
* แบ่งปันโค้ดในชุมชนนักพัฒนา

คุณสามารถเรียนรู้เพิ่มเติมและเริ่มต้นใช้งานได้ที่ [GitHub.com](https://github.com)

**Docker** เป็นแพลตฟอร์มโอเพนซอร์สที่ช่วยให้การพัฒนาและการปรับใช้แอปพลิเคชันเป็นไปอย่างรวดเร็วและมีประสิทธิภาพ โดยใช้แนวคิดของ **คอนเทนเนอร์ (Container)** เพื่อบรรจุโค้ดและทุกสิ่งที่แอปพลิเคชันต้องการสำหรับการทำงาน ไม่ว่าจะเป็นไลบรารี, การตั้งค่าระบบปฏิบัติการ, หรือไฟล์ที่จำเป็น ซึ่งช่วยลดปัญหาความเข้ากันได้ของแอปพลิเคชันระหว่างเครื่องที่ต่างกัน

**คุณสมบัติหลักของ Docker:**

1. **คอนเทนเนอร์ (Containers):**
   * คอนเทนเนอร์เป็นเหมือนกล่องที่บรรจุแอปพลิเคชันพร้อมสภาพแวดล้อมที่จำเป็น
   * ช่วยให้แอปพลิเคชันทำงานเหมือนเดิมทุกที่ ไม่ว่าจะเป็นในเครื่องพัฒนา, เซิร์ฟเวอร์, หรือคลาวด์
2. **Docker Images:**
   * เป็นเหมือน "แม่แบบ" หรือ "สแนปชอต" สำหรับการสร้างคอนเทนเนอร์
   * Docker Image เก็บทุกอย่างที่จำเป็นสำหรับการรันแอปพลิเคชัน เช่น โค้ด, ไลบรารี, และการตั้งค่า
3. **การจัดการเวอร์ชัน:**
   * ติดตามการเปลี่ยนแปลงของ Docker Images ได้ ทำให้สามารถย้อนกลับไปยังเวอร์ชันก่อนหน้าได้หากจำเป็น
4. **การใช้งานร่วมกับ DevOps:**
   * ผสานการทำงานกับเครื่องมือ DevOps เช่น Jenkins, Kubernetes, และ CI/CD เพื่อปรับปรุงกระบวนการพัฒนาและปรับใช้
5. **ขนาดเบาและรวดเร็ว:**
   * คอนเทนเนอร์ใช้ทรัพยากรน้อยกว่าการใช้งาน Virtual Machines (VMs) และรันเร็วกว่า

**ประโยชน์ของ Docker:**

1. **ความสม่ำเสมอ (Consistency):**
   * แอปพลิเคชันที่รันบนคอนเทนเนอร์ทำงานเหมือนกันทุกที่ ลดปัญหาความแตกต่างระหว่างสภาพแวดล้อม
2. **ปรับปรุงการทำงานร่วมกัน:**
   * นักพัฒนาสามารถแบ่งปัน Docker Images เพื่อให้ทุกคนทำงานบนสภาพแวดล้อมเดียวกัน
3. **การปรับใช้ที่ง่ายและรวดเร็ว:**
   * แอปพลิเคชันใน Docker Container สามารถปรับใช้ได้ทันทีโดยไม่ต้องตั้งค่าใหม่
4. **ประหยัดทรัพยากร:**
   * คอนเทนเนอร์แชร์เคอร์เนลของระบบปฏิบัติการ ทำให้ใช้ทรัพยากรน้อยกว่าการรันบน Virtual Machines

**การใช้งาน Docker:**

* **การพัฒนา:** สร้างสภาพแวดล้อมที่เหมือนกันระหว่างนักพัฒนา
* **การทดสอบ:** รันแอปพลิเคชันในสภาพแวดล้อมแยกเพื่อการทดสอบ
* **การปรับใช้ (Deployment):** ปรับใช้แอปพลิเคชันได้อย่างรวดเร็วในสภาพแวดล้อมต่าง ๆ เช่น คลาวด์
* **Microservices:** จัดการแอปพลิเคชันที่แยกออกเป็นส่วนย่อย ๆ ที่ทำงานในคอนเทนเนอร์แยกกัน

