ชื่อ-นามสกุล\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_รหัสนักศึกษา\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Section\_\_\_\_\_\_\_

4

653380197-7

ณัฐภัทร ตรงวัฒนาวุฒิ

**Lab#8 – Software Deployment Using Docker**

# วัตถุประสงค์การเรียนรู้

1. ผู้เรียนสามารถอธิบายเกี่ยวกับ Software deployment ได้
2. ผู้เรียนสามารถสร้างและรัน Container จาก Docker image ได้
3. ผู้เรียนสามารถสร้าง Docker files และ Docker images ได้
4. ผู้เรียนสามารถนำซอฟต์แวร์ที่พัฒนาขึ้นให้สามารถรันบนสภาพแวดล้อมเดียวกันและทำงานร่วมกันกับสมาชิกในทีมพัฒนาซอฟต์แวร์ผ่าน Docker hub ได้
5. ผู้เรียนสามารถเริ่มต้นใช้งาน Jenkins เพื่อสร้าง Pipeline ในการ Deploy งานได้

# Pre-requisite

1. ติดตั้ง Docker desktop ลงบนเครื่องคอมพิวเตอร์ โดยดาวน์โหลดจาก <https://www.docker.com/get-started>
2. สร้าง Account บน Docker hub (<https://hub.docker.com/signup>)
3. กำหนดให้ $ หมายถึง Command prompt และ <> หมายถึง ให้ป้อนค่าของพารามิเตอร์ที่กำหนด

# แบบฝึกปฏิบัติที่ 8.1 Hello world - รัน Container จาก Docker image

1. เปิดใช้งาน Docker desktop และ Login ด้วย Username และ Password ที่ลงทะเบียนกับ Docker Hub เอาไว้
2. เปิด Command line หรือ Terminal บน Docker Desktop จากนั้นสร้าง Directory ชื่อ Lab8\_1
3. ย้ายตำแหน่งปัจจุบันไปที่ Lab8\_1 เพื่อใช้เป็น Working directory
4. ป้อนคำสั่ง $ docker pull busybox หรือ $ sudo docker pull busybox สำหรับกรณีที่ติดปัญหา Permission denied

(หมายเหตุ: BusyBox เป็น software suite ที่รองรับคำสั่งบางอย่างบน Unix - https://busybox.net)

1. ป้อนคำสั่ง $ docker images

**[Check point#1]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้ พร้อมกับตอบคำถามต่อไปนี้

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

Repository หมายถึงชื่อของ Image ที่ถูกดึงมาจาก Docker Hub หรือ Registry อื่น เช่น busybox ในตัวอย่างนี้

1. สิ่งที่อยู่ภายใต้คอลัมน์ Repository คืออะไร\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tag ใช้บ่งบอกถึง **เวอร์ชัน** หรือ **สถานะ** ของ Image ตัวนั้น เช่น latest คือ Tag ที่ใช้ระบุเวอร์ชันล่าสุด

1. Tag ที่ใช้บ่งบอกถึงอะไร\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ป้อนคำสั่ง $ docker run busybox
2. ป้อนคำสั่ง $ docker run -it busybox sh
3. ป้อนคำสั่ง ls
4. ป้อนคำสั่ง ls -la
5. ป้อนคำสั่ง exit
6. ป้อนคำสั่ง $ docker run busybox echo “Hello ชื่อและนามสกุลของนักศึกษา from busybox”
7. ป้อนคำสั่ง $ docker ps -a

**[Check point#2]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้ตั้งแต่ขั้นตอนที่ 6-12 พร้อมกับตอบคำถามต่อไปนี้

A screenshot of a computer

Description automatically generated

1. เมื่อใช้ option -it ในคำสั่ง run ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป

Option -it คือการรวม -i (interactive mode) และ -t (allocate a pseudo-TTY) ซึ่งทำให้สามารถโต้ตอบกับ container ได้แบบ interactive ผ่าน terminal (เช่น ใช้คำสั่ง sh หรือ bash ได้).

