

## Lab Worksheet

ชื่อ-นามสกุล \_\_\_\_\_ นายณดล มุลลาลาด \_\_\_\_\_ รหัสนักศึกษา \_\_\_\_\_ 653380325 - 4 \_\_\_\_\_ Section \_\_\_\_\_ 4 \_\_\_\_\_

## 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
1. เปิด Command line หรือ Terminal บน Docker Desktop จากนั้นสร้าง Directory ชื่อ Lab8\_1
2. ย้ายตำแหน่งปัจจุบันไปที่ Lab8\_1 เพื่อใช้เป็น Working directory
3. ป้อนคำสั่ง \$ docker pull busybox หรือ \$ sudo docker pull busybox สำหรับกรณีที่ติดปัญหา Permission denied  
(หมายเหตุ: BusyBox เป็น software suite ที่รองรับคำสั่งบางอย่างบน Unix - <https://busybox.net>)
4. ป้อนคำสั่ง \$ docker images

## Lab Worksheet

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

```
PS C:\Users\DRAGON\Lab8_1> docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
97e70d161e81: Pull complete
Digest: sha256:37f7b378a29ceb4c551b1b5582e27747b855bbfaa73fa11914fe0df028dc581f
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
PS C:\Users\DRAGON\Lab8_1> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
busybox	latest	ff7a7936e930	5 months ago	4.28MB

```
PS C:\Users\DRAGON\Lab8_1>
```

(1) สิ่งที่อยู่ภายใต้คอลัมน์ Repository คืออะไร

- ในบริบทของ Docker, คอลัมน์ Repository ในผลลัพธ์ของคำสั่ง docker images หรือ docker image ls แสดงชื่อของ repository ที่ภาพ (image) นั้นถูกเก็บไว้
- Repository ใน Docker Hub หรือ registry อื่น ๆ เป็นที่เก็บรวบรวมของ Docker images ที่เกี่ยวข้องกัน โดยปกติจะใช้เพื่อจัดเก็บภาพที่เกี่ยวข้องกับแอปพลิเคชันหรือบริการเฉพาะ
- ตัวอย่างเช่น, ถ้าคุณเห็น busybox ในคอลัมน์ Repository, หมายความว่า Docker image นั้นมาจาก repository ที่ชื่อ busybox

(2) Tag ที่ใช้บ่งบอกถึงอะไร

- ในบริบทของ Docker, คอลัมน์ Repository ในผลลัพธ์ของคำสั่ง docker images หรือ docker image ls แสดงชื่อของ repository ที่ภาพ (image) นั้นถูกเก็บไว้
- Repository ใน Docker Hub หรือ registry อื่น ๆ เป็นที่เก็บรวบรวมของ Docker images ที่เกี่ยวข้องกัน โดยปกติจะใช้เพื่อจัดเก็บภาพที่เกี่ยวข้องกับแอปพลิเคชันหรือบริการเฉพาะ
- ตัวอย่างเช่น, ถ้าคุณเห็น busybox ในคอลัมน์ Repository, หมายความว่า Docker image นั้นมาจาก repository ที่ชื่อ busybox

5. ป้อนคำสั่ง \$ docker run busybox

6. ป้อนคำสั่ง \$ docker run -it busybox sh

## Lab Worksheet

7. ป้อนคำสั่ง ls
8. ป้อนคำสั่ง ls -la
9. ป้อนคำสั่ง exit
10. ป้อนคำสั่ง \$ docker run busybox echo "Hello ชื่อและนามสกุลของนักศึกษา from busybox"
11. ป้อนคำสั่ง \$ docker ps -a

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

```
drwxr-xr-x  4 root   root       4096 Sep 26 21:31 var
/ # exit
PS C:\Users\DRAGON\Lab8_1> docker run busybox echo "Hello นายคณ มฤตลาต from busybox"
Hello นายคณ มฤตลาต from busybox
PS C:\Users\DRAGON\Lab8_1> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3ac7a7181f5c	busybox	"echo 'Hello นายคณ ...'"	12 seconds ago	Exited (0) 11 seconds ago		optimistic_mayer
455d99eb3b2e	busybox	"sh"	About a minute ago	Exited (0) 54 seconds ago		upbeat_gates
498b05e43409	busybox	"sh"	About a minute ago	Exited (0) About a minute ago		blissful_keltysh

```
PS C:\Users\DRAGON\Lab8_1>
```

(1) เมื่อใช้ option -it ในคำสั่ง run ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป

a. -i หรือ --interactive:

- i. ตัวเลือกนี้จะทำให้คอนเทนเนอร์ที่รันอยู่สามารถรับข้อมูลจาก standard input (STDIN) ได้ ซึ่งหมายความว่า คุณสามารถพิมพ์คำสั่งหรือข้อมูลเข้าไปในคอนเทนเนอร์ได้ในขณะที่มันกำลังทำงาน

b. -t หรือ --tty:

- i. ตัวเลือกนี้จะจัดสรร pseudo-TTY ให้กับคอนเทนเนอร์ ซึ่งจะจำลอง terminal interface ทำให้การแสดงผลและการรับข้อมูลในคอนเทนเนอร์มีลักษณะเหมือนกับการทำงานใน terminal ปกติ

การใช้ -it ร่วมกันจะทำให้คุณสามารถโต้ตอบกับคอนเทนเนอร์ได้อย่างเต็มที่ เช่น การเปิดเชลล์ภายในคอนเทนเนอร์เพื่อพิมพ์คำสั่งต่าง ๆ หรือการรันแอปพลิเคชันที่ต้องการการโต้ตอบกับผู้ใช้ผ่าน terminal

โดยทั่วไป, การใช้ -it จะมีประโยชน์เมื่อคุณต้องการเข้าไปจัดการหรือทดสอบภายในคอนเทนเนอร์แบบ interactive.

