

Lab Worksheet

ชื่อ-นามสกุล : พิศันติ ศิริพุทธา รหัสนักศึกษา : 653380142-2 Section : 1

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

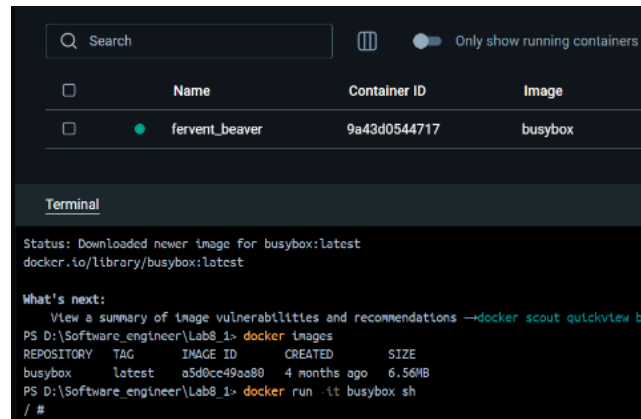
1. ติดตั้ง Docker desktop ลงบนเครื่องคอมพิวเตอร์ โดยดาวน์โหลดจาก <https://www.docker.com/get-started>
2. สร้าง Account บน Docker hub (<https://hub.docker.com/signup>)
3. กำหนดให้ \$ หมายถึง Command prompt และ <> หมายถึง ให้ป้อนค่าของพารามิเตอร์ที่กำหนด

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



- (1) สิ่งที่อยู่ภายใต้คอลัมน์ Repository คืออะไร : รายชื่อของ image
- (2) Tag ที่ใช้บ่งบอกถึงอะไร : ใช้บ่งบอกถึง version ของ busybox
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

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

```
What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quick
PS D:\Software_engineer\Lab8_1> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
busybox latest a5d0ce49aa80 4 months ago 6.56MB
PS D:\Software_engineer\Lab8_1> docker run -it busybox sh
/ # ls
bin dev etc home lib lib64 proc root sys tmp usr var
/ #

PS D:\Software_engineer\Lab8_1> docker run -it busybox sh
/ # ^C

drwxr-xr-x 1 root root 4096 Jan 27 10:29 ..
-rwxr-xr-x 1 root root 0 Jan 27 10:29 .dockerenv
drwxr-xr-x 2 root root 12288 Sep 26 21:31 bin
drwxr-xr-x 5 root root 360 Jan 27 10:29 dev
drwxr-xr-x 1 root root 4096 Jan 27 10:29 etc
drwxr-xr-x 2 nobody nobody 4096 Sep 26 21:31 home
drwxr-xr-x 2 root root 4096 Sep 26 21:31 lib
lrwxrwxrwx 1 root root 3 Sep 26 21:31 lib64 -> lib
dr-xr-xr-x 238 root root 0 Jan 27 10:29 proc
drwx----- 1 root root 4096 Jan 27 10:29 root
dr-xr-xr-x 11 root root 0 Jan 27 10:29 sys
drwxrwxrwt 2 root root 4096 Sep 26 21:31 tmp
drwxr-xr-x 4 root root 4096 Sep 26 21:31 usr
drwxr-xr-x 4 root root 4096 Sep 26 21:31 var
..
PS D:\Software_engineer\Lab8_1> docker run busybox echo "Hello Peerasanti Sriputta from busybox"
Hello Peerasanti Sriputta from busybox
PS D:\Software_engineer\Lab8_1>

PS D:\Software_engineer\Lab8_1> docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
6b55822cdaf7 busybox "echo 'Hello Peerasa..." 38 seconds ago Exited (0) 36 seconds ago nice_goodall
2b33e35b722f busybox "sh" 2 minutes ago Exited (0) About a minute ago laughing_keldysh
PS D:\Software_engineer\Lab8_1>
```

(1) เมื่อใช้ option -it ในคำสั่ง run ส่งผลต่อการทำงานของคำสั่งอย่างไรบ้าง อธิบายมาพอสังเขป

ตอบ : คำสั่ง -it เป็นคำสั่งที่มีผลต่อการทำงานของ container ในลักษณะของการตอบโต้กับ terminal ได้

(2) คอลัมน์ STATUS จากการรันคำสั่ง docker ps -a แสดงถึงข้อมูลอะไร

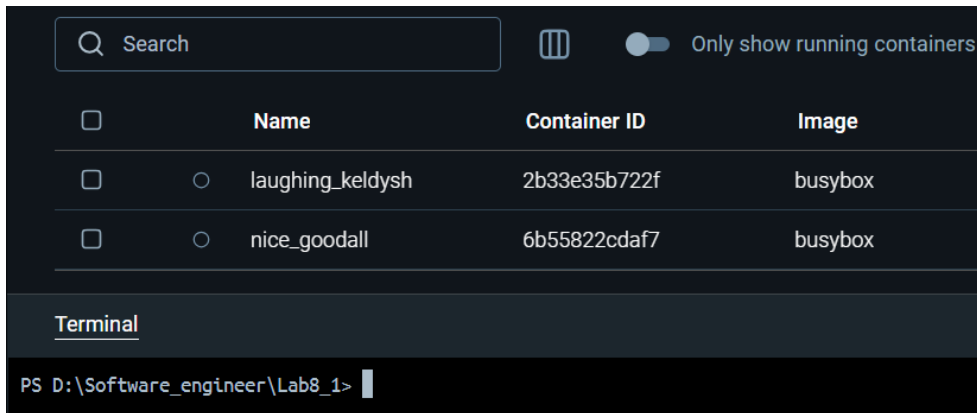
ตอบ : แสดงให้เห็นถึงสถานะของแต่ละ container ระบบโดยจะมี 3 สถานะนั้นก็คือ (1) กำลังทำงานอยู่ (2) หยุดการทำงานไปแล้วและ (3) กำลังรอการทำงาน