* -i ทำให้ container ยังคงเปิดอยู่และรับ input จากผู้ใช้.
* -t สร้าง terminal จำลองสำหรับการพิมพ์คำสั่งและแสดงผลลัพธ์.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. คอลัมน์ STATUS จากการรันคำสั่ง docker ps -a แสดงถึงข้อมูลอะไร

คอลัมน์ STATUS ใช้แสดงสถานะของ container ว่ากำลังทำงานหรือหยุดทำงานแล้ว โดยประกอบด้วย:

* **"Up X seconds/minutes"**: แสดงว่า container กำลังรันอยู่ในช่วงเวลาที่ระบุ.
* **"Exited (Code) X seconds/minutes ago"**: แสดงว่า container ได้หยุดทำงานแล้ว พร้อมรหัสสถานะการหยุด (Code).
* **"Created"**: แสดงว่า container ถูกสร้างขึ้น แต่ยังไม่เริ่มทำงาน

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ป้อนคำสั่ง $ docker rm <container ID ที่ต้องการลบ>

**[Check point#3]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้ในขั้นตอนที่ 13

A computer screen shot of a black screen

Description automatically generated

# แบบฝึกปฏิบัติที่ 8.2: สร้าง Docker file และ Docker image

1. เปิดใช้งาน Docker desktop และ Login ด้วย Username และ Password ที่ลงทะเบียนกับ Docker Hub เอาไว้
2. เปิด Command line หรือ Terminal จากนั้นสร้าง Directory ชื่อ Lab8\_2
3. ย้ายตำแหน่งปัจจุบันไปที่ Lab8\_2 เพื่อใช้เป็น Working directory
4. สร้าง Dockerfile.swp ไว้ใน Working directory

สำหรับเครื่องที่ใช้ระบบปฏิบัติการวินโดวส์ (Windows) บันทึกคำสั่งต่อไปนี้ลงในไฟล์ โดยใช้ Text Editor ที่มี

FROM busybox

CMD echo “Hi there. This is my first docker image.”

CMD echo “ชื่อ-นามสกุล รหัสนักศึกษา ชื่อเล่น”

สำหรับเครื่องที่ใช้ระบบปฏิบัติการ MacOS หรือ Linux บนหน้าต่าง Terminal และป้อนคำสั่งต่อไปนี้

$ cat > Dockerfile << EOF

FROM busybox

CMD echo “Hi there. This is my first docker image.”

CMD echo “ชื่อ-นามสกุล รหัสนักศึกษา ชื่อเล่น”

EOF

หรือใช้คำสั่ง

$ touch Dockerfile

แล้วใช้ Text Editor ในการใส่เนื้อหาแทน

1. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

$ docker build -t <ชื่อ Image> .

1. เมื่อ Build สำเร็จแล้ว ให้ทำการรัน Docker image ที่สร้างขึ้นในขั้นตอนที่ 5

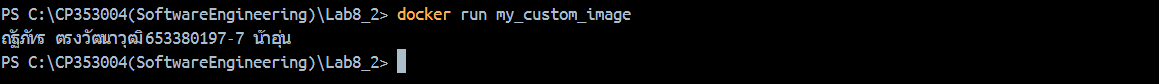
**[Check point#4]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้ในขั้นตอนที่ 5 พร้อมกับตอบคำถามต่อไปนี้

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated



1. คำสั่งที่ใช้ในการ run คือ

docker run <ชื่อ Image>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Option -t ในคำสั่ง $ docker build ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป

Option -t ใช้สำหรับกำหนดชื่อ (tag) ให้กับ Docker Image ที่กำลังถูกสร้าง.

* ช่วยให้สามารถอ้างอิง Image ได้สะดวกโดยใช้ชื่อที่กำหนด (แทนการอ้างอิงด้วย Image ID).
* ชื่อ Image ที่สร้างจะอยู่ในรูปแบบ <Repository>:<Tag> เช่น myfirstimage:latest (ถ้าไม่ได้กำหนด Tag เฉพาะ จะใช้ latest เป็นค่าเริ่มต้น)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# แบบฝึกปฏิบัติที่ 8.3: การแชร์ Docker image ผ่าน Docker Hub

1. เปิดใช้งาน Docker desktop และ Login ด้วย Username และ Password ที่ลงทะเบียนกับ Docker Hub เอาไว้
2. เปิด Command line หรือ Terminal จากนั้นสร้าง Directory ชื่อ Lab8\_3
3. ย้ายตำแหน่งปัจจุบันไปที่ Lab8\_3 เพื่อใช้เป็น Working directory
4. สร้าง Dockerfile.swp ไว้ใน Working directory

สำหรับเครื่องที่ใช้ระบบปฏิบัติการวินโดวส์ บันทึกคำสั่งต่อไปนี้ลงในไฟล์ โดยใช้ Text Editor ที่มี

FROM busybox

CMD echo “Hi there. My work is done. You can run them from my Docker image.”

CMD echo “ชื่อ-นามสกุล รหัสนักศึกษา”

สำหรับเครื่องที่ใช้ระบบปฏิบัติการ MacOS หรือ Linux บนหน้าต่าง Terminal และป้อนคำสั่งต่อไปนี้

$ cat > Dockerfile << EOF

FROM busybox

CMD echo “Hi there. My work is done. You can run them from my Docker image.”