## Lab Worksheet

(2) คอลัมน์ STATUS จากการรันคำสั่ง `docker ps -a` แสดงถึงข้อมูลอะไร

- a. คำสั่งนี้แสดงรายการคอนเทนเนอร์ทั้งหมดที่เคยรันบนระบบ ไม่ว่าจะกำลังรันอยู่หรือหยุดทำงานแล้ว รวมถึงรายละเอียดต่าง ๆ เช่น ชื่อคอนเทนเนอร์, สถานะ, และคำสั่งที่ใช้รัน

12. ป้อนคำสั่ง `$ docker rm <container ID ที่ต้องการลบ>`

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

### 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

## Lab Worksheet

5. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

\$ docker build -t <ชื่อ Image> .

6. เมื่อ Build สำเร็จแล้ว ให้ทำการรัน Docker image ที่สร้างขึ้นในขั้นตอนที่ 5

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

```
PS C:\Users\DRAGON\Lab8-2> docker build t my_custom_image f Dockerfile.txt .
ERROR: docker: 'docker buildx build' requires 1 argument

Usage:  docker buildx build [OPTIONS] PATH | URL | -

Run 'docker buildx build --help' for more information
PS C:\Users\DRAGON\Lab8-2> ^C
PS C:\Users\DRAGON\Lab8-2> docker build -t my_custom_image -f Dockerfile.txt .
[+] Building 0.2s (5/5) FINISHED
=> [internal] load build definition from Dockerfile.txt
=> => transferring dockerfile: 196B
=> WARN: JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 2)
=> WARN: MultipleInstructionsDisallowed: Multiple CMD instructions should not be used in the same stage because only the last one will be used (line 2)
=> WARN: JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 3)
=> [internal] load metadata for docker.io/library/busybox:latest
=> exporting to image
=> => exporting layers
=> => writing image sha256:8719cfbfe9a70b098b11e3756827321c7af8a6c8821001ef13b2ec6276a2f9c4
=> => naming to docker.io/library/my_custom_image

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/0xelzf3wrz288kmwunxzkvsw

3 warnings found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 2)
- MultipleInstructionsDisallowed: Multiple CMD instructions should not be used in the same stage because only the last one will be used (line 2)
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 3)
PS C:\Users\DRAGON\Lab8-2> docker run my_custom_image
"นายกมล นฤมล 653380325-4 แห่ง"
```

(1) คำสั่งที่ใช้ในการ run คือ

a. docker run my\_custom\_image

(2) Option -t ในคำสั่ง \$ docker build ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป

a. ออปชัน -t ในคำสั่ง docker build ใช้สำหรับการตั้งชื่อและแท็กให้กับอิมเมจ ซึ่งทำให้ง่ายต่อการอ้างอิงอิมเมจในภายหลัง ไม่ว่าจะเป็นการรัน การส่งไปยังรีจิสทรี หรือการแชร์กับผู้อื่น

i. รูปแบบ: -t <name>:<tag>

1. <name>: ชื่อที่คุณต้องการตั้งให้กับอิมเมจ

2. <tag>: แท็กที่ใช้ระบุเวอร์ชันหรือรูปแบบของอิมเมจ ถ้าไม่ได้ระบุจะ

ใช้ latest เป็นค่าเริ่มต้น

## Lab Worksheet

b. ตัวอย่าง:

i. `docker build -t my_custom_image:1.0 .`

c. ในตัวอย่างนี้ `my_custom_image` คือชื่อของอิมเมจ และ `1.0` คือแท็ก ซึ่งช่วยให้คุณจัดการเวอร์ชันต่าง ๆ ของอิมเมจได้ง่ายขึ้น หากคุณไม่ได้ระบุแท็ก Docker จะใช้ `latest` เป็นค่าเริ่มต้น

d. การใช้ `docker build -t` เป็นแนวปฏิบัติที่ดีเพราะช่วยในการจัดระเบียบและระบุอิมเมจได้ง่าย โดยเฉพาะเมื่อคุณต้องจัดการกับหลายเวอร์ชันหรือรูปแบบต่าง ๆ ครับ

### 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 และบันทึกคำสั่งต่อไปนี้

```
$ 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
```

## Lab Worksheet

แล้วใช้ Text Editor ในการใส่เนื้อหาแทน

7. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

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

5. ทำการรัน Docker image บน Container ในเครื่องของตัวเองเพื่อทดสอบผลลัพธ์ ด้วยคำสั่ง

```
$ docker run <username ที่ลงทะเบียนกับ Docker Hub>/lab8
```

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

```
Terminal

=> [internal] load build definition from Dockerfile.txt
=> => transferring dockerfile: 208B
=> [internal] load metadata for docker.io/library/busybox:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/1] FROM docker.io/library/busybox:latest
=> exporting to image
=> => exporting layers
=> => writing image sha256:64f68de1ca6994c0e06c6a80b01b470c8b92b904f115f5c3e4514f7716999f0f
=> => naming to docker.io/library/nadon

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/v90d08itre2aaf0w2xgc91iu6

3 warnings found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 2)
- MultipleInstructionsDisallowed: Multiple CMD instructions should not be used in the same stage because only the last one will be used (line 2)
=> => writing image sha256:64f68de1ca6994c0e06c6a80b01b470c8b92b904f115f5c3e4514f7716999f0f
=> => naming to docker.io/nadon/lab8

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/qcyfxyb8wcjmf2q187wfhf508

3 warnings found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 2)
- MultipleInstructionsDisallowed: Multiple CMD instructions should not be used in the same stage because only the last one will be used (line 2)
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 3)
PS C:\Users\DRAGON\Lab8_3> docker run nadon/lab8
"นายณัฏฐ มุสิกมศักดิ์ 653380325-4"
PS C:\Users\DRAGON\Lab8_3>
```

6. ทำการ Push ตัว Docker image ไปไว้บน Docker Hub โดยการใช้คำสั่ง

```
$ docker push <username ที่ลงทะเบียนกับ Docker Hub>/lab8
```