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

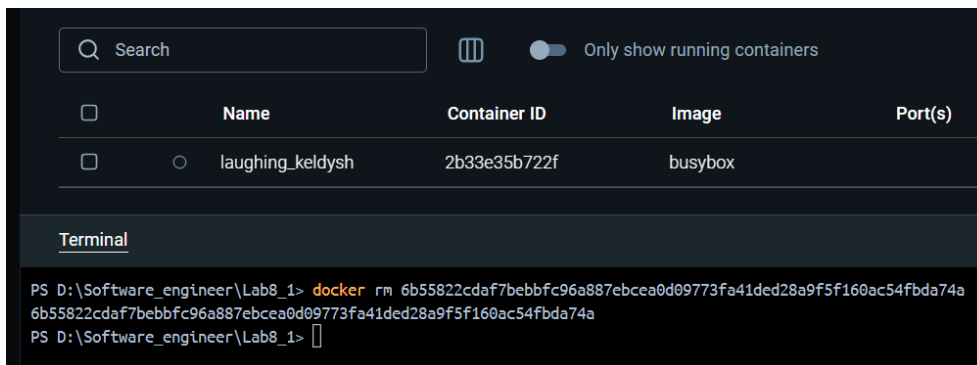
Lab Worksheet

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

- ก่อนที่จะทำการลบ container



- หลังจากทำการลบ container ออกไปโดย container ที่ลบออกไปนั้นก็คือ container ที่ชื่อ nice_goodall



Lab Worksheet

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

5. ทำการ Build Docker image ที่สร้างขึ้นด้วยคำสั่งต่อไปนี้

```
$ docker build -t <ชื่อ Image> .
```

6. เมื่อ Build สำเร็จแล้ว ให้ทำการรัน Docker image ที่สร้างขึ้นในขั้นตอนที่ 5

Lab Worksheet

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

```

Terminal

PS D:\Software_engineer\Lab8_2> docker build -t lab8-2 .
[+] Building 0.7s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 152B
=> [internal] load metadata for docker.io/library/busybox:latest
=> [internal] load .dockerignore
=> => transferring context: 2B

3 warnings found (use docker --debug to expand):
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS s
- MultipleInstructionsDisallowed: Multiple CMD instructions should not be used in the same stage because
)
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS s

What's next:
  View a summary of image vulnerabilities and recommendations →docker scout quickview
PS D:\Software_engineer\Lab8_2>

```

(1) คำสั่งที่ใช้ในการ run คือ

ตอบ : \$ docker run lab8-2

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

ตอบ : ใช้ในการกำหนด tag ให้กับ docker image ที่จะสร้างขึ้นเพื่อช่วยให้ระบุชื่อกับ version ของ image ได้
ง่ายมากขึ้น

Lab Worksheet

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

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
```

Lab Worksheet

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

```

Terminal
PS D:\Software_engineer\Lab8_3> docker run peerasanti/lab8
Peerasanti Sriputta 653380142-2
PS D:\Software_engineer\Lab8_3>

```

6. ทำการ Push ตัว Docker image ไปไว้บน Docker Hub โดยการใช้คำสั่ง

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

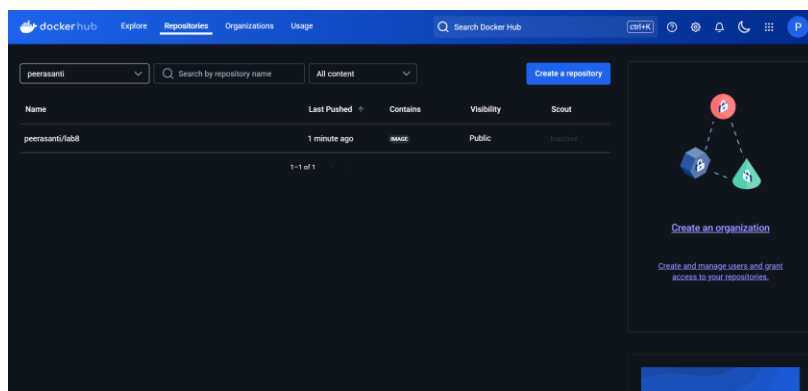
ในกรณีที่ติดปัญหาไม่ได้ Login ไว้ก่อน ให้ใช้คำสั่งต่อไปนี้ เพื่อ Login ก่อนทำการ Push

\$ 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

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



```

PS D:\Software_engineer\Lab8_3> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
peerasanti/lab8     latest     836829bf023a  4 months ago  6.56MB
lab8-2              latest     160f1a5dcd8f  4 months ago  6.56MB
busybox             latest     a5d0ce49aa80  4 months ago  6.56MB
PS D:\Software_engineer\Lab8_3>

```


Lab Worksheet

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

```

Terminal
PS D:\Software_engineer\Lab8_4> ls

Directory: D:\Software_engineer\Lab8_4

Mode                LastWriteTime         Length Name
----                -
d-----          1/28/2025  12:41 AM             getting-started

PS D:\Software_engineer\Lab8_4> cd getting-started
PS D:\Software_engineer\Lab8_4\getting-started>

```

```

package.json X
D: > Software_engineer > Lab8_4 > getting-started > app > package.json > ...

1  {
2    "name": "101-app",
3    "version": "1.0.0",
4    "main": "index.js",
5    "license": "MIT",
6    "scripts": {
7      "prettify": "prettier -l --write \"**/*.js\"",
8      "test": "jest",
9      "dev": "nodemon src/index.js"
10   },
11   "dependencies": {
12     "express": "^4.18.2",
13     "mysql2": "^2.3.3",
14     "sqlite3": "^5.1.2",
15     "uuid": "^9.0.0",
16     "wait-port": "^1.0.4"
17   },
18   "resolutions": {
19     "ansi-regex": "5.0.1"
20   },
21   "prettier": {
22     "trailingComma": "all",
23     "tabWidth": 4,
24     "useTabs": false,
25     "semi": true,
26     "singleQuote": true
27   },
28   "devDependencies": {
29     "jest": "^29.3.1",
30     "nodemon": "^2.0.20",
31     "prettier": "^2.7.1"
32   }
33 }
34

```

Lab Worksheet

4. ภายใต้ getting-started/app ให้สร้าง Dockerfile พร้อมกับใส่เนื้อหาดังต่อไปนี้ลงไปไฟล์

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_รหัสสนศ. ไม่มีขีด

\$ docker build -t <myapp_รหัสสนศ. ไม่มีขีด> .

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

```
PS D:\Software_engineer\Lab8_4\getting-started> cd app
PS D:\Software_engineer\Lab8_4\getting-started\app> echo "" > Dockerfile
PS D:\Software_engineer\Lab8_4\getting-started\app> docker build -t myapp_6533801422 .
[+] Building 25.9s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 150B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/node:18-alpine@sha256:974af6b6cbc0314dc6502b14243b8a39fbb2d04d975e9059dd066be3e274fbb25
=> resolve docker.io/library/node:18-alpine@sha256:974af6b6cbc0314dc6502b14243b8a39fbb2d04d975e9059dd066be3e274fbb25
=> sha256:5650d6de56fd0bb419872b876ac1df28f577b39573c3b72fb0d15bf426d01bc1 1.26MB / 1.26MB
=> sha256:37892ffbfcaa871a10f813803949d18c3015a482051d51b7e0da02525e63167c 40.01MB / 40.01MB
=> sha256:1f3e46996e2966e4faa5846e56e76e3748b7315e2ded61476c24403d592134f0 3.64MB / 3.64MB
=> sha256:6504e29600c8d5213b52cda800370abb3d12639802d06b46b6fce368990ca771 444B / 444B
```

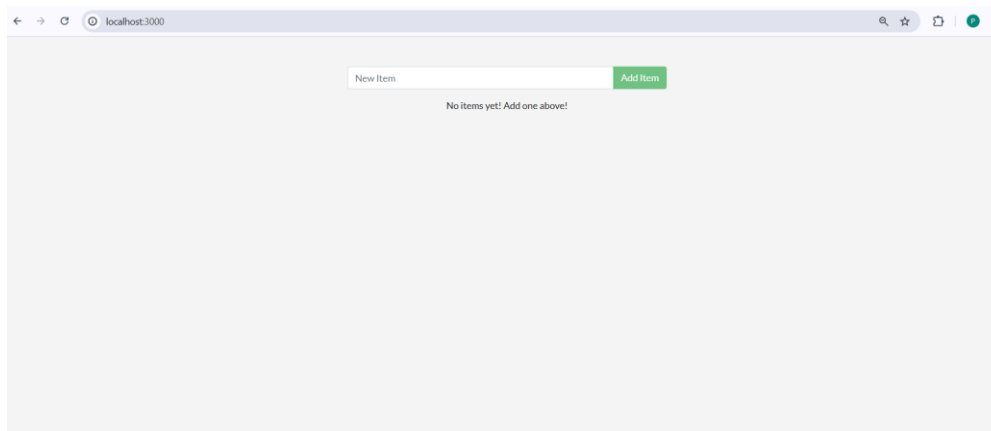
6. ทำการ Start ตัว Container ของแอปพลิเคชันที่สร้างขึ้น โดยใช้คำสั่ง

\$ 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



Container CPU usage ⓘ

0.00% / 1200% (12 CPUs available)

Container memory usage ⓘ

22.15MB / 3.66GB

Show charts

Q

Search

☰

Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	<div><div></div><div>awesome_hugle</div></div>	b5f82fa80041	lab8-2		0%	36 minutes ago	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	<div><div></div><div>intelligent_heisenberg</div></div>	44f90d8bc6c1	peerasanti/lab8		0%	24 minutes ago	<div><div></div><div></div><div></div></div>
<input checked="" type="checkbox"/>	<div><div></div><div>pedantic_mccarthy</div></div>	579395fbbf42	myapp_6533801422	3000:3000 ↗	0%	1 minute ago	<div><div></div><div></div><div></div></div>

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

Lab Worksheet

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

```

Terminal

View build details: docker_desktop://dashboard/build/desktop_linux/desktop_linux/ovnpdc5agtkrq43fzih5zs3w

What's next:
  View a summary of image vulnerabilities and recommendations →docker scout quickview
PS D:\Software_engineer\Lab8_4\getting-started\app> docker run -dp 3000:3000 myapp_6533801422_fixed
567a70627abb711bce12aec9709fbbe02c9e8101cfa64e7175f710fad6daf03b
docker: Error response from daemon: driver failed programming external connectivity on endpoint gifted_kowalewski (87e42d05cc7737f1dc62efb5ba9fb10d951405c585ace4aec238e2815f09577f): Bind for 0.0.0.0:3000 failed: port is already allocated.
PS D:\Software_engineer\Lab8_4\getting-started\app>
  
```

(1) Error ที่เกิดขึ้นหมายความว่าอย่างไร และเกิดขึ้นเพราะอะไร

ตอบ : Error ที่ขึ้นเกิดขึ้นจากที่ container ที่เพิ่งสร้างขึ้นใหม่ต้องการจะรันในพอร์ต 3000 แต่มี container ก่อนหน้าที่รันพอร์ตนี้ไปก่อนหน้าแล้ว (container : myapp_6533801422)

11. ลบ Container ของ Web application เวอร์ชันก่อนแก้ไขออกจากระบบ โดยใช้วิธีใดวิธีหนึ่งดังต่อไปนี้

a. ผ่าน Command line interface

- i. ใช้คำสั่ง \$ docker ps เพื่อดู Container ID ที่ต้องการจะลบ
- ii. Copy หรือบันทึก Container ID ไว้
- iii. ใช้คำสั่ง \$ docker stop <Container ID ที่ต้องการจะลบ> เพื่อหยุดการทำงานของ Container ดังกล่าว
- iv. ใช้คำสั่ง \$ docker rm <Container ID ที่ต้องการจะลบ> เพื่อทำการลบ

b. ผ่าน Docker desktop

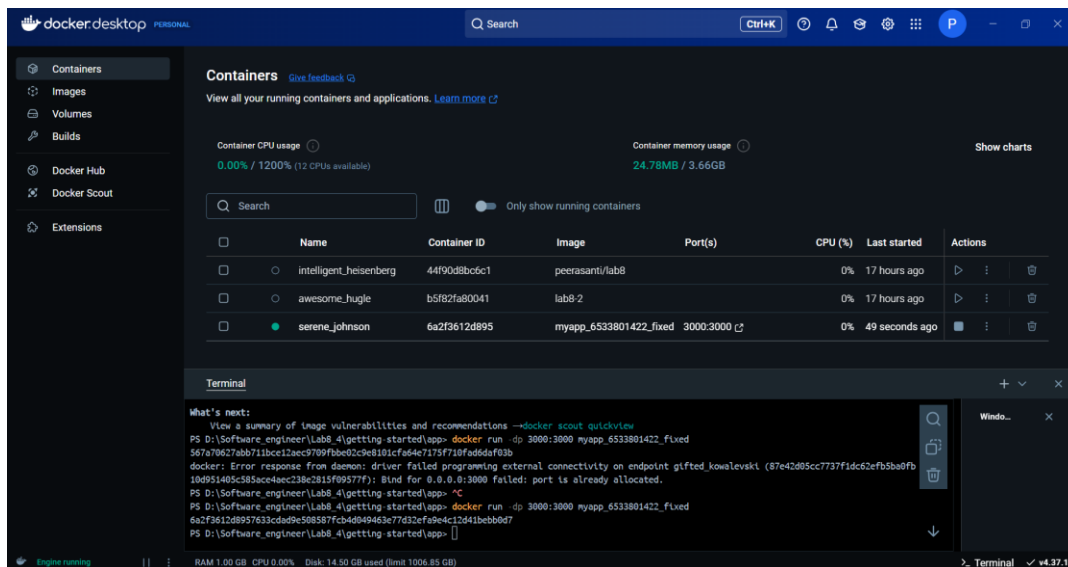
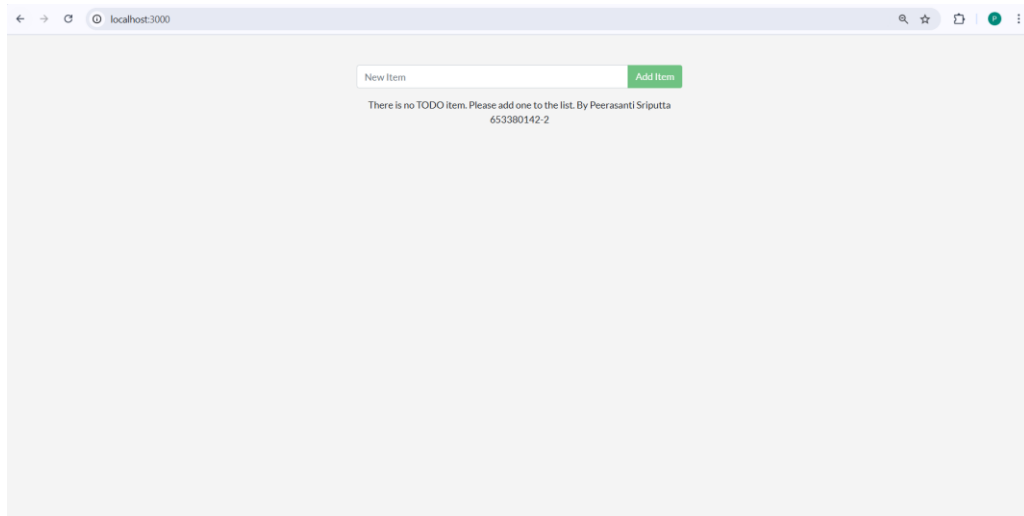
- i. ไปที่หน้าต่าง Containers
- ii. เลือกไอคอนถังขยะในแถวของ Container ที่ต้องการจะลบ
- iii. ยืนยันโดยการกด Delete forever

12. Start และรัน Container ตัวใหม่อีกครั้ง โดยใช้คำสั่งเดียวกันกับข้อ 6

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

Lab Worksheet

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



Lab Worksheet

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
```

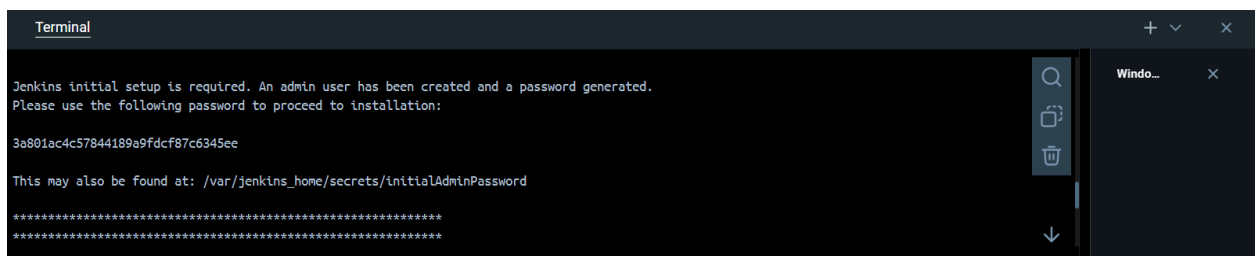
หรือ

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

```
jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk17
```

3. บันทึกรหัสผ่านของ Admin user ไว้สำหรับ log-in ในครั้งแรก

[Check point#12] Capture หน้าจอที่แสดงผล Admin password

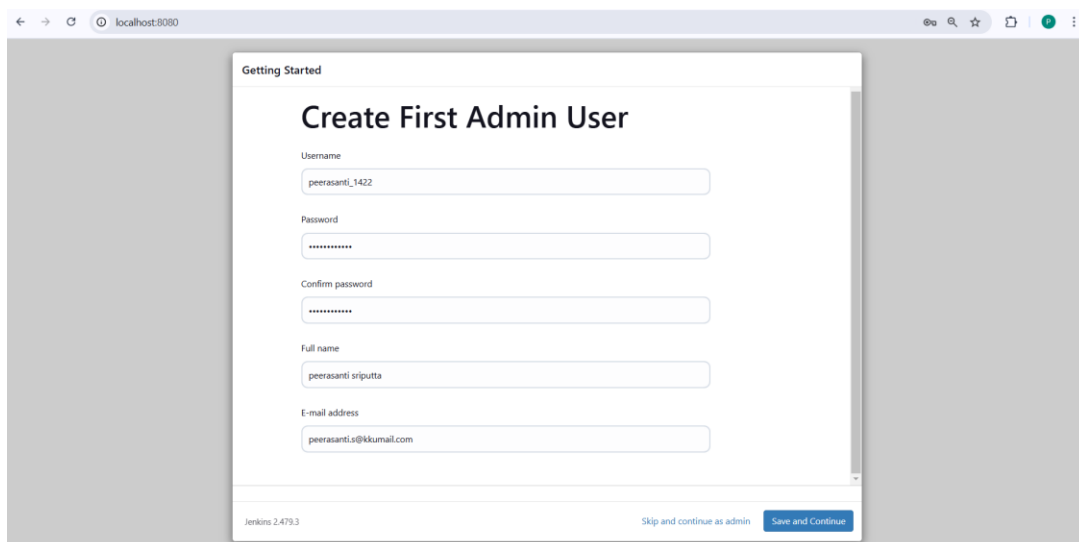


4. เมื่อได้รับการยืนยันว่า Jenkins is fully up and running ให้เปิดบราวเซอร์ และป้อนที่อยู่เป็น localhost:8080

5. ทำการ Unlock Jenkins ด้วยรหัสผ่านที่ได้ในข้อที่ 3

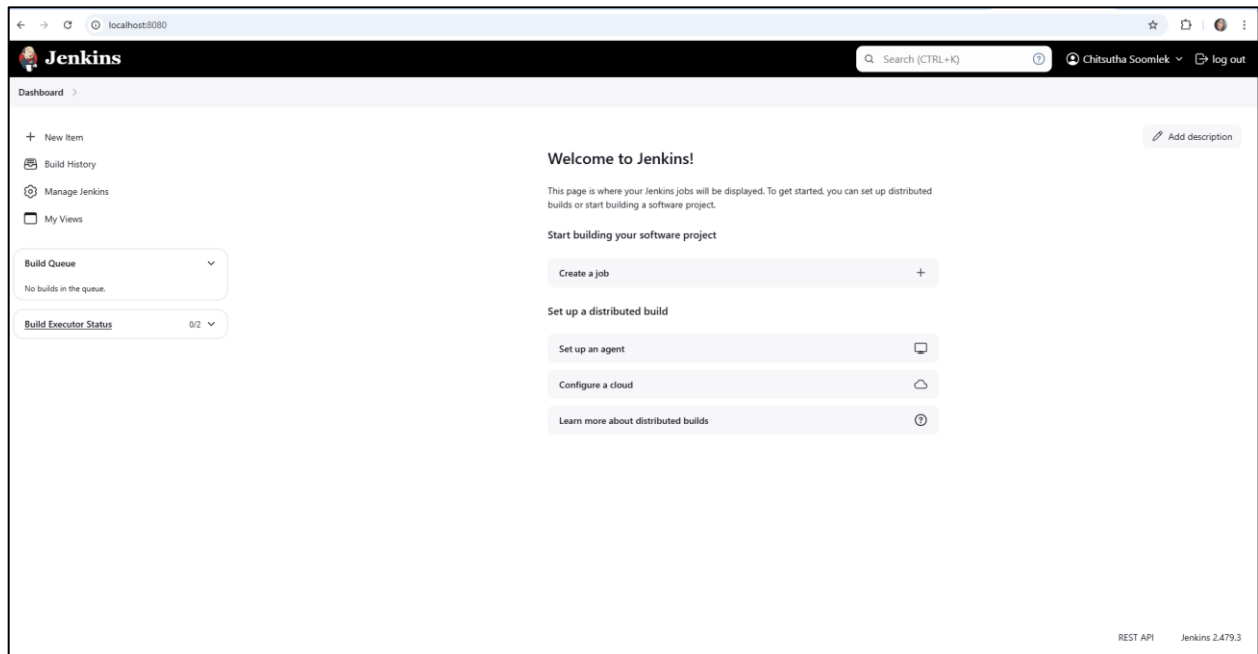
6. สร้าง Admin User โดยใช้ username เป็นชื่อจริงของนักศึกษาพร้อมรหัสสี่ตัวท้าย เช่น somsri_3062

