

**Lab Worksheet**

ชื่อ-นามสกุล \_\_\_\_\_ รหัสนักศึกษา \_\_\_\_\_ Section \_\_\_\_\_

**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**

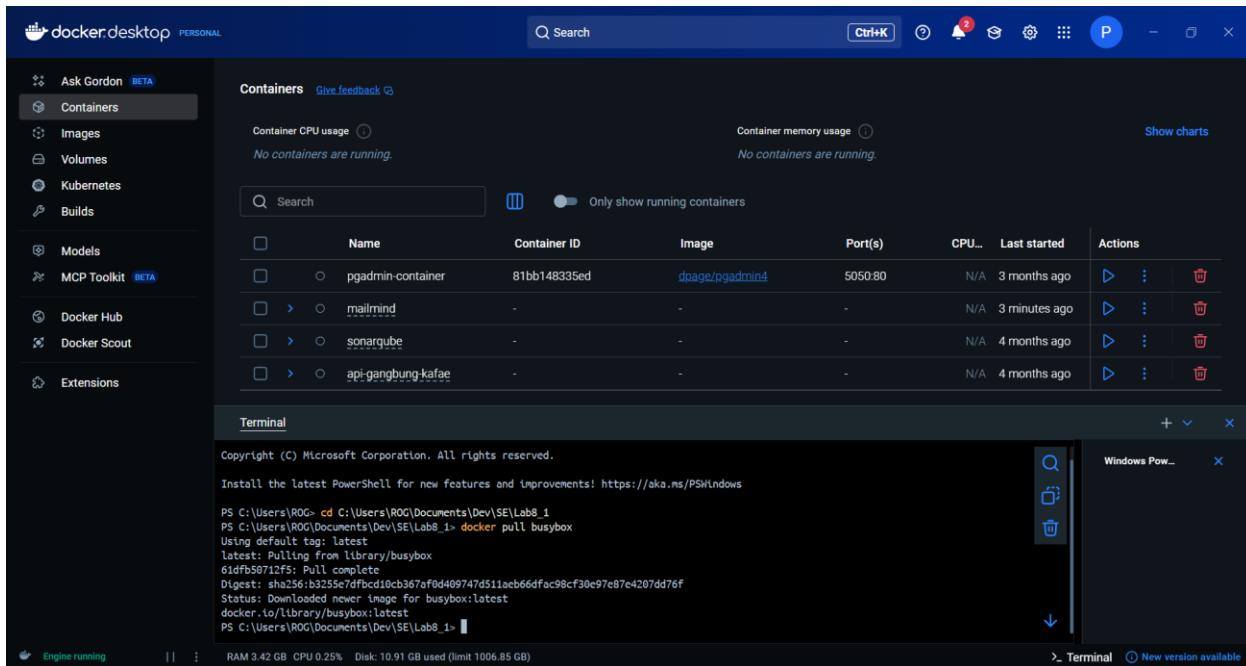
1. ติดตั้ง Docker desktop ลงบนเครื่องคอมพิวเตอร์ โดยดาวน์โหลดจาก <https://www.docker.com/get-started>
2. สร้าง Account บน Docker hub (<https://hub.docker.com/signup>)
3. กำหนดให้ \$ หมายถึง Command prompt และ <> หมายถึง ให้ป้อนค่าของพารามิเตอร์ที่กำหนด

**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

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

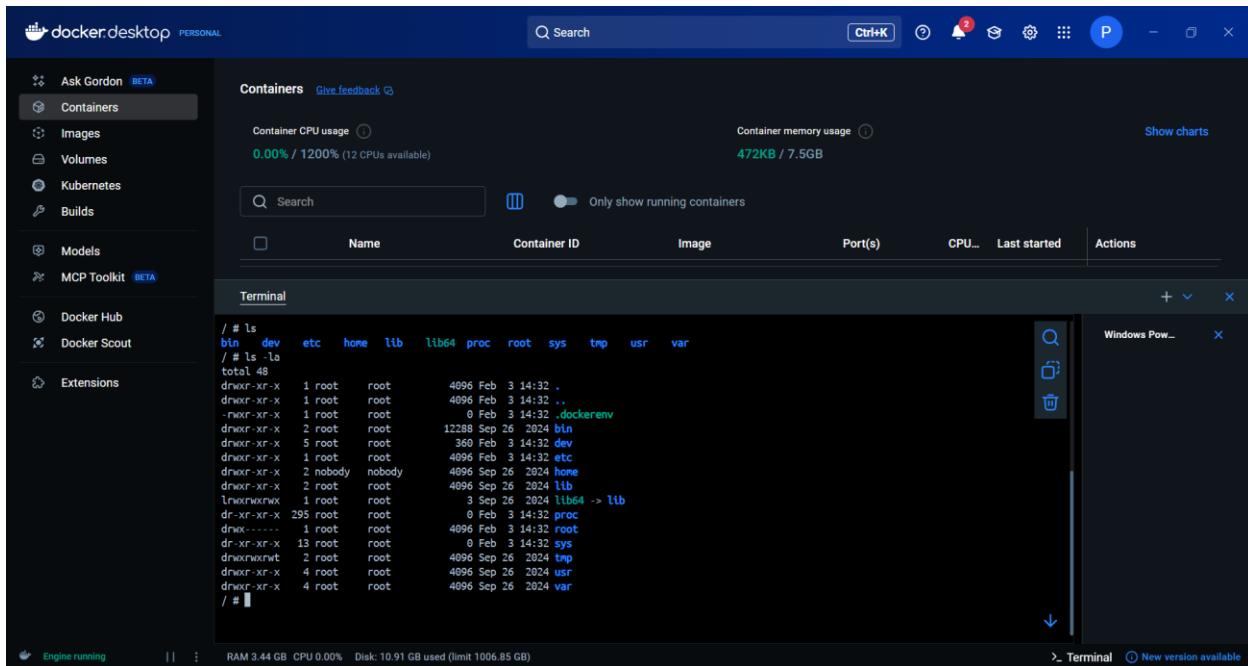
## Lab Worksheet



- (1) สิ่งที่อยู่ภายใต้คอลัมน์ Repository คืออะไร คือชื่อ Image
- (2) Tag ที่ใช้บ่งบอกถึงอะไร latest คือ โหลดเวอร์ชันล่าสุด

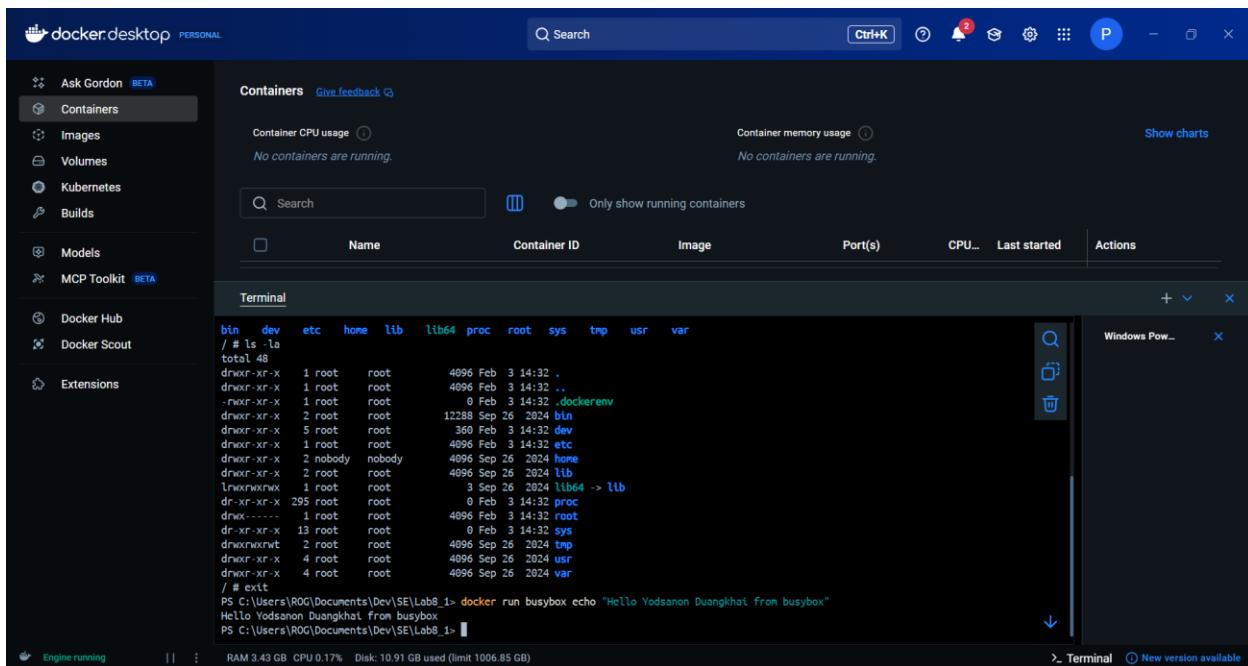
5. ป้อนคำสั่ง \$ docker run busybox
6. ป้อนคำสั่ง \$ docker run -it busybox sh
7. ป้อนคำสั่ง ls
8. ป้อนคำสั่ง ls -la
9. ป้อนคำสั่ง exit
10. ป้อนคำสั่ง \$ docker run busybox echo "Hello ชื่อและนามสกุลของนักศึกษา from busybox"
11. ป้อนคำสั่ง \$ docker ps -a

## Lab Worksheet



```
/ # ls
bin dev etc home lib lib64 proc root sys tmp usr var
/ # ls -la
total 48
drwxr-xr-x 1 root root 4096 Feb 3 14:32 .
drwxr-xr-x 1 root root 4096 Feb 3 14:32 ..
-rw-r--r-- 1 root root 0 Feb 3 14:32 .dockerenv
drwxr-xr-x 2 root root 12288 Sep 26 2024 bin
drwxr-xr-x 5 root root 360 Feb 3 14:32 dev
drwxr-xr-x 1 root root 4096 Feb 3 14:32 etc
drwxr-xr-x 2 nobody nobody 4096 Sep 26 2024 home
drwxr-xr-x 2 root root 4096 Sep 26 2024 lib
lwxrwxrwx 1 root root 3 Sep 26 2024 lib64 -> lib
dr-xr-xr-x 295 root root 0 Feb 3 14:32 proc
drwxr-xr-x 1 root root 4096 Feb 3 14:32 root
dr-xr-xr-x 13 root root 0 Feb 3 14:32 sys
drwxrwxrwt 2 root root 4096 Sep 26 2024 tmp
drwxr-xr-x 4 root root 4096 Sep 26 2024 usr
drwxr-xr-x 4 root root 4096 Sep 26 2024 var
/ #
```