**ตัวอย่างเครื่องมือที่ใช้งานร่วมกับ Docker:**

* **Docker Compose:** ใช้สำหรับจัดการคอนเทนเนอร์หลายตัวที่ทำงานร่วมกัน
* **Kubernetes:** ใช้สำหรับจัดการคอนเทนเนอร์ในระดับใหญ่
* **Docker Hub:** แพลตฟอร์มสำหรับแชร์และจัดเก็บ Docker Images

สามารถเริ่มต้นใช้งาน Docker ได้ที่ [Docker.com](https://www.docker.com)

ใน **Docker** คำว่า **"Container"** หมายถึงหน่วยที่แยกส่วนของแอปพลิเคชันที่สามารถรันได้อย่างอิสระ ซึ่งใช้เทคโนโลยีการจำลองระบบเพื่อแบ่งปันทรัพยากรของระบบปฏิบัติการเดียวกัน โดยไม่ต้องรันระบบปฏิบัติการแยกต่างหากเหมือนใน Virtual Machine (VM)

**คุณสมบัติของ Container:**

1. **แยกส่วน (Isolation):**
   * Container ทำงานอย่างอิสระจากกันและกัน และไม่ก้าวก่ายทรัพยากรของ Container อื่น
   * การแยกนี้ทำให้แอปพลิเคชันปลอดภัยและสามารถทดสอบหรือพัฒนาโดยไม่รบกวนระบบหลัก
2. **น้ำหนักเบา (Lightweight):**
   * Container ใช้ทรัพยากรน้อยกว่า Virtual Machine เพราะใช้ Kernel ของระบบปฏิบัติการเจ้าบ้าน (Host OS)
   * เริ่มต้นและหยุดทำงานได้รวดเร็ว
3. **พกพาได้ (Portable):**
   * Container สามารถรันได้ในทุกที่ที่มี Docker Engine เช่น บนเครื่อง Local, Cloud, หรือ Server ต่าง ๆ
4. **การจัดการ Dependency:**
   * Container รวมทุกอย่างที่แอปพลิเคชันต้องการ (เช่น โค้ด, ไลบรารี, การตั้งค่า) ไว้ในหน่วยเดียว ทำให้ไม่มีปัญหาเกี่ยวกับ Dependency

**เปรียบเทียบ Container กับ Virtual Machine (VM):**

| **คุณสมบัติ** | **Container** | **Virtual Machine (VM)** |
| --- | --- | --- |
| **น้ำหนักเบา** | ใช้ Kernel ของ Host OS | รัน OS แยกเต็มรูปแบบ |
| **ความเร็ว** | เริ่มทำงานเร็วมาก | ช้ากว่า (ต้องบูต OS) |
| **การใช้ทรัพยากร** | ใช้น้อยกว่า | ใช้ทรัพยากรมากกว่า |
| **ความแยกส่วน** | แยกเฉพาะแอปพลิเคชัน | แยกระบบเต็มรูปแบบ |

**ตัวอย่างการใช้งาน Docker Container:**

1. **สร้างและรัน Container:**

bash

Copy code

docker run -d --name my\_app nginx

คำสั่งนี้สร้าง Container ชื่อ my\_app และรันเซิร์ฟเวอร์ Nginx

1. **แสดงรายการ Container ที่กำลังทำงาน:**

bash

Copy code

docker ps

1. **หยุดและลบ Container:**

bash

Copy code

docker stop my\_app

docker rm my\_app

Docker Container เหมาะกับการพัฒนาและรันแอปพลิเคชันที่ต้องการความยืดหยุ่นและประสิทธิภาพสูง เช่น การพัฒนา Microservices หรือการจัดการเวิร์กโหลดในระบบคลาวด์

ls =

pwd

172.21.20.120

docker run --name docker-nginx -p 80:80 -d -v ~/docker-nginx/html:/usr/share/nginx/html nginx

docker run --name docker-nginx -p 80:80 -d -v .\Desktop\docker-nginx\html:/usr/share/nginx/html nginxคำสั่งที่คุณให้มาเป็นคำสั่งสำหรับการสร้างและรัน Docker Container โดยใช้ **Nginx** ซึ่งเป็นเว็บเซิร์ฟเวอร์ คำสั่งนี้มีการกำหนดชื่อ, การเชื่อมต่อพอร์ต, การรันแบบแบ็กกราวด์ และการเชื่อมโยงโฟลเดอร์ในเครื่องกับโฟลเดอร์ใน Container รายละเอียดของคำสั่งคือ:

**คำสั่ง:**

bash

Copy code

docker run --name docker-nginx -p 80:80 -d -v .\Desktop\docker-nginx\html:/usr/share/nginx/html nginx

**อธิบายส่วนต่าง ๆ:**

1. **docker run**  
   คำสั่งนี้ใช้เพื่อสร้างและรัน Container ใหม่จาก Docker Image
2. **--name docker-nginx**  
   กำหนดชื่อ Container เป็น docker-nginx เพื่อให้ง่ายต่อการอ้างอิงในคำสั่งอื่น ๆ
3. **-p 80:80**  
   ใช้สำหรับการแมปพอร์ต:
   * **พอร์ตแรก (80):** พอร์ตของเครื่อง Host (เครื่องของคุณ)
   * **พอร์ตที่สอง (80):** พอร์ตที่ Container ใช้ (ในที่นี้คือพอร์ตของ Nginx)

ดังนั้น คุณสามารถเข้าถึง Nginx ผ่าน http://localhost หรือ http://127.0.0.1 บนเครื่องของคุณ

1. **-d**  
   รัน Container ในโหมดแบ็กกราวด์ (detached mode) ซึ่งหมายความว่าคุณจะไม่เห็นผลลัพธ์หรือข้อความจาก Container ในเทอร์มินัลโดยตรง
2. **-v .\Desktop\docker-nginx\html:/usr/share/nginx/html**  
   ใช้สำหรับการแมปโฟลเดอร์ในเครื่อง (Volume Mapping):
   * **ฝั่งซ้าย (.\Desktop\docker-nginx\html):**  
     โฟลเดอร์บนเครื่องของคุณที่มีไฟล์ HTML หรือไฟล์อื่น ๆ ที่ต้องการให้ Nginx เสิร์ฟ
   * **ฝั่งขวา (/usr/share/nginx/html):**  
     โฟลเดอร์ใน Container ที่ Nginx ใช้เป็น root directory สำหรับการให้บริการไฟล์

ด้วยการแมปนี้ การเปลี่ยนแปลงไฟล์ใน .\Desktop\docker-nginx\html จะส่งผลกับไฟล์ใน Container โดยอัตโนมัติ

1. **nginx**  
   ชื่อของ Docker Image ที่ใช้สร้าง Container ในกรณีนี้คือ Image ของ Nginx

**การทำงานของคำสั่ง:**

เมื่อรันคำสั่งนี้:

1. Docker จะดาวน์โหลด Image ของ Nginx (ถ้ายังไม่มีในเครื่อง)
2. สร้าง Container ชื่อ docker-nginx
3. แมปพอร์ต 80 ของเครื่อง Host กับพอร์ต 80 ของ Container
4. แมปโฟลเดอร์ในเครื่อง (.\Desktop\docker-nginx\html) กับโฟลเดอร์ /usr/share/nginx/html ใน Container
5. รัน Container ในโหมดแบ็กกราวด์

**ผลลัพธ์:**

* คุณสามารถเปิดเบราว์เซอร์และเข้าถึงเว็บไซต์ผ่าน http://localhost หรือ http://127.0.0.1
* ไฟล์ HTML หรือไฟล์อื่น ๆ ในโฟลเดอร์ .\Desktop\docker-nginx\html จะถูกเสิร์ฟโดย Nginx

**ตัวอย่าง:**

ถ้าคุณมีไฟล์ index.html อยู่ใน .\Desktop\docker-nginx\html ที่มีเนื้อหา:

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Welcome</title>

</head>

<body>

<h1>Hello from Docker Nginx!</h1>

</body>

</html>

เมื่อเปิด http://localhost ในเบราว์เซอร์ คุณจะเห็นข้อความ **"Hello from Docker Nginx!"**

4o