CMD echo “ชื่อ-นามสกุล รหัสนักศึกษา”

EOF

หรือใช้คำสั่ง

$ touch Dockerfile

แล้วใช้ Text Editor ในการใส่เนื้อหาแทน

1. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

$ docker build -t <username ที่ลงทะเบียนกับ Docker Hub>/lab8

1. ทำการรัน Docker image บน Container ในเครื่องของตัวเองเพื่อทดสอบผลลัพธ์ ด้วยคำสั่ง

$ docker run <username ที่ลงทะเบียนกับ Docker Hub>/lab8

**[Check point#5]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้ในขั้นตอนที่ 5

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

1. ทำการ Push ตัว Docker image ไปไว้บน Docker Hub โดยการใช้คำสั่ง

$ docker push <username ที่ลงทะเบียนกับ Docker Hub>/lab8

ในกรณีที่ติดปัญหาไม่ได้ Login ไว้ก่อน ให้ใช้คำสั่งต่อไปนี้ เพื่อ Login ก่อนทำการ Push

$ docker login แล้วป้อน Username และ Password ตามที่ระบุใน Command prompt หรือใช้คำสั่ง

$ docker login -u <username> -p <password>

1. ไปที่ Docker Hub กด Tab ชื่อ Tags หรือไปที่ Repository ก็ได้

**[Check point#6]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดง Repository ที่มี Docker image (<username>/lab8)

A computer screen with white text

Description automatically generated

# แบบฝึกปฏิบัติที่ 8.4: การ Build แอปพลิเคชันจาก Container image และการ Update แอปพลิเคชัน

1. เปิด Command line หรือ Terminal จากนั้นสร้าง Directory ชื่อ Lab8\_4
2. ทำการ Clone ซอร์สโค้ดของเว็บแอปพลิเคชันจาก GitHub repository <https://github.com/docker/getting-started.git> ลงใน Directory ที่สร้างขึ้น โดยใช้คำสั่ง

$ git clone https://github.com/docker/getting-started.git

1. เปิดดูองค์ประกอบภายใน getting-started/app เมื่อพบไฟล์ package.json ให้ใช้ Text editor ในการเปิดอ่าน

**[Check point#7]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงที่อยู่ของ Source code ที่ Clone มาและเนื้อหาของไฟล์ package.json

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. ภายใต้ getting-started/app ให้สร้าง Dockerfile พร้อมกับใส่เนื้อหาดังต่อไปนี้ลงไปในไฟล์

FROM node:18-alpine

WORKDIR /app

COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000

1. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้ โดยกำหนดใช้ชื่อ image เป็น myapp\_รหัสนศ. ไม่มีขีด

$ docker build -t <myapp\_รหัสนศ. ไม่มีขีด> .

**[Check point#8]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงคำสั่งและผลลัพธ์ที่ได้ทางหน้าจอ

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

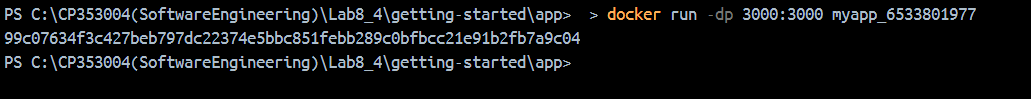
Description automatically generated

1. ทำการ Start ตัว Container ของแอปพลิเคชันที่สร้างขึ้น โดยใช้คำสั่ง

$ docker run -dp 3000:3000 <myapp\_รหัสนศ. ไม่มีขีด>

1. เปิด Browser ไปที่ URL = <http://localhost:3000>

**[Check point#9]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้บน Browser และ Dashboard ของ Docker desktop



A screenshot of a computer

Description automatically generated

หมายเหตุ: นศ.สามารถทดลองเล่น Web application ที่ทำงานอยู่ได้

1. ทำการแก้ไข Source code ของ Web application ดังนี้
   1. เปิดไฟล์ src/static/js/app.js ด้วย Editor และแก้ไขบรรทัดที่ 56 จาก

<p className="text-center">No items yet! Add one above!</p> เป็น

<p className="text-center">**There is no TODO item. Please add one to the list. By ชื่อและนามสกุลของนักศึกษา**</p>

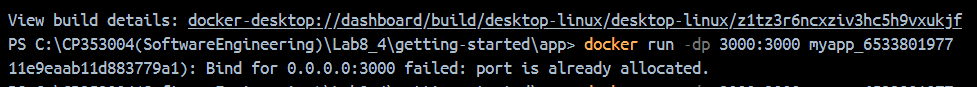
* 1. Save ไฟล์ให้เรียบร้อย

1. ทำการ Build Docker image โดยใช้คำสั่งเดียวกันกับข้อ 5
2. Start และรัน Container ตัวใหม่ โดยใช้คำสั่งเดียวกันกับข้อ 6