ในกรณีที่ติดปัญหาไม่ได้ Login ไว้ก่อน ให้ใช้คำสั่งต่อไปนี้ เพื่อ Login ก่อนทำการ Push

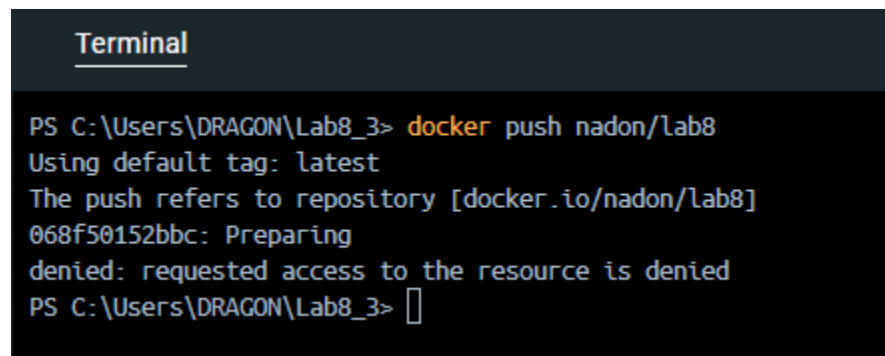
```
$ docker login แล้วป้อน Username และ Password ตามที่ระบุใน Command prompt หรือใช้คำสั่ง
```

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

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

## Lab Worksheet

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



```
Terminal
PS C:\Users\DRAGON\Lab8_3> docker push nadon/lab8
Using default tag: latest
The push refers to repository [docker.io/nadon/lab8]
068f50152bbc: Preparing
denied: requested access to the resource is denied
PS C:\Users\DRAGON\Lab8_3> 
```

---

แบบฝึกปฏิบัติที่ 8.4: การ Build แอปพลิเคชันจาก Container image และการ Update แอปพลิเคชัน

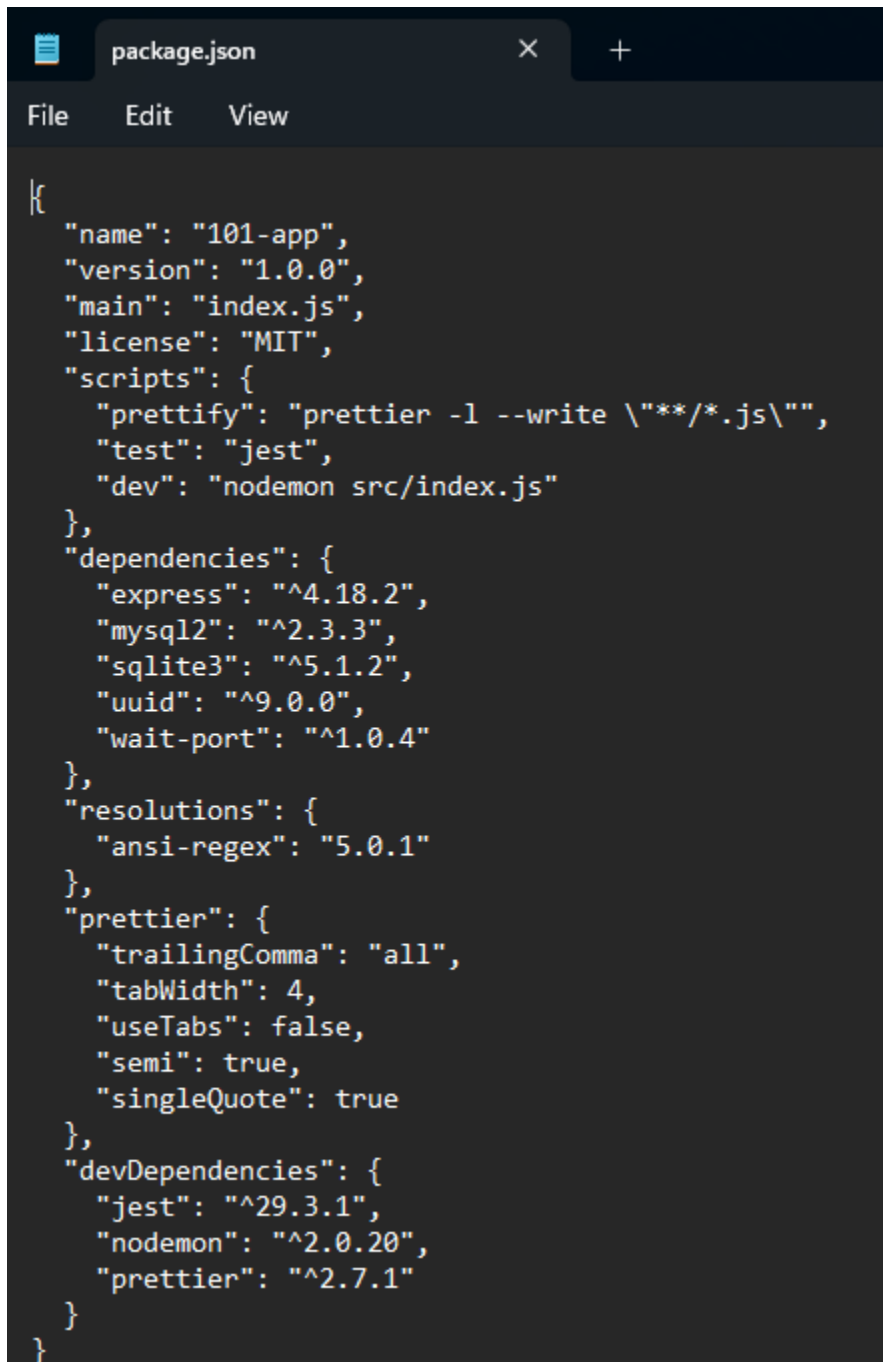
---

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
3. 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



## Lab Worksheet



```
{
  "name": "101-app",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "prettify": "prettier -l --write \"**/*.js\"",
    "test": "jest",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mysql2": "^2.3.3",
    "sqlite3": "^5.1.2",
    "uuid": "^9.0.0",
    "wait-port": "^1.0.4"
  },
  "resolutions": {
    "ansi-regex": "5.0.1"
  },
  "prettier": {
    "trailingComma": "all",
    "tabWidth": 4,
    "useTabs": false,
    "semi": true,
    "singleQuote": true
  },
  "devDependencies": {
    "jest": "^29.3.1",
    "nodemon": "^2.0.20",
    "prettier": "^2.7.1"
  }
}
```

4. ภายใต้ getting-started/app ให้สร้าง Dockerfile พร้อมกับใส่เนื้อหาดังต่อไปนี้ลงไปไฟล์  
FROM node:18-alpine  
WORKDIR /app  
COPY . .