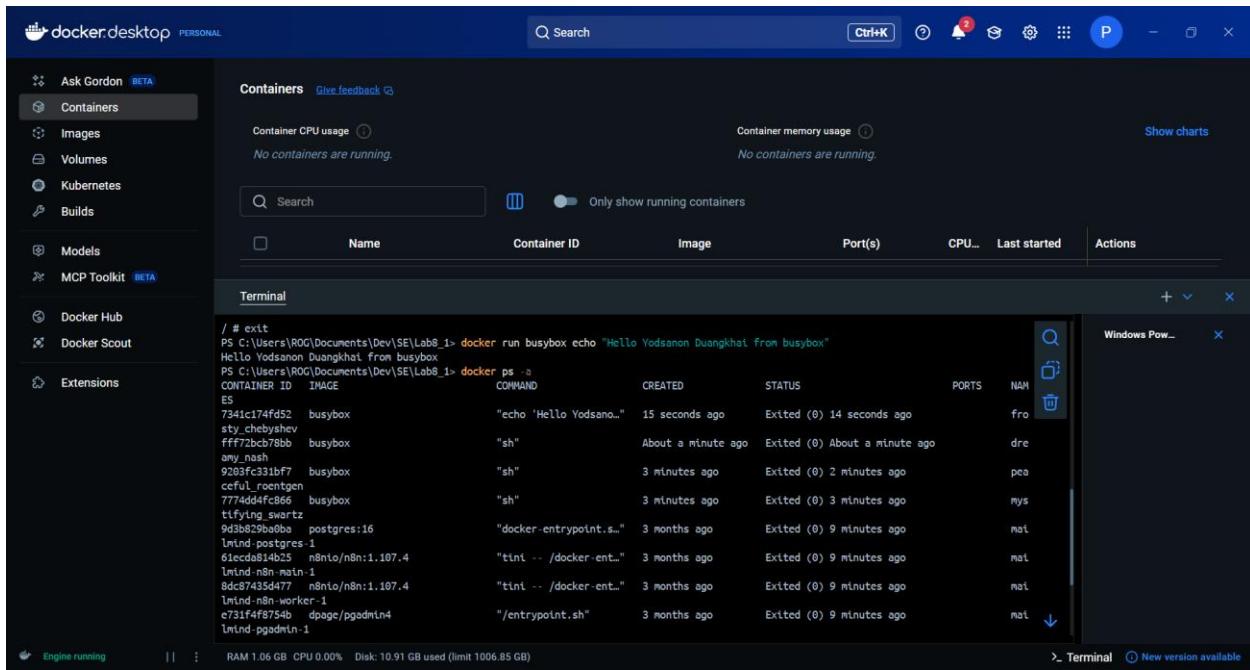
Docker Desktop interface showing the terminal window. The sidebar on the left has 'Containers' selected. The terminal shows the output of 'ls' and 'ls -la' commands. The status bar at the bottom indicates 'Engine running', 'RAM 3.44 GB', 'CPU 0.00%', and 'Disk: 10.91 GB used (limit 1006.85 GB)'.



```
/ # ls -la
total 48
drwxr-xr-x 1 root root 4096 Feb 3 14:32 .
drwxr-xr-x 1 root root 4096 Feb 3 14:32 ..
-rw-r--r-- 1 root root 0 Feb 3 14:32 .dockerenv
drwxr-xr-x 2 root root 12288 Sep 26 2024 bin
drwxr-xr-x 5 root root 360 Feb 3 14:32 dev
drwxr-xr-x 1 root root 4096 Feb 3 14:32 etc
drwxr-xr-x 2 nobody nobody 4096 Sep 26 2024 home
drwxr-xr-x 2 root root 4096 Sep 26 2024 lib
lwxrwxrwx 1 root root 3 Sep 26 2024 lib64 -> lib
dr-xr-xr-x 295 root root 0 Feb 3 14:32 proc
drwxr-xr-x 1 root root 4096 Feb 3 14:32 root
dr-xr-xr-x 13 root root 0 Feb 3 14:32 sys
drwxrwxrwt 2 root root 4096 Sep 26 2024 tmp
drwxr-xr-x 4 root root 4096 Sep 26 2024 usr
drwxr-xr-x 4 root root 4096 Sep 26 2024 var
/ # exit
PS C:\Users\ROG\Documents\Dev\SE\Lab8> docker run busybox echo "Hello Yodanon Duangkhai from busybox"
Hello Yodanon Duangkhai from busybox
PS C:\Users\ROG\Documents\Dev\SE\Lab8>
```

Docker Desktop interface showing the terminal window. The sidebar on the left has 'Containers' selected. The terminal shows the output of a 'docker run busybox' command. The status bar at the bottom indicates 'Engine running', 'RAM 3.43 GB', 'CPU 0.17%', and 'Disk: 10.91 GB used (limit 1006.85 GB)'.

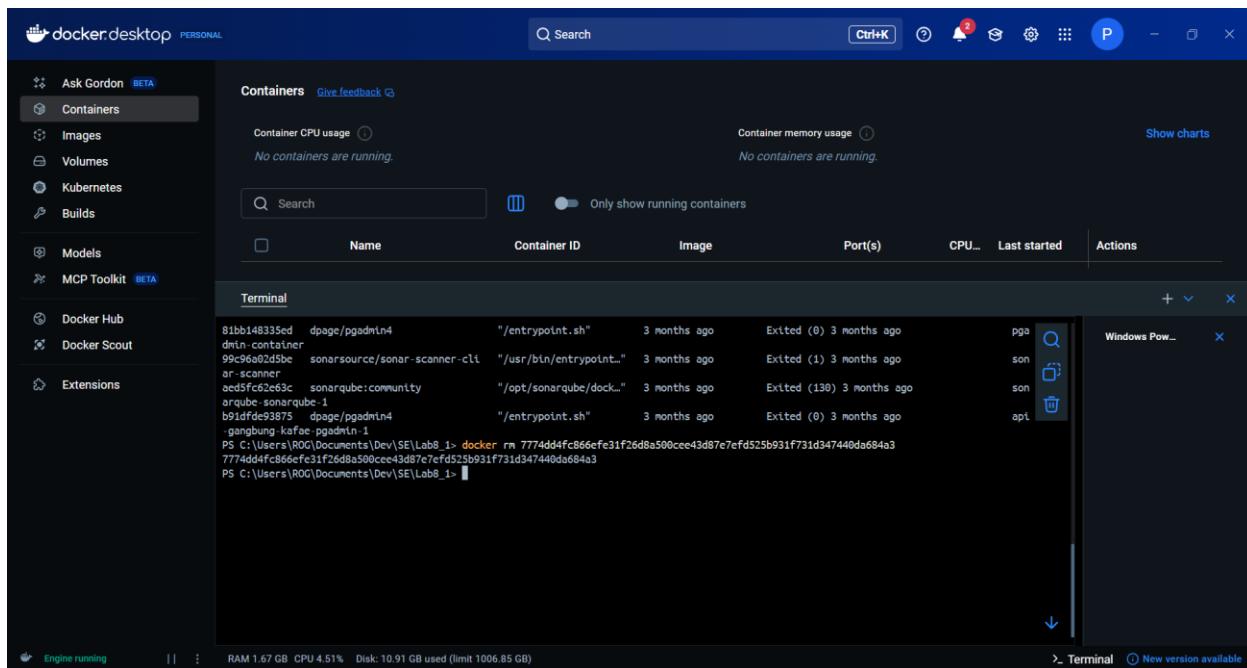
## Lab Worksheet



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

- (1) เมื่อใช้ option -it ในคำสั่ง run ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสั้นๆ เช่น สามารถเข้าไปใช้งาน shell ภายใน container ได้
- (2) คอลัมน์ STATUS จากการรันคำสั่ง docker ps -a แสดงถึงข้อมูลอะไรของ container หยุดหรือเริ่มทำงานไปแล้วนานเท่าใด
12. ป้อนคำสั่ง \$ docker rm <container ID ที่ต้องการลบ>

## Lab Worksheet



[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 เօไว
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

สำหรับเครื่องที่ใช้ระบบปฏิบัติการวินโดว์ส (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

## Lab Worksheet

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

CMD echo “ชื่อ-นามสกุล รหัสนักศึกษา ชื่อเล่น”

EOF

หรือใช้คำสั่ง

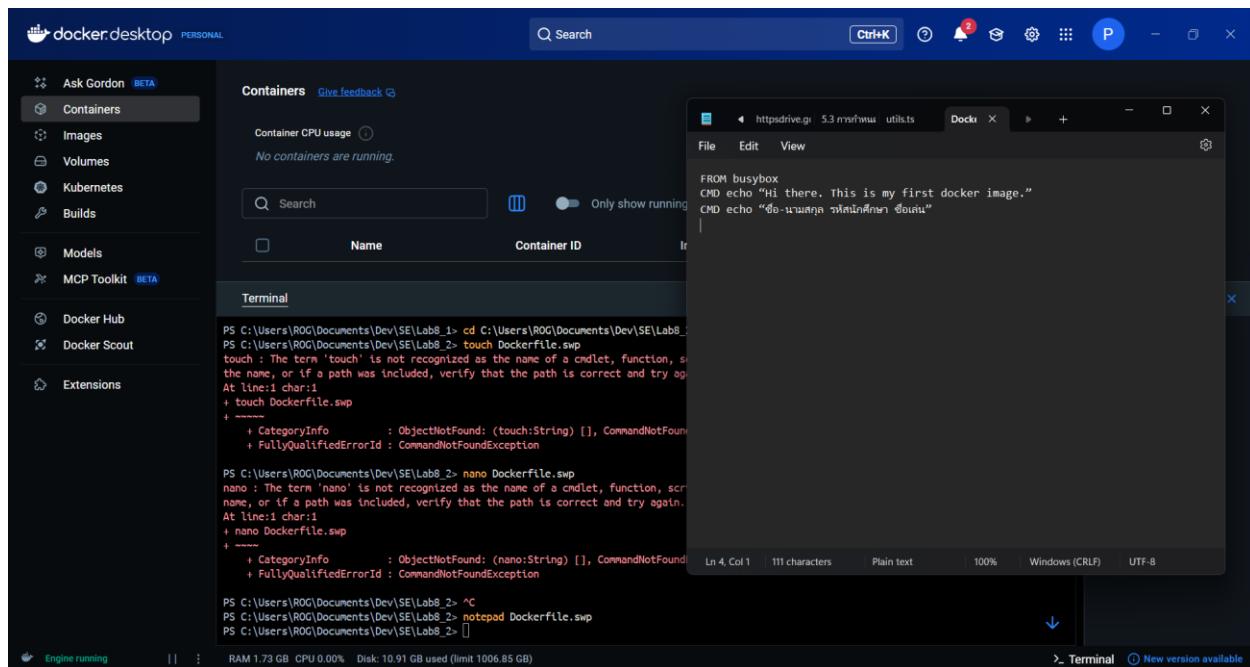
\$ touch Dockerfile

แล้วใช้ Text Editor ในการใส่เนื้อหาแทน

5. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

\$ docker build -t <ชื่อ Image> .

6. เมื่อ Build สำเร็จแล้ว ให้ทำการรัน Docker image ที่สร้างขึ้นในขั้นตอนที่ 5



## Lab Worksheet

The screenshot shows the Docker Desktop interface. On the left, the sidebar includes options like Ask Gordon, Containers (selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' with a search bar and a chart showing 'Container CPU usage' and 'Container memory usage'. A table lists running containers, with a single row for '150 Dockerfile.swp'. Below this is a 'Terminal' window showing the command:

```
PS C:\Users\ROG\Documents\Dev\SE\Lab8_2> docker build -t lab8-image .
[+] Building 0.5s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> [internal] transfer Dockerfile: 109B
=> [internal] load metadata for docker.io/library/busybox:latest
=> [internal] load .dockerignore
=> [internal] transfer context: 2B
=> [1/1] FROM docker.io/library/busybox:latest@sha256:b3255c7dfbcd10cb367af0d409747d511eb66dfac98cf30e97c87e4207dd76f
=> => resolve docker.io/library/busybox:latest@sha256:b3255c7dfbcd10cb367af0d409747d511eb66dfac98cf30e97c87e4207dd76f
=> => exporting manifest sha256:8d943e52102e4b86e2914f7cd465ccf79f43130d366e48cd6d363ad501758f47
=> => naming to docker.io/library/lab8-image:latest
=> => unpacking to docker.io/library/lab8-image:latest
```

On the right, a file browser window titled 'Windows Pow...' is open. The status bar at the bottom indicates: Engine running, RAM 1.74 GB, CPU 0.08%, Disk: 10.91 GB used (limit 1006.85 GB).

This screenshot is similar to the first one, showing the Docker Desktop interface and a terminal window. The terminal output shows a Docker build command and a warning message:

```
PS C:\Users\ROG\Documents\Dev\SE\Lab8_2> docker build -t lab8-image .
[+] Building 0.1s (5/5) FINISHED
=> [internal] load metadata for docker.io/library/busybox:latest
=> [internal] transfer Dockerfile: 109B
=> [internal] load .dockerignore
=> [internal] transfer context: 2B
=> [1/1] FROM docker.io/library/busybox:latest@sha256:b3255c7dfbcd10cb367af0d409747d511eb66dfac98cf30e97c87e4207dd76f
=> => resolve docker.io/library/busybox:latest@sha256:b3255c7dfbcd10cb367af0d409747d511eb66dfac98cf30e97c87e4207dd76f
=> => exporting manifest sha256:8d943e52102e4b86e2914f7cd465ccf79f43130d366e48cd6d363ad501758f47
=> => naming to docker.io/library/lab8-image:latest
=> => unpacking to docker.io/library/lab8-image:latest
```

Below the terminal, a message states: '3 warnings found (use docker --debug to expand):'. The warnings listed are:

- 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)

The status bar at the bottom indicates: Engine running, RAM 1.74 GB, CPU 0.00%, Disk: 10.91 GB used (limit 1006.85 GB).

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

(1) คำสั่งที่ใช้ในการ run คือ docker run lab8-image

**Lab Worksheet**

- (2) Option -t ในคำสั่ง \$ docker build ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป  
ใช้กำหนดชื่อและแท็กของ Docker Image ที่สร้างขึ้น เพื่อให้เวลาเรียกใช้งานเรียนได้ง่ายขึ้น ไม่ต้องเรียกจาก  
Image ID

**แบบฝึกปฏิบัติที่ 8.3: การแชร์ Docker image ผ่าน Docker Hub**

1. เปิดใช้งาน Docker desktop และ Login ด้วย Username และ Password ที่ลงทะเบียนกับ Docker Hub เก่าไว้
  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
- สำหรับเครื่องที่ใช้ระบบปฏิบัติการวินโดว์ส 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

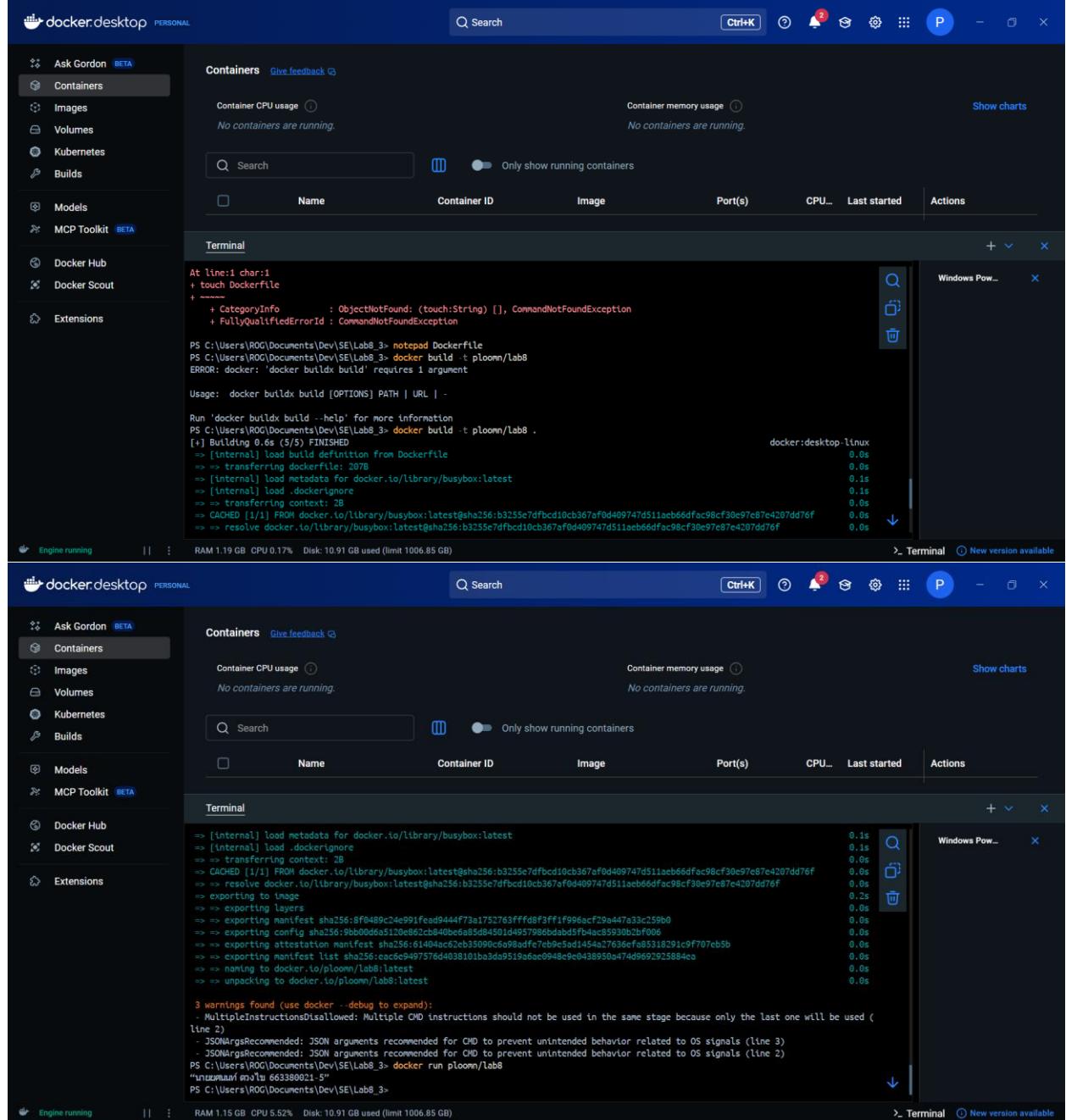
```
$ touch Dockerfile
แล้วใช้ Text Editor ในการใส่เนื้อหาแทน
```

7. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้
 

```
$ docker build -t <username> /lab8
```
5. ทำการรัน Docker image บน Container ในเครื่องของตัวเองเพื่อทดสอบผลลัพธ์ ด้วยคำสั่ง

## Lab Worksheet

§ docker run <username ที่ลงทะเบียนกับ Docker Hub>/lab8



[Check point#5] Capture หน้าจอ (ทั้งหน้าต่างและทกหน้าต่างที่เกี่ยวข้อง) และแสดงผลลัพธ์ที่ได้ในขันตอนที่ 5

6. ทำการ Push ตัว Docker image ไปไว้บน Docker Hub โดยการใช้คำสั่ง

§ docker push <username ที่ลงทะเบียนกับ Docker Hub>/lab8

ในการนี้ที่ติดปัญหาไม่ได้ Login ไว้ก่อน ให้ใช้คำสั่งต่อไปนี้ เพื่อ Login ก่อนทำการ Push

## Lab Worksheet

\$ 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

The screenshot displays two instances of the Docker Desktop application running simultaneously. Both windows have a dark-themed header bar with the Docker logo, the text "docker desktop PERSONAL", and various system icons.

**Left Window (Top):**

- Sidebar:** Contains links for Ask Gordon, Containers (BETA), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions.
- Containers Tab:** Shows a search bar, a "Container CPU usage" section with the message "No containers are running.", a "Container memory usage" section with the message "No containers are running.", and a "Show charts" link.
- Terminal Tab:** Displays a terminal session with the following output:

```
>>> exporting layers
>>> exporting manifest sha256:8f0489c24e991fecd044ff73a753763ffdf3ff1f996acf209a47a23c35fb0
>>> exporting config sha256:9bb0886a510e861cb840be6a85d84501d4957986bdb5fb4ac85930b2fb006
>>> exporting configuration manifest sha256:61404ac62b3590cd98ad7fc9e5ad1454a27636cfab8318291c9f707eb5b
>>> exporting manifest list sha256:ccde9497578d4038101ba3da9519a0ae0948e9c0438950a474d9692925884ca
>>> naming to docker.io/ploomn/lab8:latest
>>> unpacking to docker.io/ploomn/lab8:latest

3 warnings found (use docker --debug to expand):
- MultipleInstructionsNotAllowed: 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)
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 2)
PS C:\Users\ROG\Documents\Dev\SE\Lab8_3> docker run ploomn/lab8
"muzuuuuu"!@ms-663388021:5"
PS C:\Users\ROG\Documents\Dev\SE\Lab8_3> docker push ploomn/lab8
Using default tag: latest
The push refers to repository [docker.io/ploomn/lab8]
61dfb59712f5: Mounted from library/busybox
c1394cf9fcf9: Pushed
latest: digest: sha256:ccde9497576d4038101ba3da9519a0ae0948e9c0438950a474d9692925884ca size: 855
PS C:\Users\ROG\Documents\Dev\SE\Lab8_3>
```
- Bottom Status Bar:** Shows "Engine running", RAM 1.09 GB, CPU 0.17%, Disk 10.91 GB used (limit 1006.85 GB), and a "Terminal" link.

**Right Window (Bottom):**

- Sidebar:** Contains links for Ask Gordon, Containers (BETA), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions.
- Docker Hub Tab:** Shows a search bar, a "Docker Hub / ploomn/lab8" section with a user icon, the repository name "ploomn/lab8", a "Tag" dropdown set to "latest" (highlighted in blue), a "Pull" button, and a "Run" button.
- Overview Tab:** Shows a placeholder icon and the message "No repositories found". Below it, a note says "This repository doesn't have an overview."
- Recent Tags:** A table with a single row labeled "LATEST".
- Terminal Tab:** Displays a terminal session with the same command history as the left window.
- Bottom Status Bar:** Shows "Engine running", RAM 1.06 GB, CPU 0.00%, Disk 10.91 GB used (limit 1006.85 GB), and a "Terminal" link.

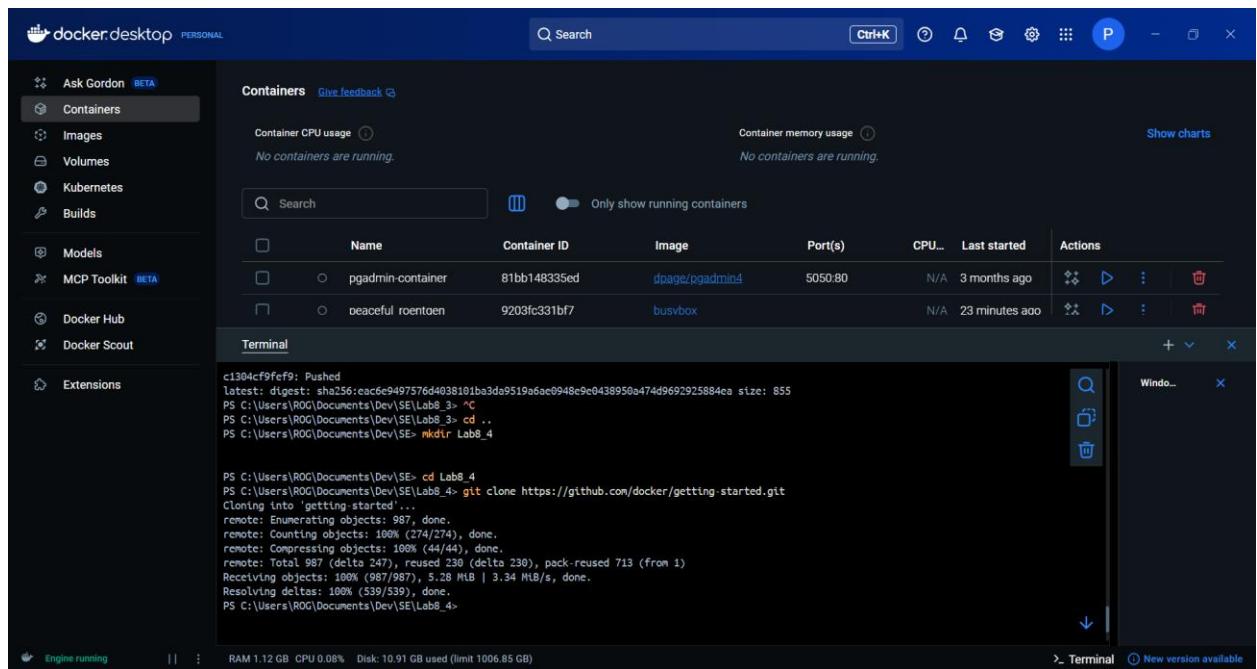
[Check point#6] Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) และส่ง Repository ที่มี Docker image (<username>/lab8)

## Lab Worksheet

## แบบฝึกปฏิบัติที่ 8.4: การ Build และ Update แอปพลิเคชันจาก Container image

1. เปิด Command line หรือ Terminal จากนั้นสร้าง Directory ชื่อ Lab8\_4
2. ทำการ Clone ซอฟต์แวร์ Docker จาก 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 เมื่อพับไฟล์ package.json ให้ใช้ Text editor ในการเปิดอ่าน



## 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"
}

```

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

4. ภายใต้ directory getting-started/app ให้สร้าง Dockerfile พร้อมกับใส่เนื้อหาดังต่อไปนี้ลงในไฟล์

FROM node:18-alpine

WORKDIR /app

COPY . .

RUN yarn install --production

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

EXPOSE 3000

5. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้ โดยกำหนดใช้ชื่อ image เป็น myapp\_รหัสนศ.

ศ. ไม่มีชื่อ

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

## Lab Worksheet

```

FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD [ "node", "src/index.js" ]
EXPOSE 3000
    
```

The screenshot shows the VS Code interface with the Dockerfile open in the editor. The file content is as follows:

```

FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD [ "node", "src/index.js" ]
EXPOSE 3000
    
```

The sidebar on the left shows the project structure with files like package.json, Dockerfile, and build.sh. A 'Build with Agent' panel is visible on the right.

[Check point#8] Capture หน้าจอ (ทั้งหน้าต่างและทุกหน้าต่างที่เกี่ยวข้อง) แสดงคำสั่งและผลลัพธ์ที่ได้ทาง

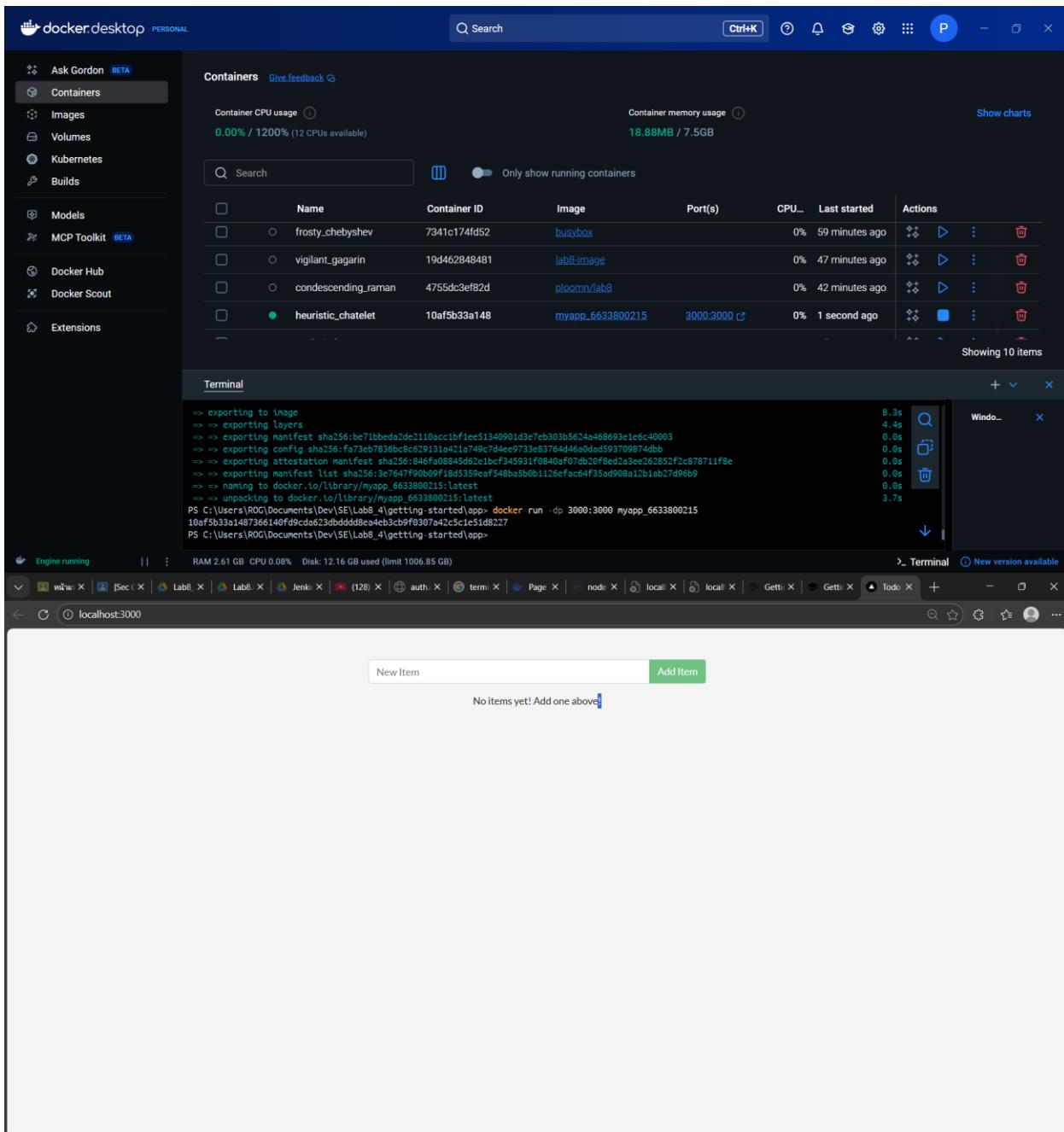
หน้าจอ

6. ทำการ Start ตัว Container ของแอปพลิเคชันที่สร้างขึ้น โดยใช้คำสั่ง

\$ docker run -dp 3000:3000 <myapp\_รหัส> ไม่มีชีด>

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

## Lab Worksheet



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

หมายเหตุ: นศ.สามารถทดลองเล่น Web application ที่ทำงานอยู่ได้

## Lab Worksheet

## 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>
```

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

By ชื่อและนามสกุลของนักศึกษา

- b. Save ไฟล์ให้เรียบร้อย

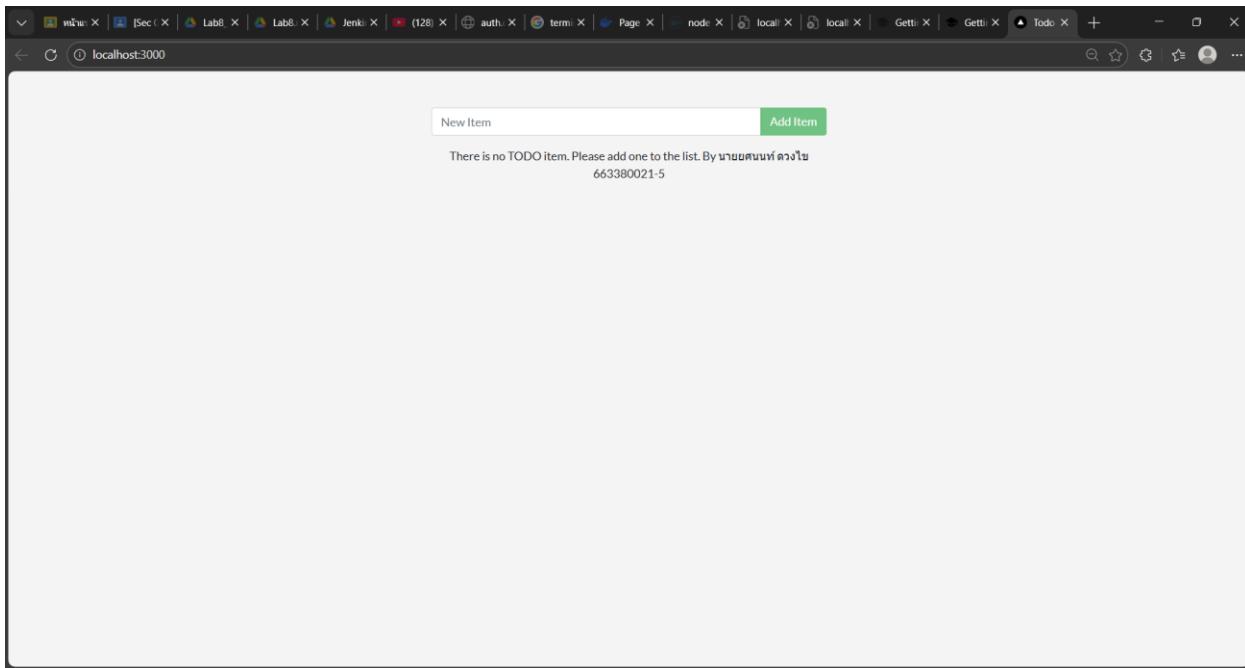
## 9. ทำการ Build Docker image โดยใช้คำสั่งเดียวกันกับข้อ 5

## 10. Start และรัน Container ตัวใหม่ โดยใช้คำสั่งเดียวกันกับข้อ 6

```

14  oListCard() {
15    ItemUpdate = React.useCallback(
16      ...
17    );
18    ms],
19  );
20  ItemRemoval = React.useCallback(
21    ...
22  );
23  const index = items.findIndex(i => i.id === item.id);
24  setItems([...items.slice(0, index), ...items.slice(index + 1)]);
25  ms,
26  );
27  s === null) return 'Loading...';
28  );
29  ct.Fragment>
30  <AddItemForm onNewItem={onNewItem} />
31  (items.length === 0 && (
32    <p className="text-center">There is no TODO item. Please add one to the list. By นายยศานันท์ ดวงไชย 663380021-5</p>
33  ))
34  (items.map(item => (
35    ...
36    <ItemDisplay
37      ...
38      item={item}
39      key={item.id}
40    >
41  ));
42  );
43  );
44  );
45  );
46  );
47  );
48  );
49  );
50  );
51  );
52  );
53  );
54  );
55  );
56  );
57  );
58  );
59  );
60  );
61  );

```

**Lab Worksheet**

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

(1) Error ที่เกิดขึ้นหมายความอย่างไร และเกิดขึ้นเพราะอะไร

ยังไม่มี Item ให้ทำการเพิ่มก่อน เกิดขึ้น เพราะ ไม่มี database เวลารันใหม่ข้อมูลก็จะหายหมด

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
- ใช้คำสั่ง \$ docker stop <Container ID> ที่ต้องการจะลบ > เพื่อหยุดการทำงานของ Container ดังกล่าว
- ใช้คำสั่ง \$ docker rm <Container ID> ที่ต้องการจะลบ > เพื่อทำการลบ

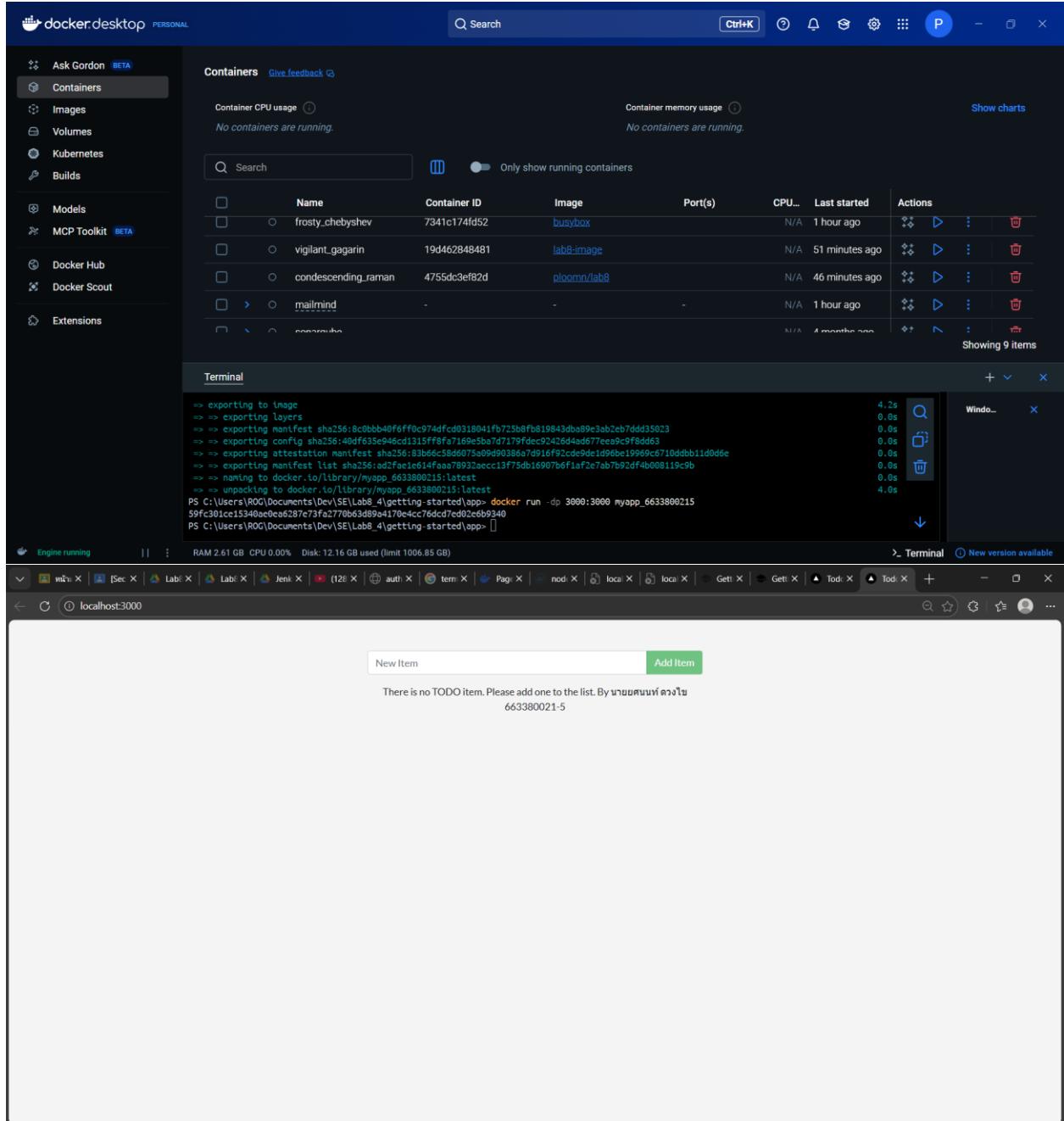
b. ผ่าน Docker desktop

- ไปที่หน้าต่าง Containers
- เลือกไอคอนถังขยะในແລາວของ Container ที่ต้องการจะลบ
- ยืนยันโดยการกด Delete forever

12. Start และรัน Container ตัวใหม่อีกครั้ง โดยใช้คำสั่งเดียวกันกับข้อ 6

## Lab Worksheet

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



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

## Lab Worksheet

### แบบฝึกปฏิบัติที่ 8.5: เริ่มต้นสร้าง Pipeline อย่างง่ายสำหรับการ Deploy ด้วย Jenkins

- สร้าง Dockerfile เพื่อสร้าง Jenkins และ Environment ที่เหมาะสมกับการรัน Robot framework ใน Container

[Check point#12] ส่ง Dockerfile ที่ใส่คำสั่งที่เกี่ยวข้องไว้

- เปิด Command line หรือ Terminal บน Docker Desktop
- Build image ที่สร้างขึ้นในข้อที่ 1 พร้อมกับตั้งชื่อของ image เป็น jenkins-robot-local  
\$ docker build -t jenkins-robot-local .
- รัน container โดยผูกพอร์ตให้เรียบร้อย เช่น

```
$ docker run -d \
  --name jenkins-robot \
  -p 8080:8080 -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
jenkins-robot-local
```

หรือ

```
$ docker run -it \
  --name jenkins-debug \
  -p 8080:8080 \
  -v /var/run/docker.sock:/var/run/docker.sock \
jenkins-robot-local
```

## Lab Worksheet

The screenshot shows the Docker Desktop interface. The top half displays a code editor with a Jenkinsfile. The file content is as follows:

```

FROM jenkins/jenkins:lts-jdk17 (last pushed 1 week ago)
USER root
RUN apt-get update && apt-get install -y --no-install-recommends \
    python3 python3-venv python3-pip \
    chromium chromium-driver \
    ca-certificates curl unzip \
    fonts-liberation \
    libnss3 libatk-bridge2.0-0 libgtk-3-0 libgbm1 \
    && rm -rf /var/lib/apt/lists/*
# :এঞ্জিন বেন্বি
RUN python3 -m venv /opt/venv \
&& /opt/venv/bin/pip install --no-cache-dir --upgrade pip \
&& /opt/venv/bin/pip install --no-cache-dir \
    robotframework \
    robotframework-seleniumlibrary \
    selenium
ENV PATH="/opt/venv/bin:$PATH"
USER jenkins

```

The bottom half of the interface shows a list of running Docker containers:

Name	Container ID	Image	Port(s)	CPU...	Last started	Actions
dreamy_pare	f5ca08d49bd2	myapp_6633800215	3000:3000	0%	22 minutes ago	<span>⋮</span> <span>⟳</span> <span>⋮</span> <span>刪</span>
mailmind	-	-	-	0%	2 hours ago	<span>⋮</span> <span>⟳</span> <span>⋮</span> <span>刪</span>
sonarqube	-	-	-	0%	4 months ago	<span>⋮</span> <span>⟳</span> <span>⋮</span> <span>刪</span>
api-gangbung-kafae	-	-	-	0%	4 months ago	<span>⋮</span> <span>⟳</span> <span>⋮</span> <span>刪</span>

A terminal window at the bottom shows the output of a Jenkins build command:

```

PS C:\Users\ROG\Documents\Dev\SE\Lab8_5> docker build -t jenkins-robot-local:latest .
=> >> exporting config sha256:4036e7c79fc6c8063d9380f29419204ba3ab6f4f247ca3357cb92f5934767cf
=> >> exporting attestation manifest sha256:2db49c82978611ca4417df530a85c84077de68db602a4ec9c53747a220b320b
=> >> exporting manifest list sha256:8bb296c0307e9283638cc204055bd2a50a5b72bcc634ec90968bea37f8a85b5d8
=> >> naming to docker.io/library/jenkins-robot-local:latest
=> >> unpacking to docker.io/library/jenkins-robot-local:latest

```

## Lab Worksheet

The screenshot shows the Docker Desktop interface. On the left, a sidebar lists various sections: Ask Gordon (BETA), Containers (selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions. The main area displays a table of running containers:

	Name	Container ID	Image	Port(s)	CPU...	Last started	Actions
<input type="checkbox"/>	dreamy_pare	f5ca08d49bd2	myapp_6633800215	3000:3000	0%	26 minutes ago	<span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	jenkins-robot	0efd28aa7eb9	jenkins-robot-local	8080:8080	77.4%	18 seconds ago	<span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	mailmind	-	-	-	0%	2 hours ago	<span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	sonargube	-	-	-	0%	4 months ago	<span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span> <span>⋮</span>

Showing 11 items

Below the table is a terminal window displaying the following command and its output:

```
At line:2 char:5
+ - -name jenkins-robot \
+     ~~~
Unexpected token 'name' in expression or statement.
+ CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : MissingExpressionAfterOperator

PS C:\Users\ROO\Documents\Dev\SE\Lab8_5> docker run -d -name jenkins-robot -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock jenkins-robot-local
0efd28aa7eb9:0e3c4d5c9906351ccac1c5e4e0ed673ec004075b6cb47419368
PS C:\Users\ROO\Documents\Dev\SE\Lab8_5> [
```

The browser tab at the bottom shows the Jenkins setup page: [localhost:8080/login?from=%2F](http://localhost:8080/login?from=%2F). The page content is:

**Getting Started**

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

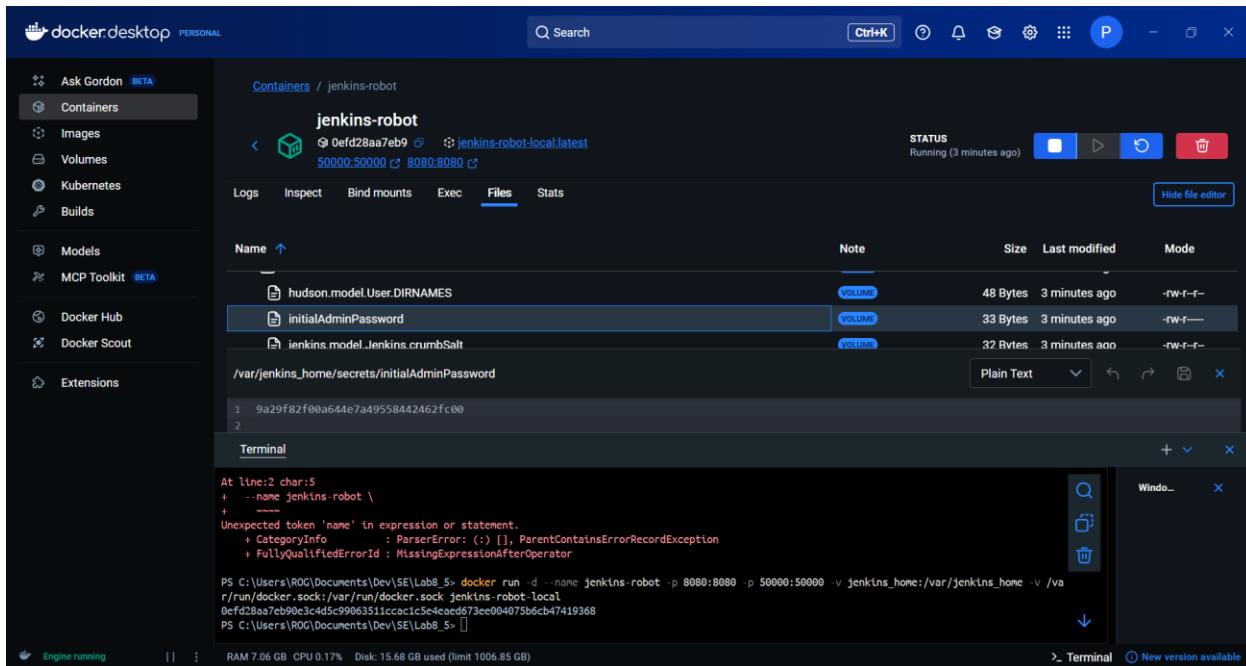
```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

**Administrator password**

**Continue**

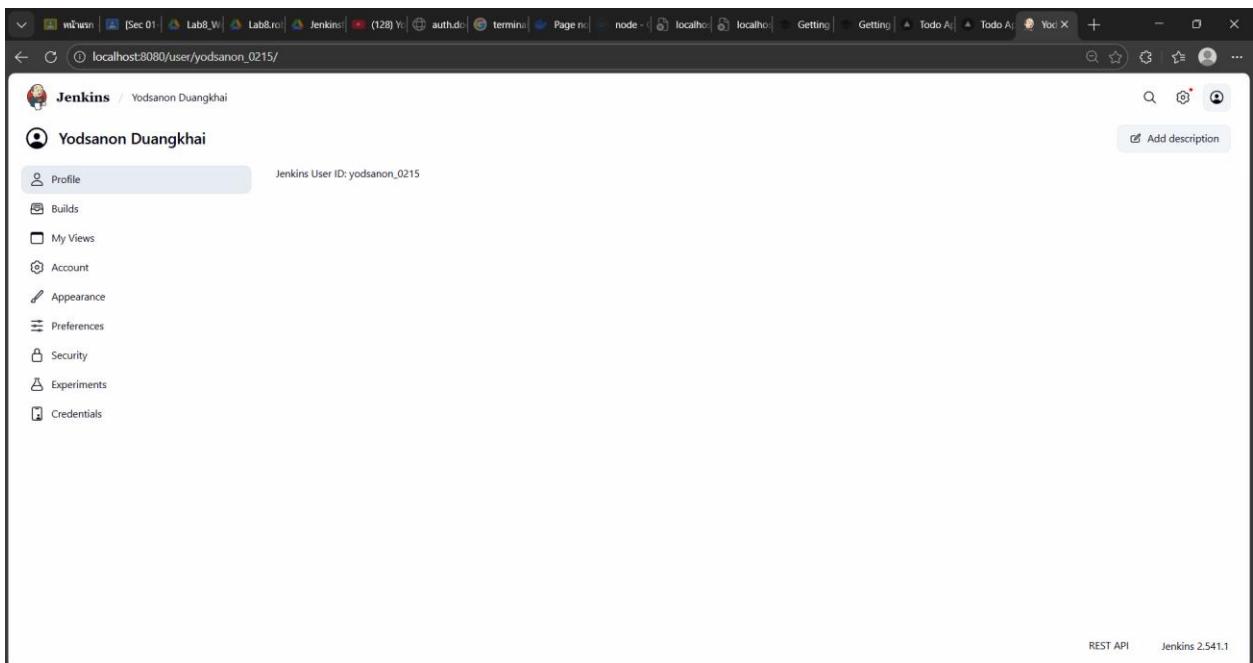
## Lab Worksheet



[Check point#13] Capture หน้าจอที่แสดงผล Admin password

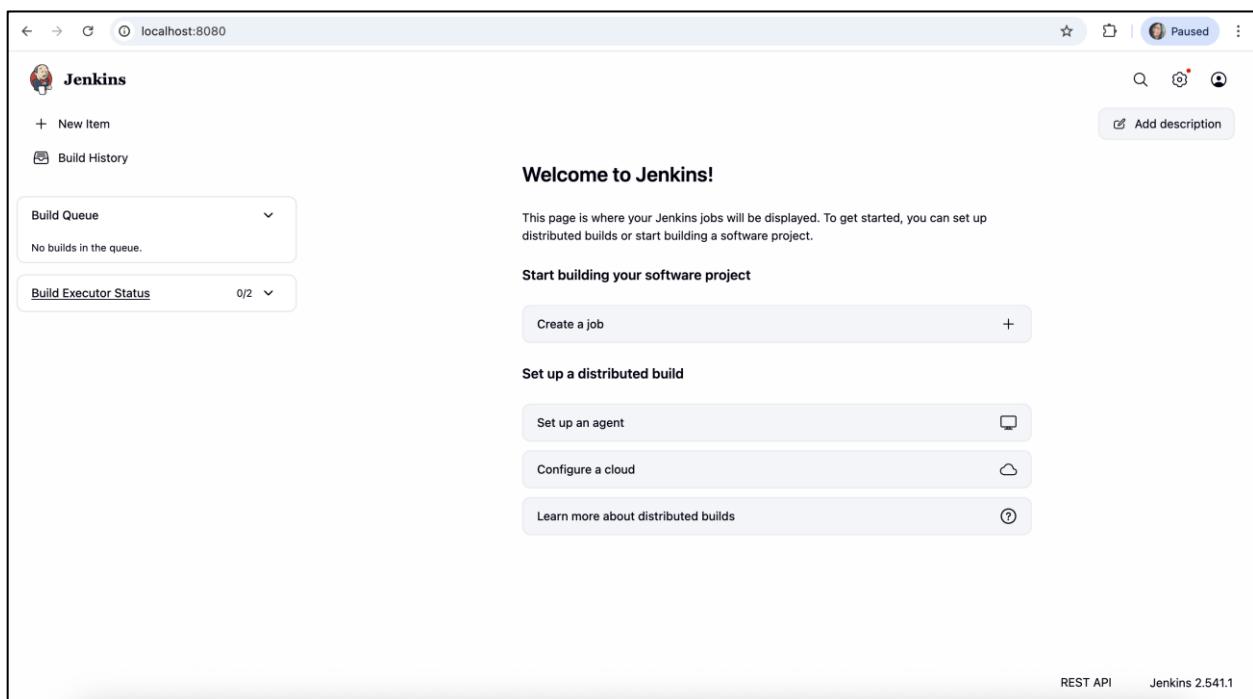
5. บันทึกรหัสผ่านของ Admin user ไว้สำหรับ log-in ในครั้งแรก
6. เมื่อได้รับการยืนยันว่า Jenkins is fully up and running ให้เปิดเบราว์เซอร์ และป้อนที่อยู่เป็น <http://localhost:8080>
7. ทำการ Unlock Jenkins ด้วยรหัสผ่านที่ได้ในข้อที่ 3
8. สร้าง Admin User โดยใช้ username เป็นชื่อจริงของนักศึกษาพร้อมรหัสสี่ตัวท้าย เช่น somsri\_3062

## Lab Worksheet



[Check point#14] Capture หน้าจอที่แสดงผลการตั้งค่า

9. เมื่อติดตั้งเรียบร้อยแล้วจะพบกันหน้า Dashboard ดังแสดงในภาพ



10. เลือก Manage Jenkins และไปที่เมนู Plugins

## Lab Worksheet

The screenshot shows the Jenkins Manage Jenkins interface at the URL [localhost:8080/manage/](http://localhost:8080/manage/). The main navigation bar includes links for Home, Manage Jenkins, Jenkins, Help, and Logout. A user icon indicates the session is Paused.

**System Configuration**

- System**: Configure global settings and paths.
- Nodes**: Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- Tools**: Configure tools, their locations and automatic installers.
- Clouds**: Add, remove, and configure cloud instances to provision agents on-demand.
- Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- Appearance**: Configure the look and feel of Jenkins.

**Security**

- Security**: Secure Jenkins; define who is allowed to access/use the system.
- Credentials**: Configure credentials.
- Users**: Create/delete/modify users that can log in to this Jenkins.
- Credential Providers**: Configure the credential providers and types.

**Status Information**

- System Information**: Displays various environmental information to assist trouble-shooting.
- System Log**: System log captures output from `java.util.logging` related to Jenkins.
- Load Statistics**: Check your resource utilization and see if you need more computers for your builds.

11. ไปที่เมนู Plugins > Available plugins และเลือกติดตั้ง Robot Framework เพิ่มเติม

The screenshot shows the Jenkins Manage Jenkins interface at the URL [localhost:8080/manage/pluginManager/available](http://localhost:8080/manage/pluginManager/available). The left sidebar has tabs for Updates, Available plugins (selected), Installed plugins, Advanced settings, and Download progress. The main area shows a search bar with "Robot" and a list of available plugins:

Install	Name	Released	Health
<input type="checkbox"/>	Robot Framework 6.2.0	Build Reports This publisher stores Robot Framework test reports for builds and shows summaries of them in project and build views along with trend graph. 3 mo 15 days ago	91

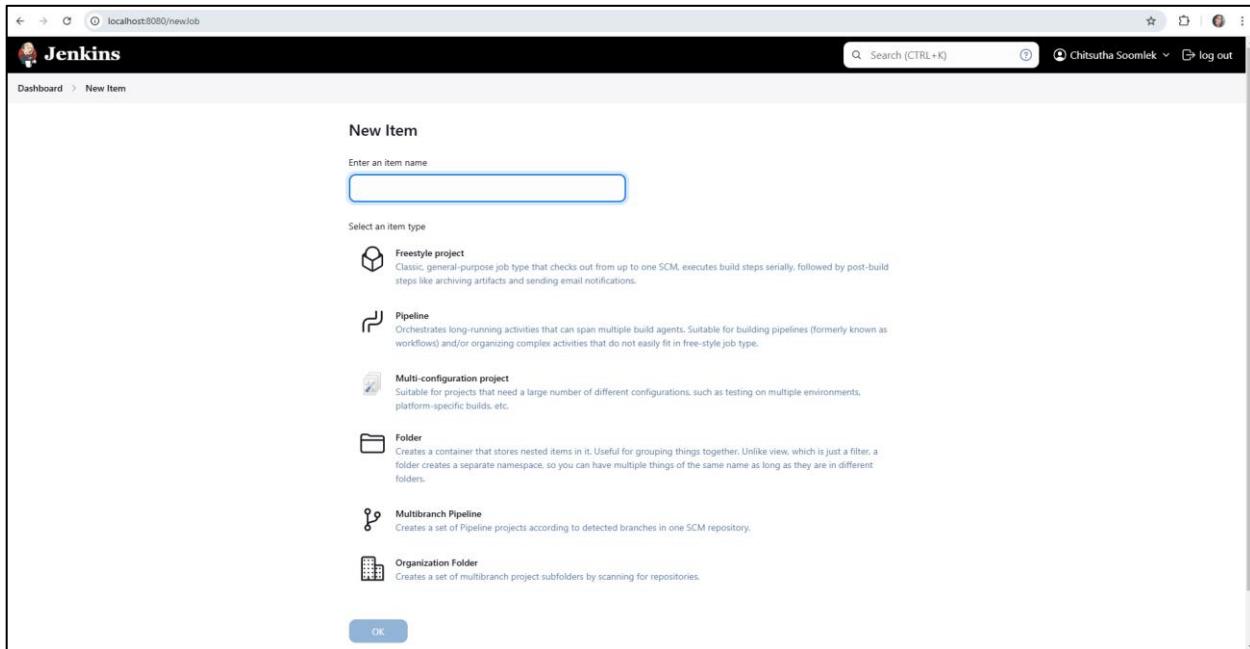
12. เมื่อติดตั้งสำเร็จจะพบกับรายการ Plugins ทั้งหมดที่ถูกติดตั้ง ถ้าติดตั้งสำเร็จให้เลือก “Restart Jenkins...” และกด Go back to the top page

## Lab Worksheet

The screenshot shows the Jenkins plugin manager interface at the URL `localhost:8080/manage/pluginManager/updates/`. The left sidebar has a 'Plugins' section with links for 'Updates', 'Available plugins', 'Installed plugins', 'Advanced settings', and 'Download progress'. The 'Download progress' link is highlighted with a grey box. The main content area lists installed plugins with green checkmarks and the word 'Success' next to each. Plugins listed include Pipeline: GitHub Groovy Libraries, Metrics, Pipeline Graph View, Git, EDDSA API, Trilead API, SSH Build Agents, Matrix Authorization Strategy, LDAP, jsoup API, Email Extension, Mailer, Theme Manager, Dark Theme, Loading plugin extensions, Robot Framework, and Loading plugin extensions. At the bottom, there are two buttons: 'Go back to the top page' and 'Restart Jenkins when installation is complete and no jobs are running'.

13. สร้างไฟล์ Jenkinsfile ไม่มีนามสกุล เพื่อ execute คำสั่งต่าง ๆ กับ built-in agent และเอาไฟล์ดังกล่าว เก็บไว้ที่ root ของ GitHub Repository ของนักศึกษา
14. สร้าง folder ชื่อ tests/ บน GitHub Repository ของนักศึกษา และสร้างไฟล์ Lab8.robot และนำไฟล์ไปไว้ folder ที่สร้าง
15. กลับไปที่หน้า Dashboard ของ Jenkins และสร้าง Pipeline อย่างง่าย โดยกำหนด New item เป็น Freestyle project และตั้งชื่อเป็น UAT

## Lab Worksheet



16. ตั้งค่าที่จำเป็นในหน้านี้ทั้งหมด ดังนี้

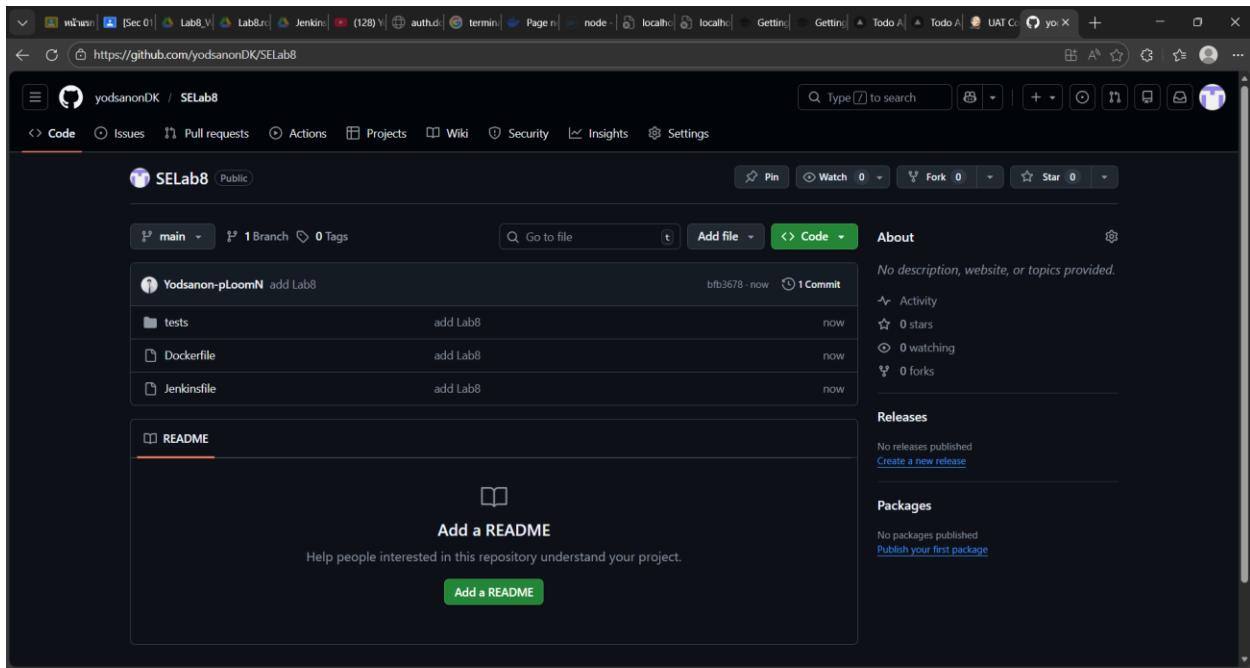
**Description:** Lab 8.5

**GitHub project:** กดเลือก และใส่ Project URL เป็น repository ที่เก็บโค้ด .robot (ดูขั้นตอนที่ 14)

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

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

## Lab Worksheet



## Lab Worksheet

The image contains two screenshots of the Jenkins configuration interface for a job named "UAT".

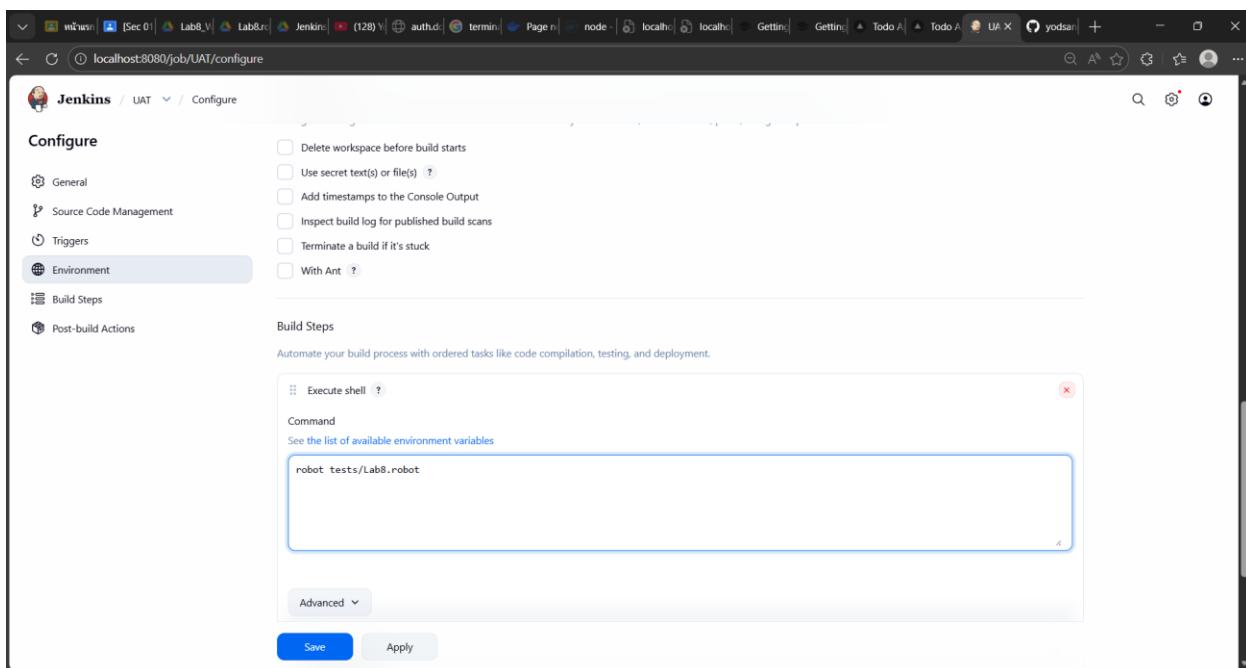
**Screenshot 1: General Configuration**

- General:**
  - Description:** Lab 8.5
  - Enabled:**
  - GitHub project:**  Project url: https://github.com/yodsonDK/SELab8
  - Advanced:**
    - This project is parameterized
    - Throttle builds
    - Execute concurrent builds if necessary
- Buttons:** Save, Apply

**Screenshot 2: Triggers Configuration**

- Triggers:**
  - Build periodically:**  Schedule: H/15 \* \* \* \*
  - GitHub hook trigger for GITScm polling**
  - Poll SCM**
- Environment:**
  - Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.**
  - Delete workspace before build starts
  - Use secret text(s) or file(s)
  - Add timestamps to the Console Output
- Buttons:** Save, Apply

## Lab Worksheet



[Check point#15] Capture หน้าจอแสดงการตั้งค่า พร้อมกับตอบคำถามต่อไปนี้

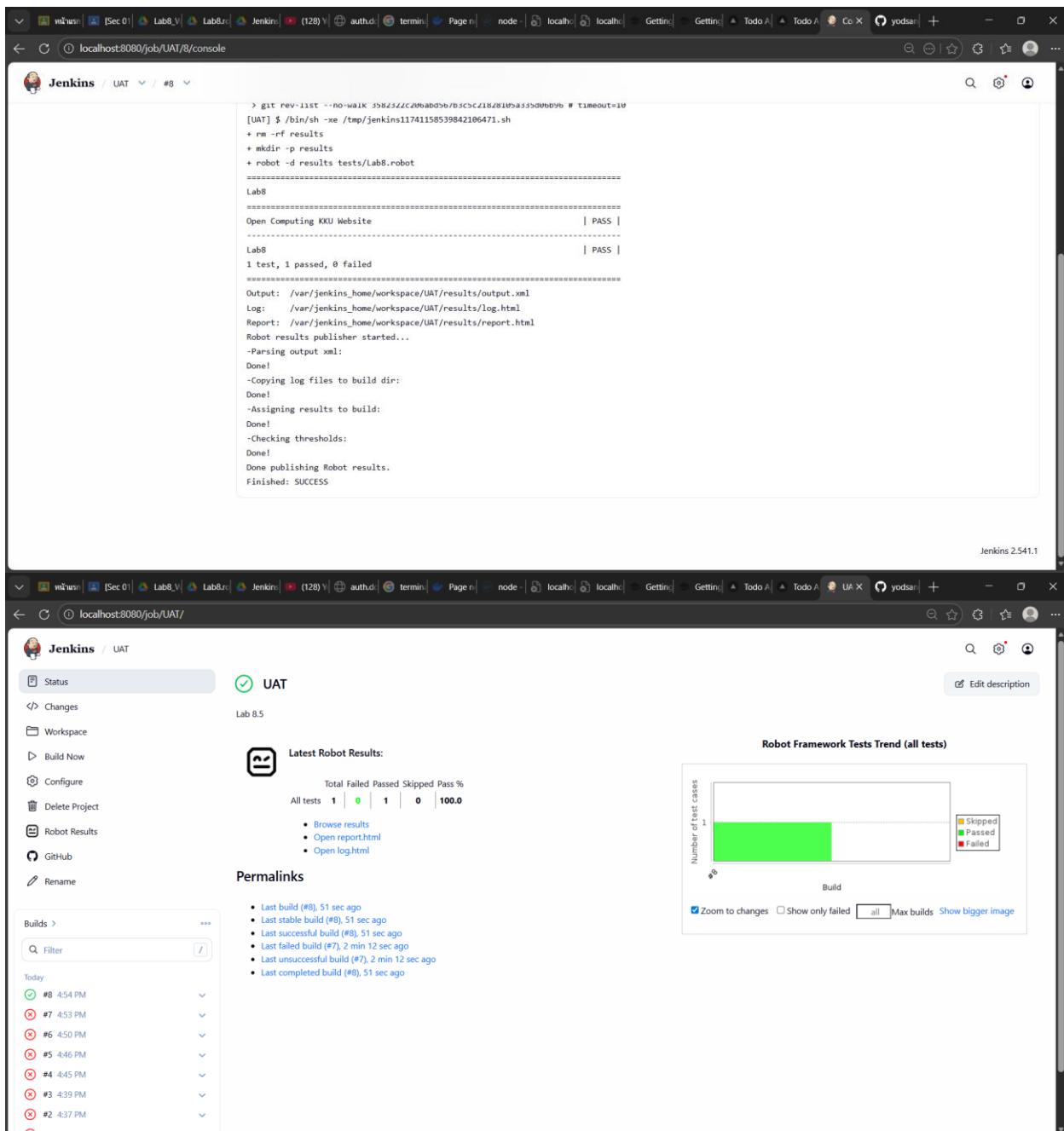
(1) คำสั่งที่ใช้ในการ Execute ไฟล์ .robot ใน Build Steps คือ **robot tests/Lab8.robot**

**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 )

17. กด Apply และ Save

18. สร้าง Build Now

## Lab Worksheet



The screenshot shows two stacked Jenkins job pages. The top page is the 'Console' output for build #8 of the 'UAT' job. It displays command-line logs for running a Robot Framework test suite named 'Lab8'. The logs show the test results: 1 test passed, 0 failed. It also shows the generation of XML and HTML reports and the publishing of results. The bottom page is the 'Build History' for the 'UAT' job, listing builds from #1 to #8. Build #8 is the most recent and successful. A 'Robot Framework Tests Trend (all tests)' chart is shown, indicating 1 total test case, all of which passed.

[Check point#16] Capture หน้าจอแสดงหน้าหลักของ Pipeline และ Console Output