[Check point#13] Capture หน้าจอที่แสดงผลการตั้งค่า

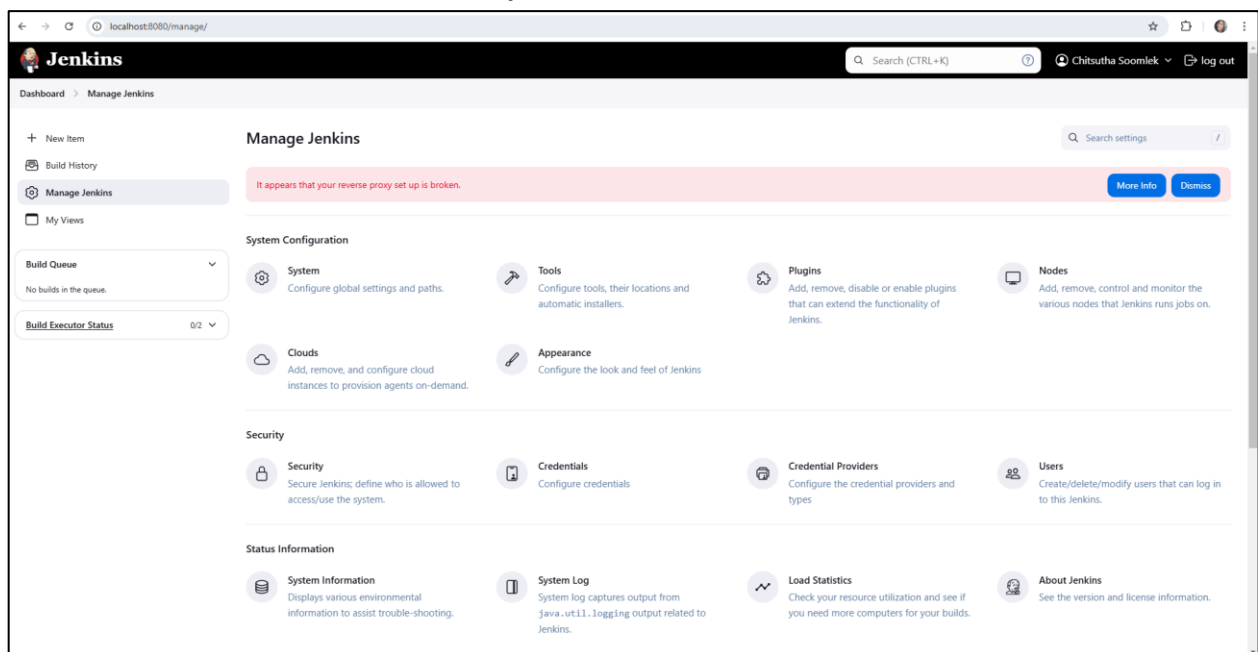


Lab Worksheet

7. กำหนด Jenkins URL เป็น <http://localhost:8080/lab8>
8. เมื่อติดตั้งเรียบร้อยแล้วจะพบหน้าจอ Dashboard ดังแสดงในภาพ

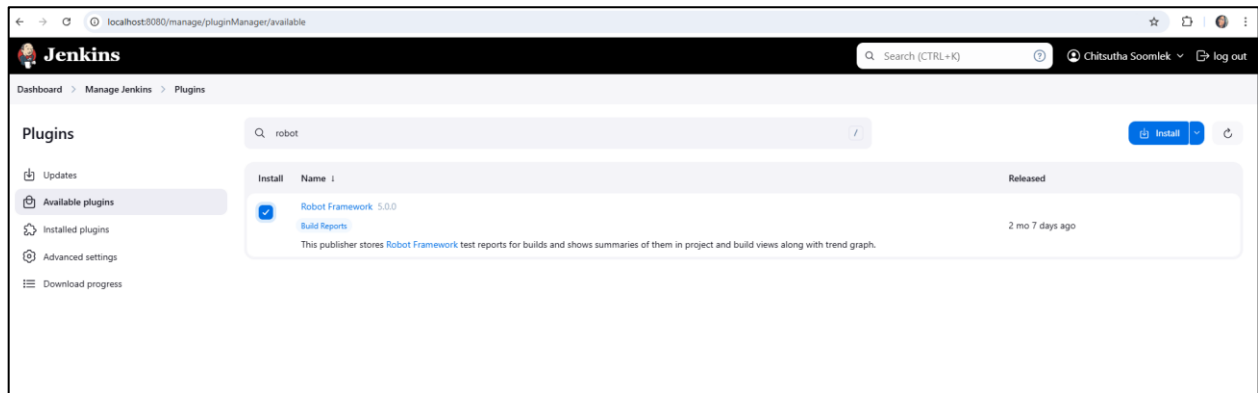


9. เลือก Manage Jenkins แล้วไปที่เมนู Plugins

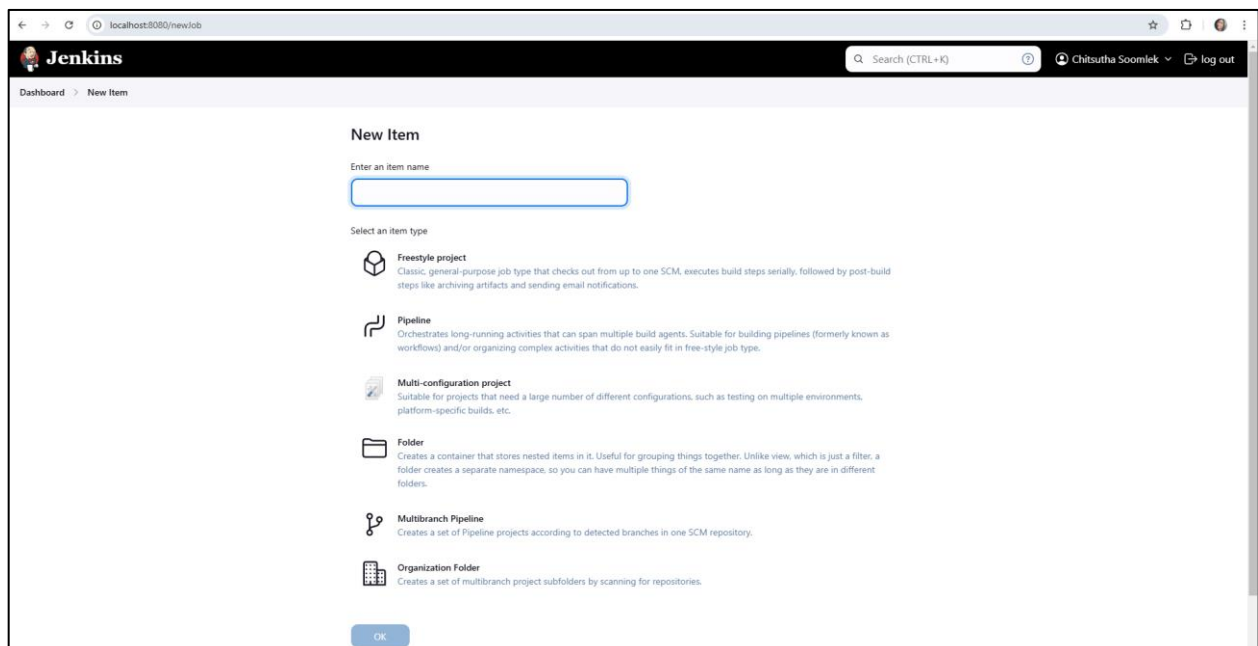


Lab Worksheet

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

Lab Worksheet

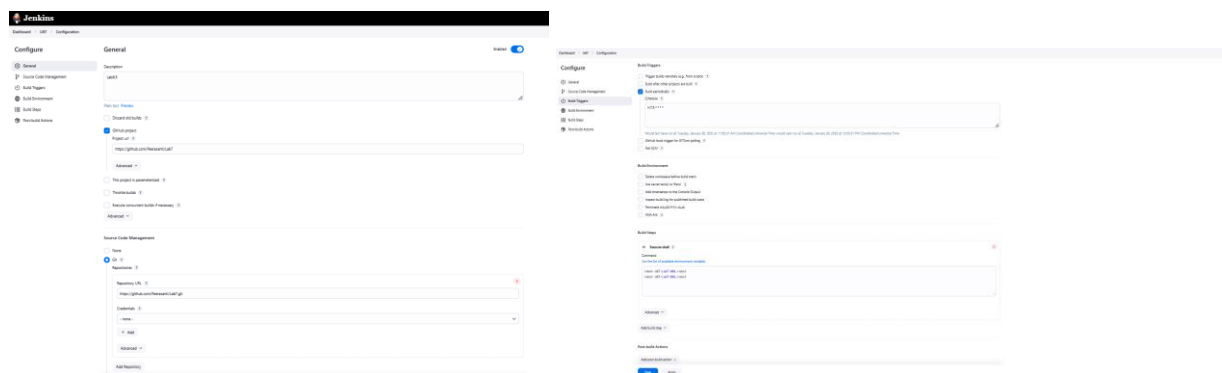
Description: Lab 8.5

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 ไปถึงไฟล์ให้เรียบร้อยแล้ว)

[Check point#14] Capture หน้าจอแสดงการตั้งค่า พร้อมกับตอบคำถามต่อไปนี้



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

ตอบ : robot UAT-Lab7-001.robot

robot UAT-Lab7-002.robot

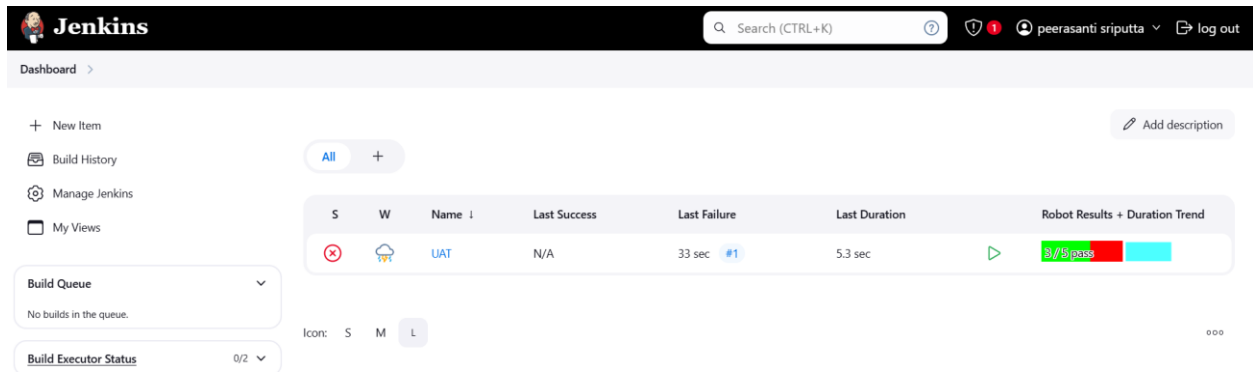
Post-build action: เพิ่ม Publish Robot Framework test results -> ระบุไดเรกทอรีที่เก็บไฟล์ผลการทดสอบโดย Robot framework ในรูป xml และ html -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ไม่ผ่าน แล้วนับว่าซอฟต์แวร์มีปัญหา -> ตั้งค่า Threshold เป็น % ของการทดสอบที่ผ่านแล้วนับว่าซอฟต์แวร์มีอยู่ในสถานะที่สามารถนำไปใช้งานได้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

13. กด Apply และ Save

14. สั่ง Build Now

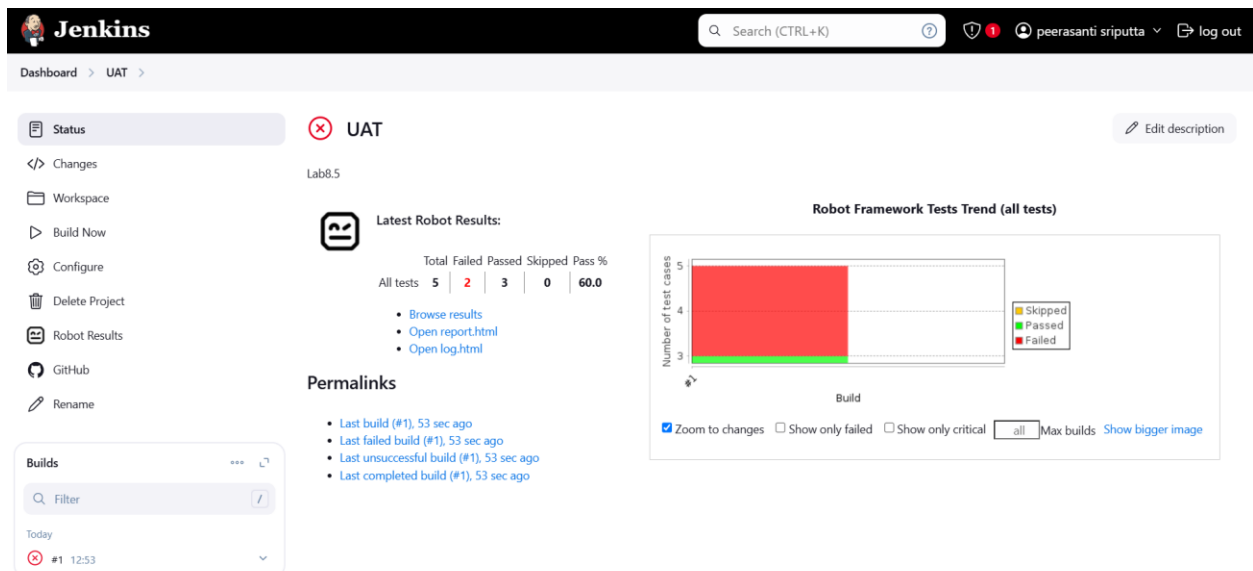
Lab Worksheet

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



The screenshot shows the Jenkins Dashboard. At the top, there's a search bar and user information. The left sidebar contains links like 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. The main area displays a table of builds. The table has columns: S (Status), W (Weather icon), Name, Last Success, Last Failure, Last Duration, and Robot Results + Duration Trend. One build is visible with status 'Failed' (red X), name 'UAT', and a duration of 5.3 sec. Below the table, there are sections for 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing '0/2').

S	W	Name	Last Success	Last Failure	Last Duration	Robot Results + Duration Trend
Failed	Cloudy	UAT	N/A	33 sec #1	5.3 sec	3/5 pass



The screenshot shows the Jenkins Pipeline view for 'UAT'. The left sidebar has links like 'Status', 'Changes', 'Workspace', 'Build Now', 'Configure', 'Delete Project', 'Robot Results', 'GitHub', and 'Rename'. The main area displays the 'UAT' pipeline status as 'Failed' (red X). Below this, there's a 'Latest Robot Results' section with a table showing test statistics: Total (5), Failed (2), Passed (3), Skipped (0), and Pass % (60.0). There are links to 'Browse results', 'Open report.html', and 'Open log.html'. To the right, there's a 'Robot Framework Tests Trend (all tests)' chart showing the number of test cases (Skipped, Passed, Failed) over builds. The chart shows 3 failed cases and 2 passed cases. Below the chart, there are checkboxes for 'Zoom to changes', 'Show only failed', and 'Show only critical', along with a 'Max builds' dropdown set to 'all'.

Latest Robot Results:

Total	Failed	Passed	Skipped	Pass %
5	2	3	0	60.0

Robot Framework Tests Trend (all tests)

Number of test cases: 5

Build: 1

Legend: Skipped (yellow), Passed (green), Failed (red)

Build 1: 3 Failed, 2 Passed