**[Check point#10]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงคำสั่งและผลลัพธ์ที่ได้ทางหน้าจอ พร้อมกับตอบคำถามต่อไปนี้

A computer screen shot of a computer code

Description automatically generated



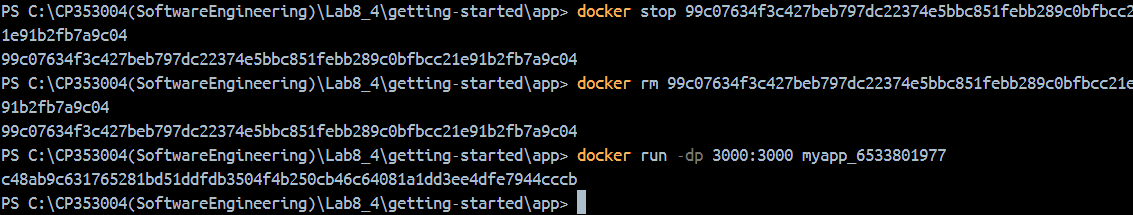
1. Error ที่เกิดขึ้นหมายความอย่างไร และเกิดขึ้นเพราะอะไร

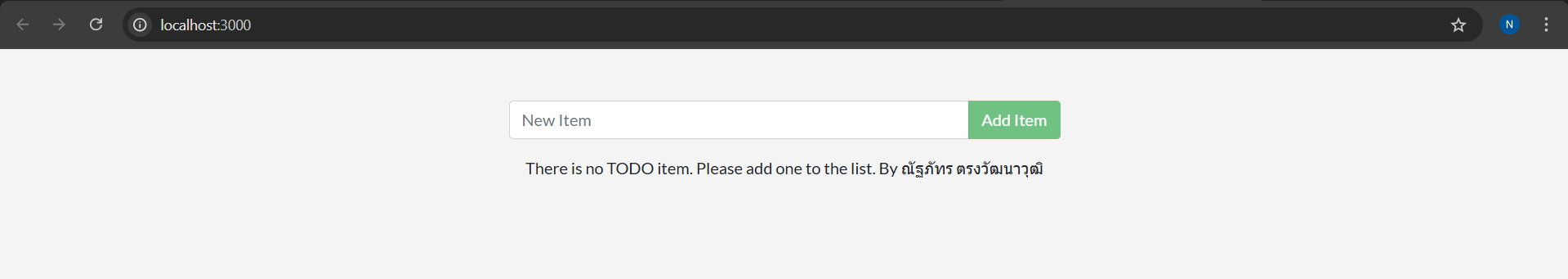
ข้อผิดพลาด **"Bind for 0.0.0.0:3000 failed: port is already allocated"** หมายความว่า Port **3000** ที่พยายามใช้งานใน Docker ถูกใช้งานอยู่แล้วจากโปรแกรมหรือ container อื่นบนเครื่องทำให้ Docker ไม่สามารถใช้ port นี้ได้

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ลบ Container ของ Web application เวอร์​ชันก่อนแก้ไขออกจากระบบ โดยใช้วิธีใดวิธีหนึ่งดังต่อไปนี้
   1. ผ่าน Command line interface
      1. ใช้คำสั่ง $ docker ps เพื่อดู Container ID ที่ต้องการจะลบ
      2. Copy หรือบันทึก Container ID ไว้
      3. ใช้คำสั่ง $ docker stop <Container ID ที่ต้องการจะลบ> เพื่อหยุดการทำงานของ Container ดังกล่าว
      4. ใช้คำสั่ง $ docker rm <Container ID ที่ต้องการจะลบ> เพื่อทำการลบ
   2. ผ่าน Docker desktop
      1. ไปที่หน้าต่าง Containers
      2. เลือกไอคอนถังขยะในแถวของ Container ที่ต้องการจะลบ
      3. ยืนยันโดยการกด Delete forever
2. Start และรัน Container ตัวใหม่อีกครั้ง โดยใช้คำสั่งเดียวกันกับข้อ 6
3. เปิด Browser ไปที่ URL = <http://localhost:3000>

**[Check point#11]** Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้บน Browser และ Dashboard ของ Docker desktop





# แบบฝึกปฏิบัติที่ 8.5: เริ่มต้นสร้าง Pipeline อย่างง่ายสำหรับการ Deploy ด้วย Jenkins

1. เปิด Command line หรือ Terminal บน Docker Desktop
2. ป้อนคำสั่งและทำการรัน container โดยผูกพอร์ต