## Lab Worksheet

RUN yarn install --production

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

EXPOSE 3000

5. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้ โดยกำหนดใช้ชื่อ image เป็น myapp\_รหัสสนศ. ไม่มีขีด

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

[Check point#8] Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง)

แสดงคำสั่งและผลลัพธ์ที่ได้ทางหน้าจอ

```
=> => transferring dockerfile: 169B 0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine 4.9s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [internal] load_dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load build context 0.5s
=> => transferring context: 4.62MB 0.5s
=> [1/4] FROM docker.io/library/node:18-alpine@sha256:e0348f26173b41066d68a3fe9bfbdb6571ab3cad0a4272919a52e36f4ae56925 21.4s
=> resolve docker.io/library/node:18-alpine@sha256:e0348f26173b41066d68a3fe9bfbdb6571ab3cad0a4272919a52e36f4ae56925 0.0s
=> sha256:a5cf158cb3746a8af43a656f15a671bb84fa704ac2413cfe57cc37d12afaf38 40.01MB / 40.01MB 19.4s
=> sha256:f46e519824fbb18a1df4becc0a40665f5c1cd193e527b3b0e26683d10f140916 1.26MB / 1.26MB 2.2s
=> sha256:e0348f26173b41066d68a3fe9bfbdb6571ab3cad0a4272919a52e36f4ae56925 7.67kB / 7.67kB 0.0s
=> sha256:b33d7471a6a5106ccdb3b6e4368841e06338ff6e5e8b2ff345e2e17f15902d7d 1.72kB / 1.72kB 0.0s
=> sha256:79649fe1a0d792c12299abc78e1bc691884f8a7b519a294c351caa0748531b7c 6.18kB / 6.18kB 0.0s
=> sha256:f18232174bc91741fd3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB 1.9s
=> extracting sha256:f18232174bc91741fd3da96d85011092101a032a93a388b79e99e69c2d5c870 0.2s
=> sha256:3d8c59f7380dbad7c144c50d7f2b78b0bb5ee2f158099c4c17aa8bdf56d74db0 444B / 444B 2.3s
=> extracting sha256:a5cf158cb3746a8af43a656f15a671bb84fa704ac2413cfe57cc37d12afaf38 1.7s
=> extracting sha256:f46e519824fbb18a1df4becc0a40665f5c1cd193e527b3b0e26683d10f140916 0.0s
=> extracting sha256:3d8c59f7380dbad7c144c50d7f2b78b0bb5ee2f158099c4c17aa8bdf56d74db0 0.0s
=> [2/4] WORKDIR /app 1.6s
=> [3/4] COPY . 0.1s
=> [4/4] RUN yarn install --production 50.0s
=> exporting to image 2.3s
=> exporting layers 2.0s
=> writing image sha256:0f31db4da3dc5ad8b180c26ab7272ee4e7d5c7d88804be645de3d6a9db9d298f 0.0s
=> naming to docker.io/library/myapp_6533883254 0.1s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/upp0jp1v1ae5dxrxalzfcqs4
PS C:\Users\DRAGON\Lab8_4\getting-started\app>
```

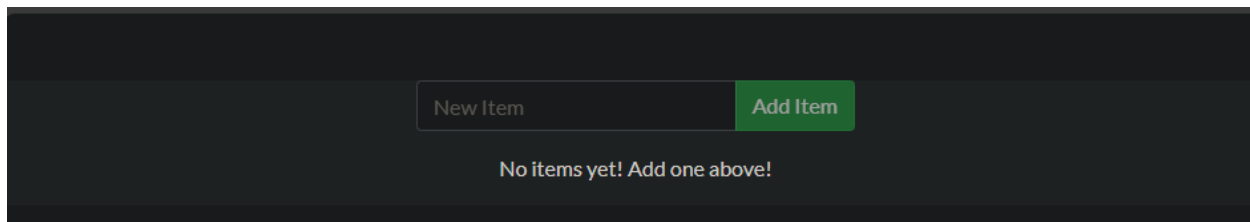
6. ทำการ Start ตัว Container ของแอปพลิเคชันที่สร้างขึ้น โดยใช้คำสั่ง

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

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

[Check point#9] Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงผลลัพธ์ที่ได้บน Browser

และ Dashboard ของ Docker desktop



## Lab Worksheet

หมายเหตุ: นศ.สามารถทดลองเล่น Web application ที่ทำงานอยู่ได้

8. ทำการแก้ไข Source code ของ Web application ดังนี้

a. 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>

b. Save ไฟล์ให้เรียบร้อย

9. ทำการ Build Docker image โดยใช้คำสั่งเดียวกันกับข้อ 5

10. Start และรัน Container ตัวใหม่ โดยใช้คำสั่งเดียวกันกับข้อ 6

[Check point#10] Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง)

แสดงคำสั่งและผลลัพธ์ที่ได้ทางหน้าจอ พร้อมกับตอบคำถามต่อไปนี้

```
PS C:\Users\DRAGON\Lab8_4\getting-started\app> docker run -dp 3000:3000 myapp_6533803254
ecc7c9f562a9f16e791a169534e27f17e55067d82af512daca27967bc4e645d1
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint hopeful_mclaren (0a03d1895014b17fe1ca7b05cf1aa9ba2a021b5cd7127240695
034d9e430b0e5): Bind for 0.0.0.0:3000 failed: port is already allocated

Run 'docker run --help' for more information
PS C:\Users\DRAGON\Lab8_4\getting-started\app> |
```

(1) Error ที่เกิดขึ้นหมายความว่าอย่างไร และเกิดขึ้นเพราะอะไร

- ข้อผิดพลาดนี้หมายความว่า Docker ไม่สามารถผูก (bind) พอร์ต 3000 บนโฮสต์ (host) กับคอนเทนเนอร์ได้ เนื่องจากพอร์ต 3000 บนโฮสต์ถูกใช้งานอยู่แล้วโดยคอนเทนเนอร์หรือโปรเซสอื่นสาเหตุที่เกิดขึ้น

- มีคอนเทนเนอร์อื่นที่กำลังใช้งานพอร์ต 3000 อยู่แล้ว
- มีโปรเซสอื่นบนเครื่องโฮสต์ที่กำลังฟัง (listening) บนพอร์ต 3000

11. ลบ Container ของ Web application เวอร์ชันก่อนแก้ไขออกจากระบบ โดยใช้วิธีใดวิธีหนึ่งดังต่อไปนี้

a. ผ่าน Command line interface

- ใช้คำสั่ง \$ docker ps เพื่อดู Container ID ที่ต้องการจะลบ
- Copy หรือบันทึก Container ID ไว้

## Lab Worksheet

iii. ใช้คำสั่ง \$ docker stop <Container ID ที่ต้องการจะลบ> เพื่อหยุดการทำงานของ Container ดังกล่าว

iv. ใช้คำสั่ง \$ docker rm <Container ID ที่ต้องการจะลบ> เพื่อทำการลบ

b. ผ่าน Docker desktop

i. ไปที่หน้าต่าง Containers

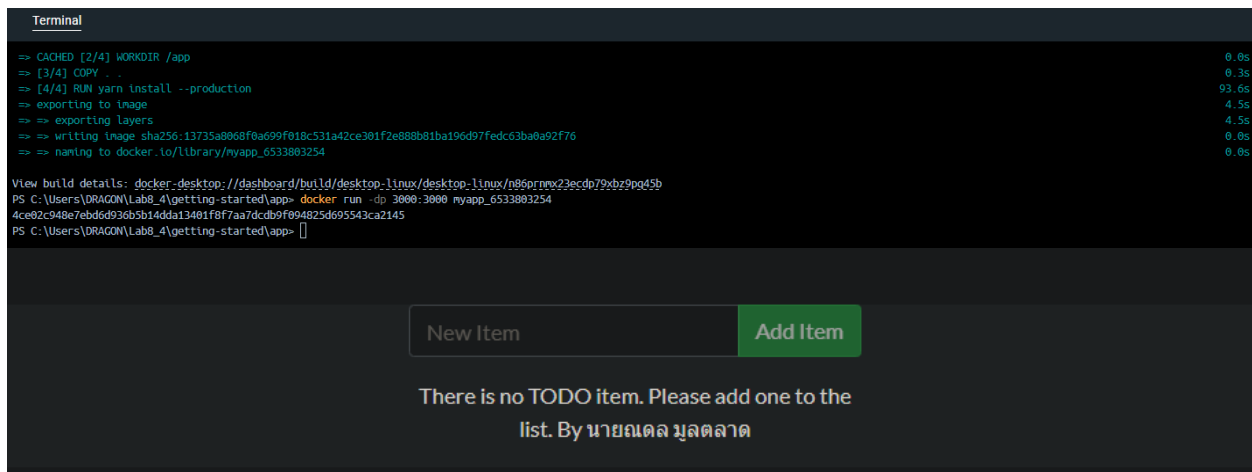
ii. เลือกไอคอนถังขยะในแถวของ Container ที่ต้องการจะลบ

iii. ยืนยันโดยการกด Delete forever

12. Start และรัน Container ตัวใหม่อีกครั้ง โดยใช้คำสั่งเดียวกันกับข้อ 6

13. 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:its-jdk17

หรือ

\$ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure -v

jenkins\_home:/var/jenkins\_home jenkins/jenkins:its-jdk17

## Lab Worksheet

3. บันทึกรหัสผ่านของ Admin user ไว้สำหรับ log-in ในครั้งแรก

[Check point#12] Capture หน้าจอที่แสดงผล Admin password

```
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

e1ba1f9917efa498d9e76cfe25732949f

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

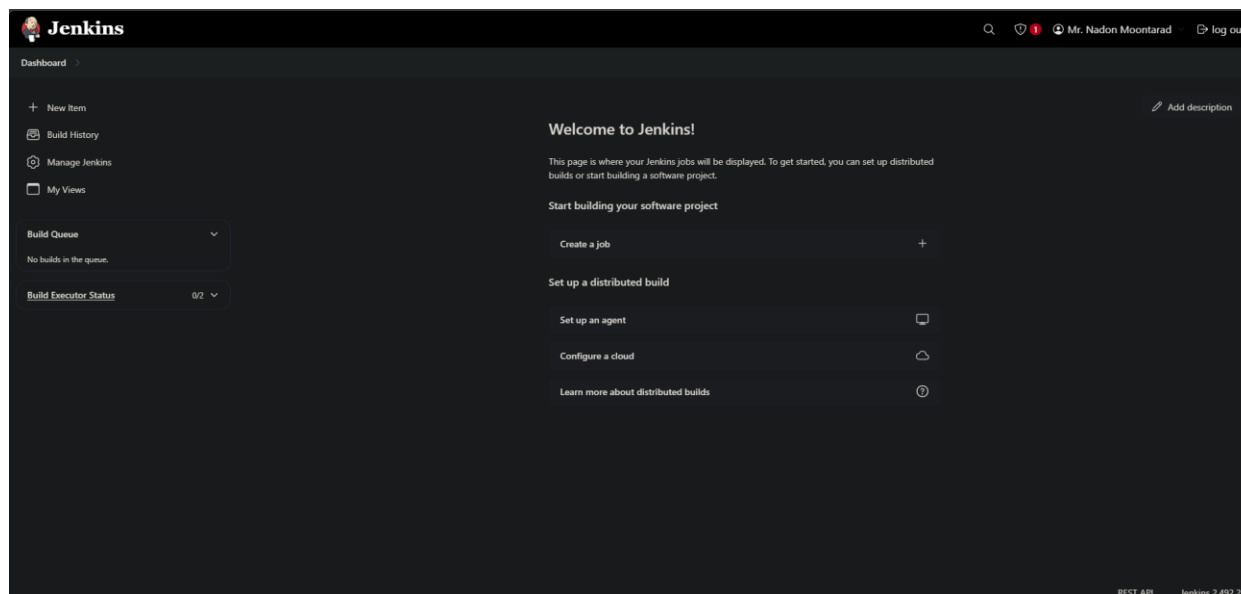
*****
```

4. เมื่อได้รับการยืนยันว่า Jenkins is fully up and running ให้เปิดบราวเซอร์ และป้อนที่อยู่เป็น  
localhost:8080

5. ทำการ Unlock Jenkins ด้วยรหัสผ่านที่ได้ในข้อที่ 3

6. สร้าง Admin User โดยใช้ username เป็นชื่อจริงของนักศึกษาพร้อมรหัสสี่ตัวท้าย เช่น somsri\_3062

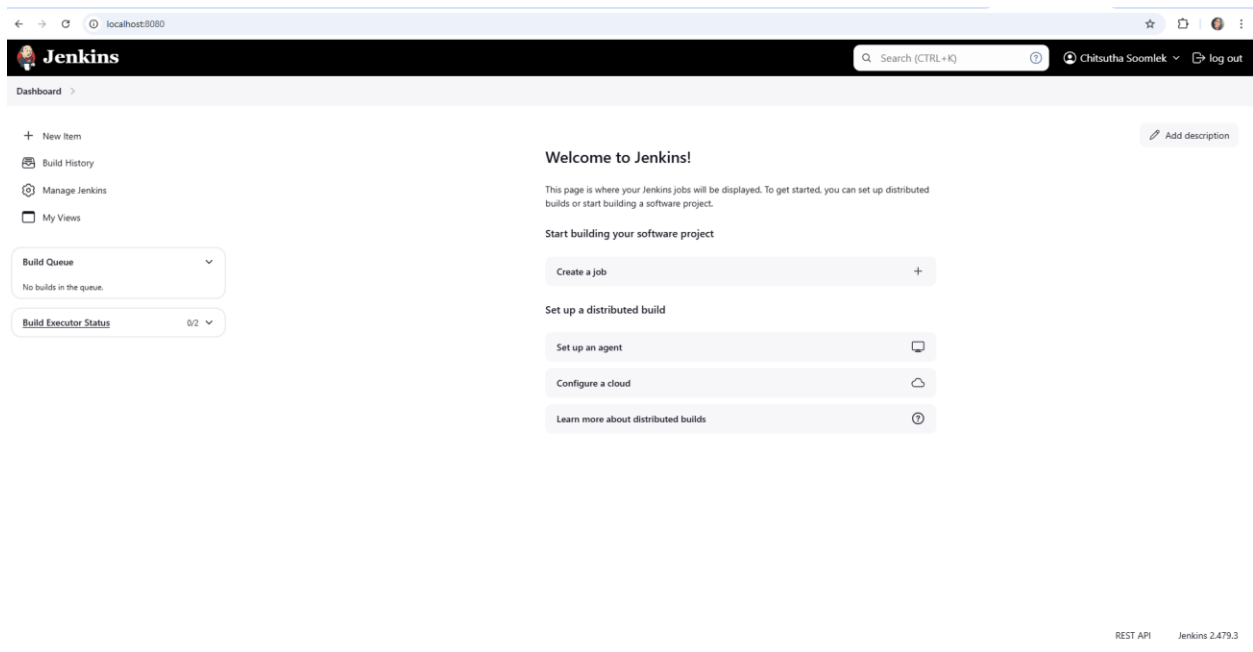
[Check point#13] Capture หน้าจอที่แสดงผลการตั้งค่า



7. กำหนด Jenkins URL เป็น <http://localhost:8080/lab8>

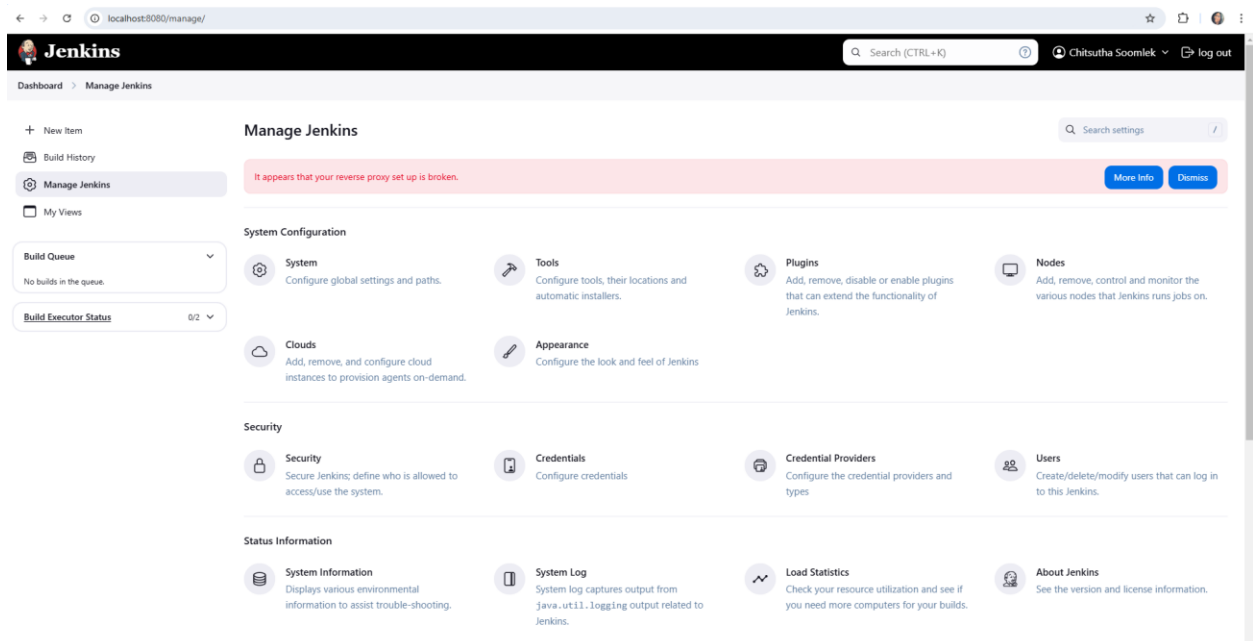
8. เมื่อติดตั้งเรียบร้อยแล้วจะพบกันหน้า Dashboard ดังแสดงในภาพ

## Lab Worksheet



The screenshot shows the Jenkins Dashboard at localhost:8080. The top navigation bar includes the Jenkins logo, a search bar, and a user profile for Chitsutha Soomlek with a log out button. The left sidebar contains links for New Item, Build History, Manage Jenkins, and My Views. The main content area features a 'Welcome to Jenkins!' message, a 'Start building your software project' section with a 'Create a job' button, and a 'Set up a distributed build' section with buttons for 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. The bottom right corner displays 'REST API' and 'Jenkins 2.479.3'.

## 9. เลือก Manage Jenkins แล้วไปที่เมนู Plugins



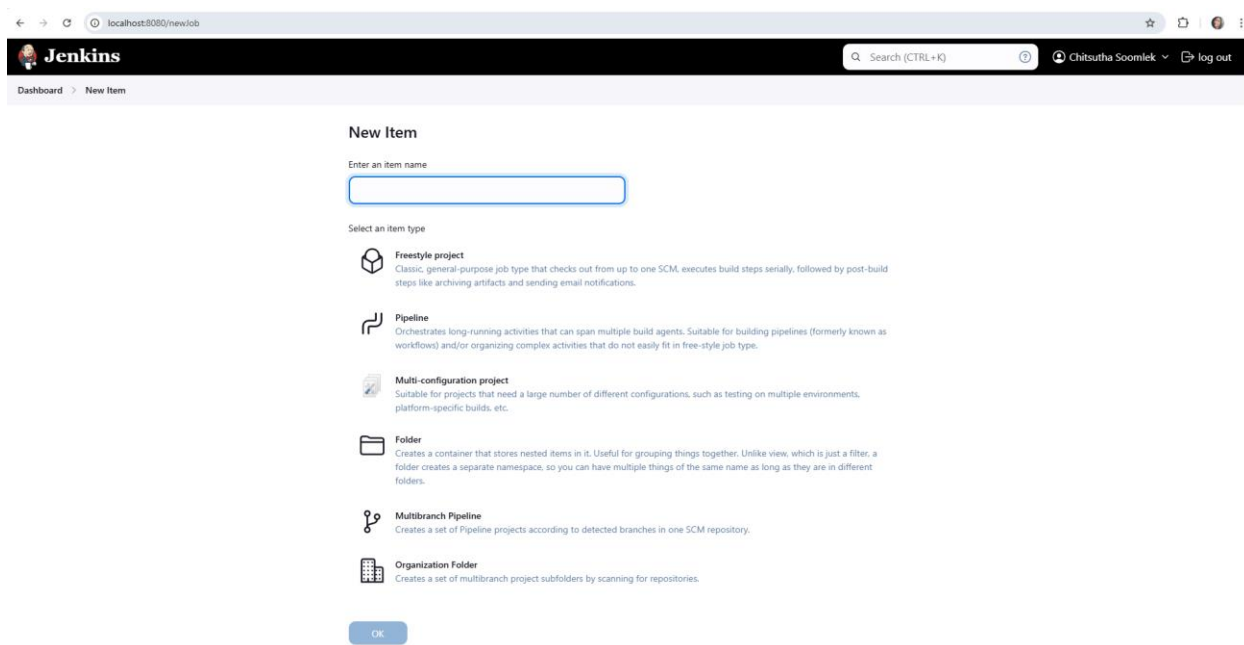
The screenshot shows the 'Manage Jenkins' page at localhost:8080/manage/. The top navigation bar is similar to the dashboard. The left sidebar highlights 'Manage Jenkins'. The main content area has a 'Manage Jenkins' title and a search settings bar. A red warning banner states 'It appears that your reverse proxy set up is broken.' with 'More Info' and 'Dismiss' buttons. Below this, the 'System Configuration' section includes links for System, Tools, Plugins, Nodes, Clouds, and Appearance. The 'Security' section includes links for Security, Credentials, Credential Providers, and Users. The 'Status Information' section includes links for System Information, System Log, Load Statistics, and About Jenkins.

## Lab Worksheet

10. ไปที่เมนู Available plugins แล้วเลือกติดตั้ง Robotframework เพิ่มเติม



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



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

Description: Lab 8.5

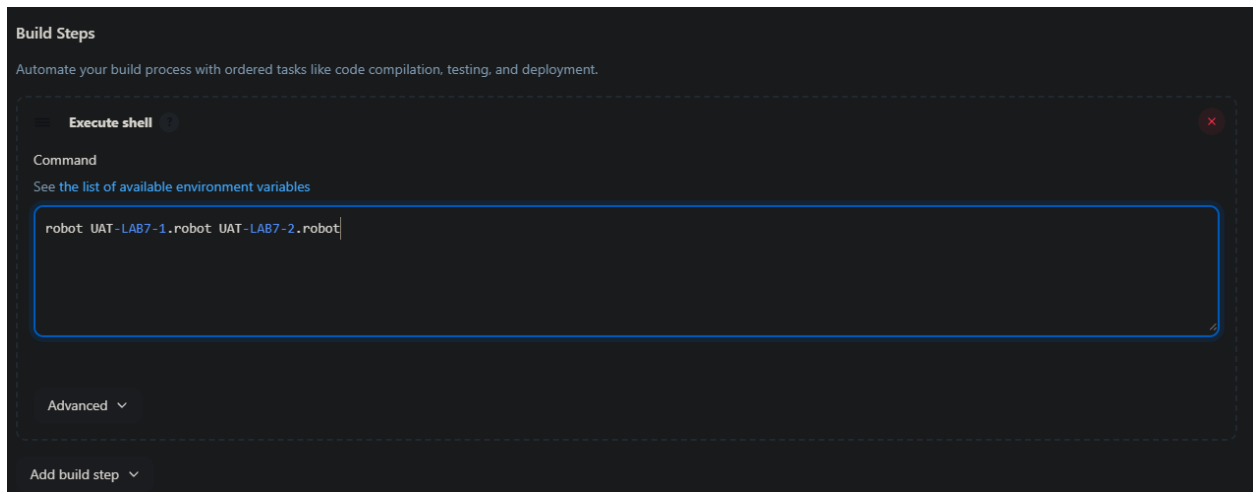
GitHub project: กดเลือก แล้วใส่ Project URL เป็น repository ที่เก็บโค้ด .robot (ดูขั้นตอนที่ 12)

Build Trigger: เลือกแบบ Build periodically แล้วกำหนดให้ build ทุก 15 นาที

## Lab Worksheet

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

[Check point#14] Capture หน้าจอแสดงการตั้งค่า พร้อมกับตอบคำถามต่อไปนี้



(1) คำสั่งที่ใช้ในการ Execute ไฟล์ .robot ใน Build Steps คือ

robot UAT-LAB7-1.robot UAT-LAB7-2.robot

**Post-build action:** เพิ่ม Publish Robot Framework test results ->

ระบุไดเรกทอรีที่เก็บไฟล์ผลการทดสอบโดย Robot framework ในรูป xml และ html -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ไม่ผ่านแล้วนับว่าซอฟต์แวร์มีปัญหา -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ผ่านแล้วนับว่าซอฟต์แวร์มีอยู่ในสถานะที่สามารถนำไปใช้งานได้ (เช่น 20, 80)

13. กด Apply และ Save

14. สั่ง Build Now

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



## Lab Worksheet

**Jenkins** Dashboard > UAT

Status **UAT** Lab 8.5

Latest Robot Results: No results available yet.

Permalinks

- Last build (#16), 3 min 11 sec ago
- Last failed build (#16), 3 min 11 sec ago
- Last unsuccessful build (#16), 3 min 11 sec ago
- Last completed build (#16), 3 min 11 sec ago

**Builds**

Filter

Today

- #16 5:08 PM
- #15 5:08 PM
- #14 5:05 PM
- #13 4:50 PM

Dashboard > UAT > Build Time Trend

**Build Time Trend**

S	Build	Time Since	Duration
⊗	#16	4 min 23 sec	14 ms
⊗	#15	5 min 20 sec	1.1 sec
⊗	#14	9 min 12 sec	1.7 sec
⊗	#13	24 min	1.3 sec

Icon S M L

**Builds**

Filter

Today

- #16 5:09 PM
- #15 5:08 PM
- #14 5:05 PM
- #13 4:50 PM

Dashboard > UAT > #16 > Console Output

Status **Console Output** Download Copy View as plain text

Started by user Mr. Nadon Moontarad

Running as SYSTEM

Building in workspace /var/jenkins\_home/workspace/UAT

```
[UAT] $ /bin/sh -xe /tmp/jenkins15865605144330833732.sh
+ robot UAT-LAB7-1.robot UAT-LAB7-2.robot
/tmp/jenkins15865605144330833732.sh: 21: robot: not found
Build step 'Execute shell' marked build as failure
Robot results publisher started...
INFO: Checking test criticality is deprecated and will be dropped in a future release!
-Parsing output xml:
Failed!
hudson.AbortException: No files found in path /var/jenkins_home/workspace/UAT with configured filemask: output.xml
    at PluginClassLoader for robot/hudson.plugins.robot.RobotParser$RobotParserCallable.invoke(RobotParser.java:81)
    at PluginClassLoader for robot/hudson.plugins.robot.RobotParser$RobotParserCallable.invoke(RobotParser.java:52)
    at hudson.FilePath.act(FilePath.java:1213)
    at hudson.FilePath.act(FilePath.java:1196)
    at PluginClassLoader for robot/hudson.plugins.robot.RobotParser.parse(RobotParser.java:48)
    at PluginClassLoader for robot/hudson.plugins.robot.RobotPublisher.parse(RobotPublisher.java:262)
    at PluginClassLoader for robot/hudson.plugins.robot.RobotPublisher.perform(RobotPublisher.java:286)
    at hudson.tasks.BuildStepCompatibilityLayer.perform(BuildStepCompatibilityLayer.java:80)
    at hudson.tasks.BuildStepMonitor$1.perform(BuildStepMonitor.java:20)
    at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:818)
    at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:767)
    at hudson.model.Build$BuildExecution.post2(Build.java:179)
    at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:711)
    at hudson.model.Run.execute(Run.java:1856)
    at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:44)
    at hudson.model.ResourceController.execute(ResourceController.java:101)
    at hudson.model.Executor.run(Executor.java:446)
Finished: FAILURE
```