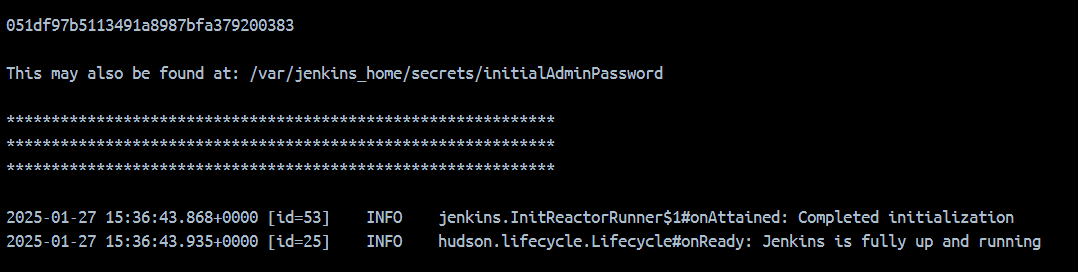
$ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17

หรือ

$ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure -v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts-jdk17

1. บันทึกรหัสผ่านของ Admin user ไว้สำหรับ log-in ในครั้งแรก

**[Check point#12]** Capture หน้าจอที่แสดงผล Admin password



1. เมื่อได้รับการยืนยันว่า Jenkins is fully up and running ให้เปิดบราวเซอร์ และป้อนที่อยู่เป็น localhost:8080
2. ทำการ Unlock Jenkins ด้วยรหัสผ่านที่ได้ในข้อที่ 3
3. สร้าง Admin User โดยใช้ username เป็นชื่อจริงของนักศึกษาพร้อมรหัสสี่ตัวท้าย เช่น somsri\_3062

**[Check point#13]** Capture หน้าจอที่แสดงผลการตั้งค่า

A screenshot of a computer

Description automatically generated

1. กำหนด Jenkins URL เป็น <http://localhost:8080/lab8>
2. เมื่อติดตั้งเรียบร้อยแล้วจะพบกันหน้า Dashboard ดังแสดงในภาพ

A screenshot of a computer

AI-generated content may be incorrect.

1. เลือก Manage Jenkins แล้วไปที่เมนู Plugins

A screenshot of a computer

AI-generated content may be incorrect.

1. ไปที่เมนู Available plugins แล้วเลือกติดตั้ง Robotframework เพิ่มเติม

A screenshot of a computer

AI-generated content may be incorrect.

1. กลับไปที่หน้า Dashboard แล้วสร้าง Pipeline อย่างง่าย โดยกำหนด New item เป็น Freestyle project และตั้งชื่อเป็น UAT

A screenshot of a computer

AI-generated content may be incorrect.

1. นำไฟล์ .robot ที่ทำให้แบบฝึกปฏิบัติที่ 7 (Lab#7) ไปไว้บน Repository ของนักศึกษา จากนั้นตั้งค่าที่จำเป็นในหน้านี้ทั้งหมด ดังนี้

**Description:** Lab 8.5

**GitHub project**: กดเลือก แล้วใส่ Project URL เป็น repository ที่เก็บโค้ด .robot (ดูขั้นตอนที่ 12)

**Build Trigger:** เลือกแบบ Build periodically แล้วกำหนดให้ build ทุก 15 นาที

**Build Steps:** เลือก Execute shell แล้วใส่คำสั่งในการรันไฟล์ .robot (หากไฟล์ไม่ได้อยู่ในหน้าแรกของ repository ให้ใส่ Path ไปถึงไฟล์ให้เรียบร้อยด้วย)

**[Check point#14]** Capture หน้าจอแสดงการตั้งค่า พร้อมกับตอบคำถามต่อไปนี้

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. คำสั่งที่ใช้ในการ Execute ไฟล์ .robot ใน Build Steps คือ

robot completely.robot

robot incompletely.robot

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Post-build action:** เพิ่ม Publish Robot Framework test results -> ระบุไดเร็คทอรีที่เก็บไฟล์ผลการทดสอบโดย Robot framework ในรูป xml และ html -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ไม่ผ่านแล้วนับว่าซอฟต์แวร์มีปัญหา -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ผ่านแล้วนับว่าซอฟต์แวร์มีอยู่ในสถานะที่สามารถนำไปใช้งานได้ (เช่น 20, 80)

1. กด Apply และ Save
2. สั่ง Build Now

**[Check point#15]** Capture หน้าจอแสดงหน้าหลักของ Pipeline และ Console Output

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated