

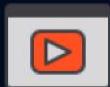
● LIVE

อบรมออนไลน์



ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins และ GitHub Actions ร่วมกับ n8n



เมวิดโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน



สอนสดผ่าน Zoom
รับจำนวนจำกัด

515
วัน
ชั่วโมงเต็ม

หมายสำคัญเริ่มต้น
เริ่มจาก 0 อย่างเข้าใจ



Samit Koyom
สถาบันไอทีเนียส

● LIVE

อบรมออนไลน์

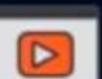
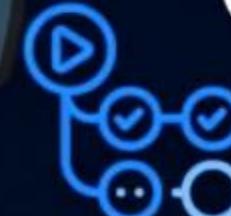


วัน
515
ชั่วโมงเต็ม

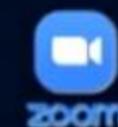
หมายสำหรับผู้เริ่มต้น
เริ่มจาก 0 อย่างเข้าใจ

ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins และ GitHub Actions ร่วมกับ n8n



มีวิดีโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน



สอนสดผ่าน Zoom
รับจำนำแຈักด้



Samit Koyom
สถาบันไอทีนานาชาติ



วิทยากร



อ.สา米ตร โภยม (ปาน)

ปริญญาโท คณะเทคโนโลยีและสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

▶ Frontend

Angular, React, Vue, Next, Nuxt, Bootstrap, Tailwind CSS

▶ Backend

PHP, Python, Java, Kotlin, Go, Rust, NodeJS, NestJS, .NET

▶ Database

MySQL, PostgreSQL, MS SQL Server, MongoDB, Supabase

▶ Mobile

Java, Kotlin, Objective C, Swift, KMP, Cordova, Flutter ,
React Native, Expo

▶ DevOps

Git, GitHub, Gitlab, Docker, Kubernetes, Jenkins, CI/CD



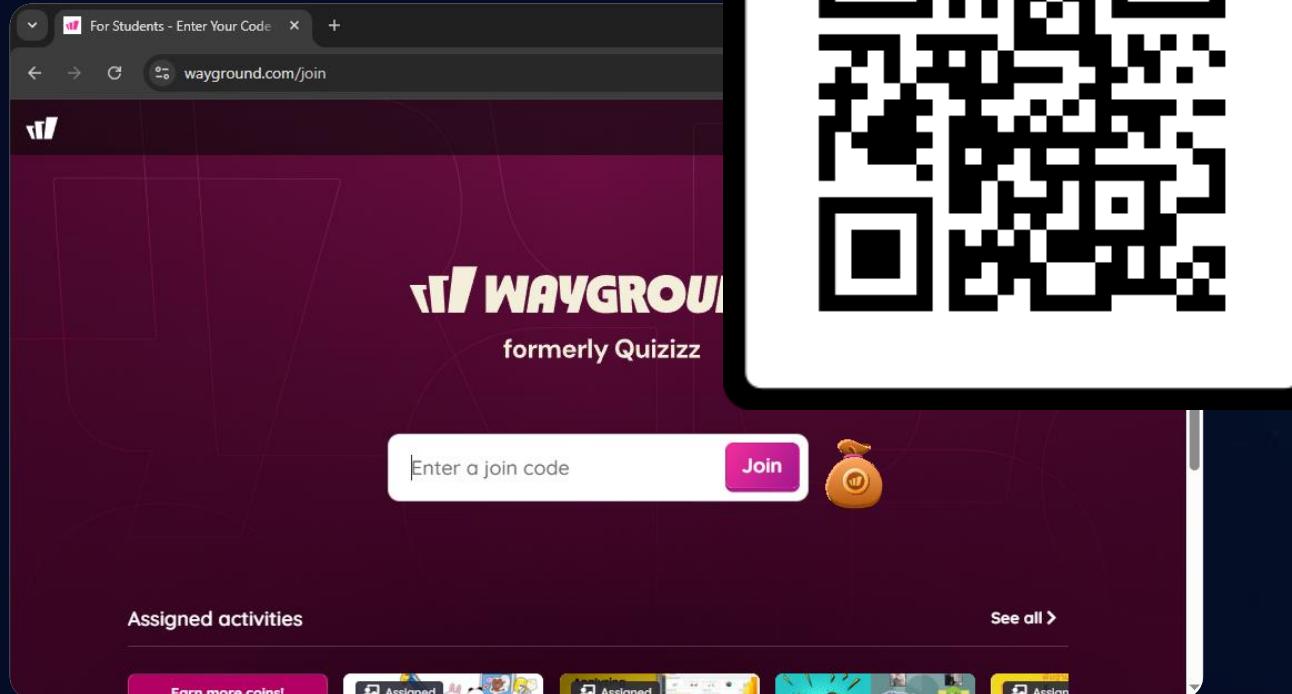
แบบทดสอบก่อนอบรม



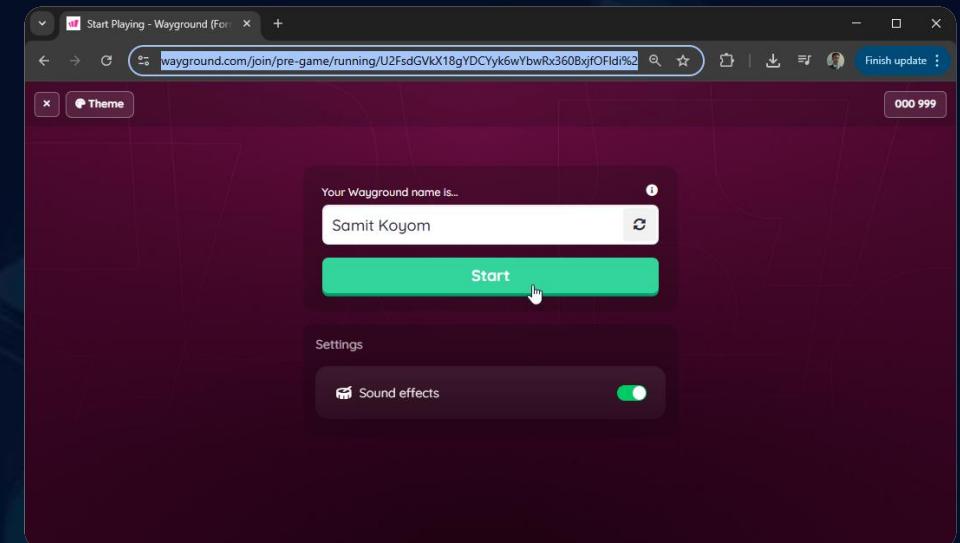
Pretest ทำแบบทดสอบก่อนเรียน

STEP 1: เข้าทำแบบทดสอบที่ลิงค์ ป้อนรหัสเข้าห้องสอบ

wayground.com/join



STEP 2: ป้อนชื่อ

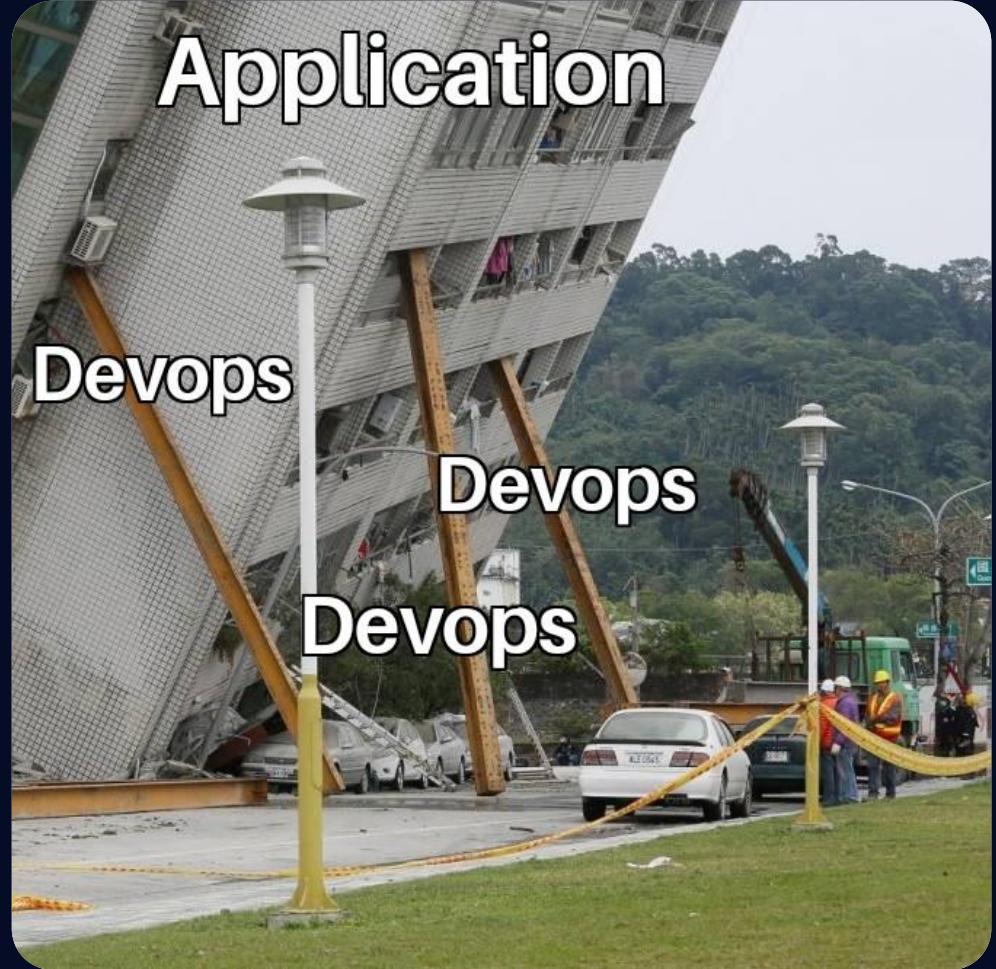


STEP 3: รอผู้สอน Start ข้อสอบ



สถาบันไอทีเกี้ยงส์

www.itgenius.co.th



“

Full Stack ให้ดี ต้องมี Dev
จะให้เทพ ต้องมี Ops ให้ครบสาย

ระบบเสร็จ งาน Deploy ต้องไม่ตาย
สร้าง Pipeline สุดอลัง พังตอนรัน

งาน DevOps เข้าเน้น "Flow" ไม่เน้น "Fix"
ไม่จุกจิก แค่ Config งานนวนเช้า

แค่ Push โค้ด เดียวมัน Test มัน Build เอง ได้ยาวๆ
จะ Fail now หรือ never เดียวเจอกัน

”



GIT CLONE TO PRODUCTION

มาหากกากา

#ผมว่าไม่គด





ดาวน์โหลดเอกสารประกอบการอบรม

bit.ly/devops_jenkins



สถาบันไอทีเกี้ยนส์

www.itgenius.co.th

● LIVE

อบรมออนไลน์

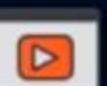
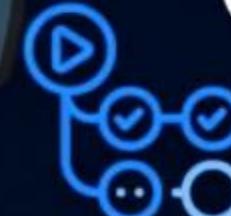


วัน
515
ชั่วโมงเต็ม

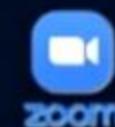
หมายสำหรับผู้เริ่มต้น
เริ่มจาก 0 อย่างเข้าใจ

ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins และ GitHub Actions ร่วมกับ n8n



มีวิดีโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน



สอนสดผ่าน Zoom
รับจำนำแຈักด้



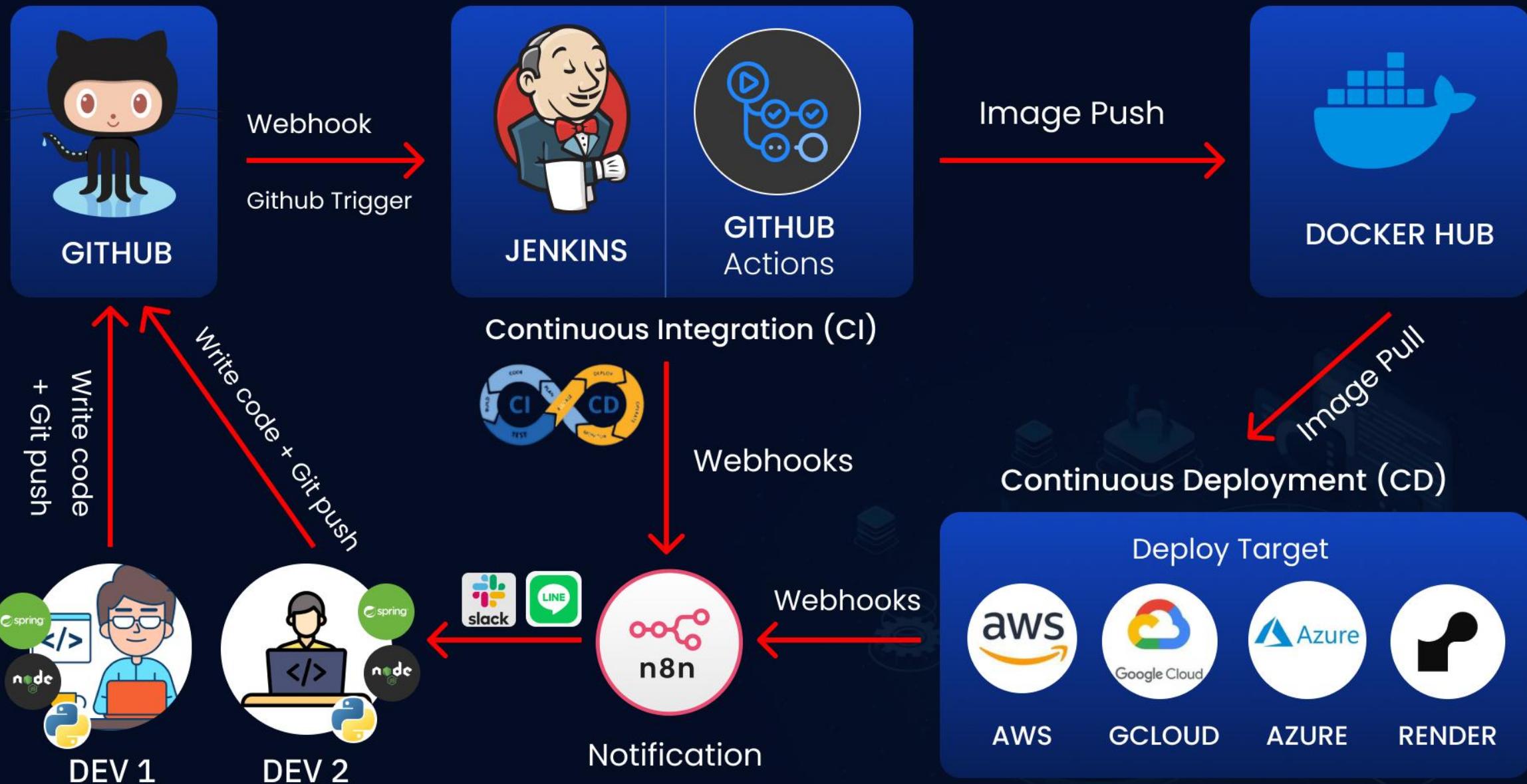
Samit Koyom
สถาบันไอทีนานาชาติ

Course Content



- | | | | |
|----|----------------------|----|--------------------------|
| 01 | Introduction | 06 | Jenkins workflow |
| 02 | Installation & Tools | 07 | Basic GitHub Actions |
| 03 | Basic Git | 08 | GitHub Action workflow |
| 04 | Basic Docker | 09 | N8N Notification |
| 05 | Basic Jenkins | 10 | Integration & Real-World |

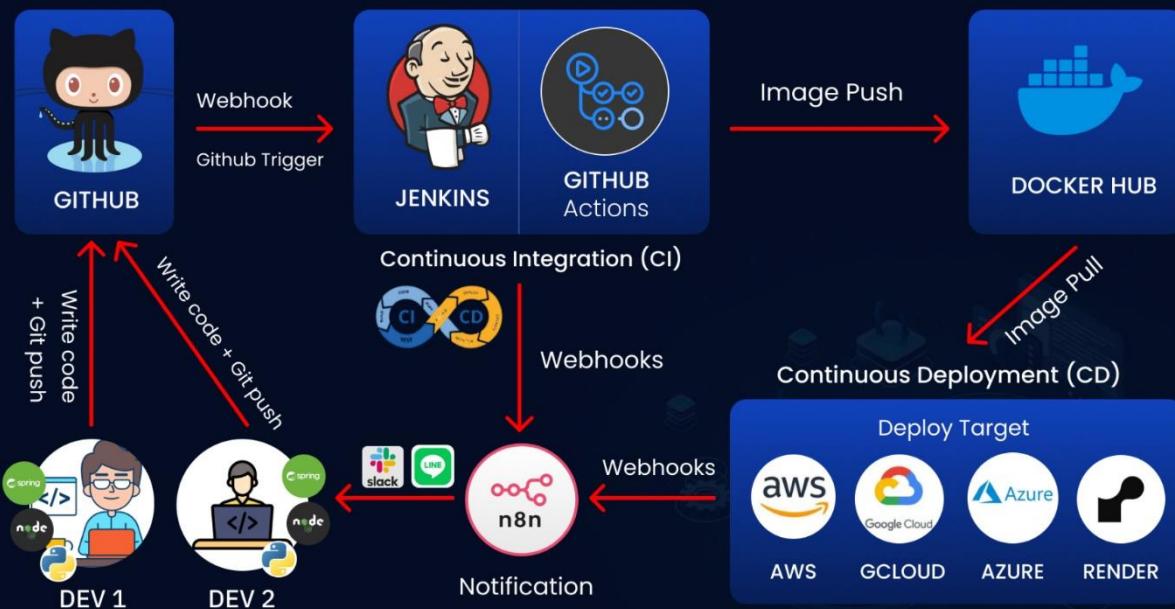




การพร้อมการทำ CI/CD

ใช้ Docker เป็นหัวใจหลักและเน้นการ Deploy ผ่าน Container

ในหลักสูตรนี้มีการใช้ทั้ง Jenkins และ Github Actions เพื่อให้เห็นความแตกต่าง และนำไปใช้ในการติดต่อระหว่างกัน รวมถึงการใช้เครื่องมือ Automation อย่าง N8N เพื่อประยุกต์ทำระบบแจ้งเตือนที่จัดทำขึ้น



Flow ของ Jenkins Pipeline

1. เริ่ม Pipeline ให้ agent ได้รับ
2. Checkout: ดึงซอฟต์แวร์มาสุดจาก Git (checkout scm)
3. Install & Test
4. Build Docker Image
5. Push Docker Image
6. Cleanup Docker
7. Deploy Local & Real Server
8. Post Success (เฉพาะขั้น Deploy Local)
9. Post Failure (ทั้ง Pipeline)
10. Deploy ไปเซิร์ฟเวอร์โดยผ่าน SSH พร้อม Webhook แจ้งผล

Screenshot of the Jenkins Pipeline Overview page for job #21:

Graph:

```
graph LR
    Start((Start)) --> CheckoutSCM[Checkout SCM]
    CheckoutSCM --> Checkout[Checkout]
    Checkout --> InstallTest[Install & Test]
    InstallTest --> BuildDockerImage[Build Docker Image]
    BuildDockerImage --> PushDockerImage[Push Docker Image]
    PushDockerImage --> CleanupDocker[Cleanup Docker]
    CleanupDocker --> DeployLocal[Deploy Local]
    DeployLocal --> End((End))
```

Step Details:

- Checkout SCM: 1.5s
- Checkout: 1.1s
- Install & Test: 17s
- Build Docker Image: 14s
- Push Docker Image: 21s
- Cleanup Docker: 1.3s
- Deploy Local: 10s

Logs:

- 1 C:\ProgramData\Jenkins\jenkins\workspace\express-docker-app\powershell -NoProfile -Command "\$body = [PSCustomObject]@{ project=\$env:JOB_NAME; stage='Deploy Local'; status='success'; build=\$env:BUILD_NUMBER; image=\$env:DOCKER_REPO + ':latest'; container=\$env:APP_NAME; url='http://localhost:3000/'; timestamp=(Get-Date -Format o) }; \$json = \$body | ConvertTo-Json; Invoke-RestMethod -Uri \$env:N8N_WEBHOOK_URL -Method Post -ContentType 'application/json' -Body \$json"
- 2
- 3

Flow ของ Github Action Pipeline

1. Trigger เริ่มงาน เรียกใช้งานแบบ Auto Push
2. กำหนดค่ารวมของ Workflow
3. Job: test (Install & Test)
4. Job: build_and_push (Build & Push Docker)
5. Job: smoke_test (Run & Curl)
6. Deploy ไปเซิร์ฟเวอร์ระยะไกลผ่าน SSH

The screenshot shows the GitHub Actions interface for the 'CI/CD - Express Docker App' workflow. It displays a summary of the most recent run, which was triggered manually 4 minutes ago, completed successfully in 2m 56s, and produced 1 artifact. The main workflow file, 'main.yml', is shown with a matrix step for 'Install & Test (Node 22.x)' and a subsequent 'Build & Push Docker' step. Both steps are marked as completed. Below this, there is a 'Docker Build summary' section with a download link for the build record archive and a table showing the details of the Docker build.

ID	Name	Status	Cached	Duration
29C058	express-docker-app (production)	completed	0%	1m5s

Push Image to Docker Hub

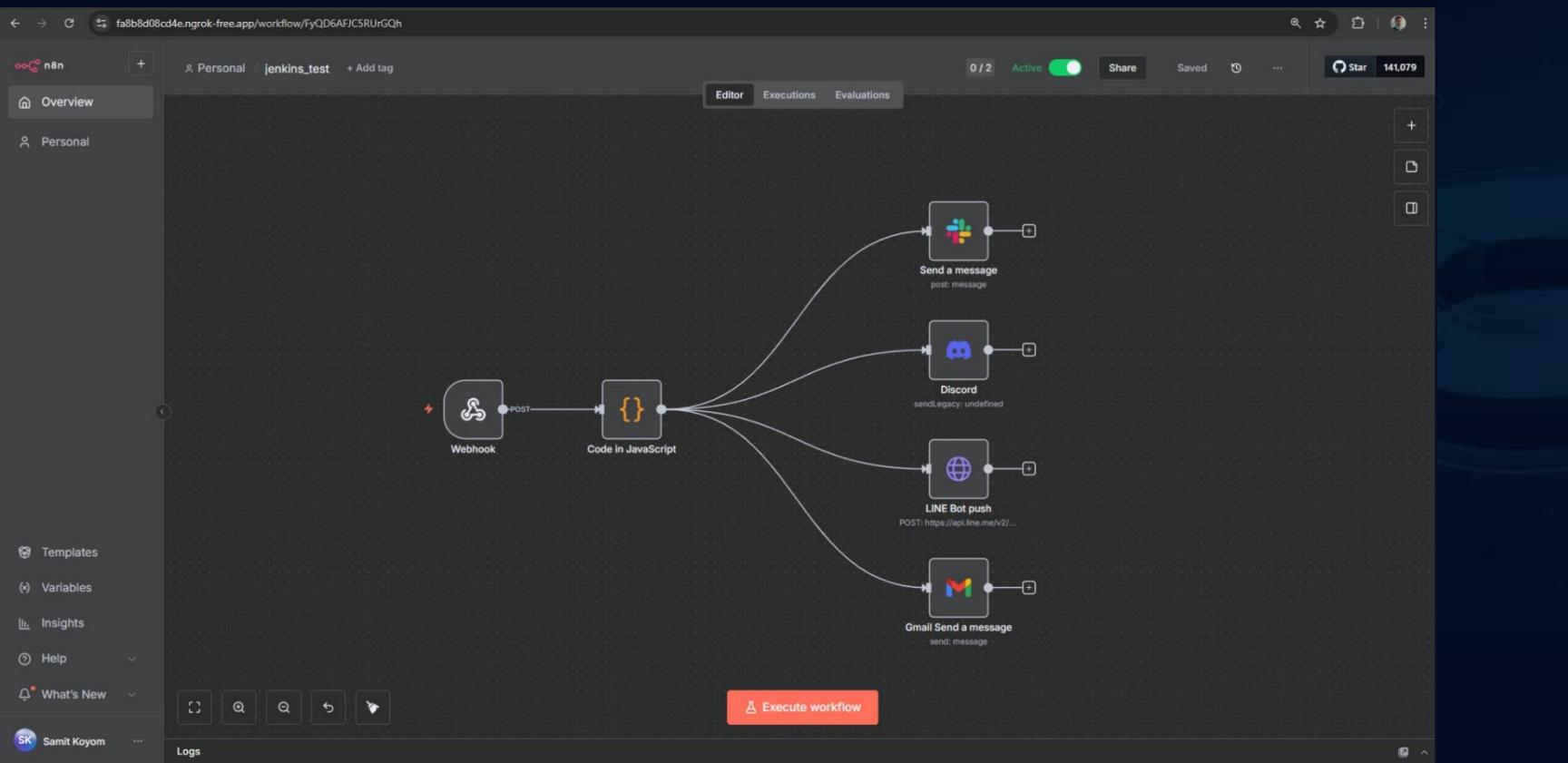
The screenshot shows the Docker Hub repository page for 'iamsamitdev/express-docker-app'. The repository was last pushed 2 minutes ago and has a size of 133.9 MB. The 'Tags' tab is selected, showing two tags: 'sha-5a86497' and 'latest'. For the 'sha-5a86497' tag, there is a command provided to pull it: 'docker pull iamsamitdev/express-docker-app:sha-5a86497'. Below the tags, there are sections for Digest, OS/ARCH, Last pull, and Compressed size. Similar sections are shown for the 'latest' tag.

Digest	OS/ARCH	Last pull	Compressed size
f70c93007659	linux/amd64	less than 1 day	55.35 MB



Webhook to n8n for Notification

- แจ้งเตือนผ่าน Slack
- แจ้งเตือนผ่าน Discord
- แจ้งเตือนผ่าน Line
- แจ้งเตือนผ่าน Email
- และอื่นๆ ตามต้องการ



Workshop การเขียน Pipeline
เพื่อ Deploy กัน 3 ภาษา 3 framework ยอดนิยม



Express Docker Application



โปรเจกต์ Express.js + TypeScript REST API ที่ทำงานใน Docker Container พร้อมด้วย CI/CD Pipeline ผ่าน Jenkins, Github Actions และการแจ้งเตือนผ่าน N8N

Flask Docker Application



โปรเจกต์ Flask API แบบ RESTful ที่ทำงานใน Docker Container พร้อมด้วย CI/CD Pipeline ผ่าน Jenkins, Github Actions และการแจ้งเตือนผ่าน N8N

Spring Boot Docker Application

โปรเจกต์ Spring Boot REST API ที่ทำงานใน Docker Container พร้อมด้วย CI/CD Pipeline ผ่าน Jenkins, GitHub Actions และการแจ้งเตือนผ่าน N8N



ตัวอย่าง Workshop Node.js Express Pipeline



Features

- ✓ **Express.js REST API** - Fast, minimal Node.js web framework
- ◆ **TypeScript Support** - Type-safe JavaScript development
- ◆ **Docker Containerization** - Lightweight Alpine-based container
- ◆ **Jest Testing** - Comprehensive test suite with Supertest
- ◆ **CI/CD Pipeline** - Automated Jenkins pipeline with deployment
- ◆ **GitHub Actions** - Modern CI/CD with GitHub-native automation
- ◆ **Test Coverage** - Code coverage reports and analysis
- ◆ **Docker Hub Integration** - Automated image publishing
- ◆ **Server Deployment** - Automated deployment to remote servers
- ◆ **Semantic Versioning** - Build number based tagging
- ◆ **Hot Reload** - Development with ts-node

Project Structure

```
express-docker-app/
├── src/
│   └── app.ts          # Main Express application (TypeScript)
├── tests/
│   └── app.test.ts     # Jest test suite with Supertest
├── dist/
│   └── app.js          # Compiled JavaScript output
├── node_modules/
└── .github/
    ├── workflows/
    │   └── main.yml      # GitHub Actions workflow
    ├── Dockerfile        # Docker build configuration
    ├── Jenkinsfile       # Jenkins CI/CD pipeline
    ├── jest.config.js    # Jest testing configuration
    ├── package.json       # Node.js project configuration
    ├── package-lock.json # Dependency lock file
    ├── tsconfig.json     # TypeScript configuration
    └── README.md         # Project documentation
```

ตัวอย่าง Workshop Python Flask Pipeline



Features

- ✓ **Flask REST API** - Simple Hello World API endpoint
- ◆ **Docker Support** - Containerized application for easy deployment
- ◆ **Unit Testing** - Comprehensive test suite with pytest
- ◆ **CI/CD Pipeline** - Automated Jenkins pipeline
- ◆ **GitHub Actions** - Modern CI/CD with GitHub-native automation
- ◆ **Test Reports** - JUnit XML test result reporting
- ◆ **Docker Hub Integration** - Automated image publishing
- ◆ **N8N Notifications** - Webhook-based notifications
- ◆ **Semantic Versioning** - Build number based tagging

Project Structure

```
flask-docker-app/
├── tests/           # Test directory
│   ├── __init__.py   # Test package initializer
│   ├── conftest.py    # Pytest configuration
│   └── test_app.py    # Application tests
├── __pycache__/      # Python bytecode cache
├── .github/
│   └── workflows/    # GitHub Actions workflow
│       └── main.yml
├── app.py            # Main Flask application
├── Dockerfile         # Docker build configuration
├── Jenkinsfile        # Jenkins CI/CD pipeline
├── pytest.ini         # Pytest configuration
├── requirements.txt   # Python dependencies
├── Readme.md          # Project documentation
└── .gitignore         # Git ignore rules
```

Prerequisites

- **Python 3.13+**
- **Docker & Docker Compose**
- **Ci/CD Pipeline**

ตัวอย่าง Workshop Java SpringBoot Pipeline



Features

- ✓ **Spring Boot REST API** - RESTful API endpoints with comprehensive functionality
- ◆ **Multi-stage Docker Build** - Optimized Docker image with build and runtime stages
- ◆ **Java 21** - Latest LTS Java version support
- ◆ **Comprehensive Testing** - Unit tests, integration tests, and test coverage
- ◆ **CI/CD Pipeline** - Automated Jenkins pipeline with Maven
- ◆ **GitHub Actions** - Modern CI/CD with GitHub-native automation
- ◆ **JaCoCo Coverage** - Code coverage reports and analysis
- ◆ **Docker Hub Integration** - Automated image publishing
- ◆ **N8N Notifications** - Webhook-based notifications
- ◆ **Semantic Versioning** - Build number based tagging
- ◆ **Health Check** - Built-in health monitoring endpoint

Project Structure

```
springboot-docker-app/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/example/demo/
│       │       ├── DemoApplication.java      # Main Spring Boot application
│       │       └── controller/
│       │           └── HelloController.java  # REST API controller
│       └── model/
│           └── GreetingRequest.java      # Request/Response model
├── resources/
│   ├── application.properties      # Application configuration
│   ├── static/
│   └── templates/                 # Template files
└── test/
```

โปรเจกต์ทำงานบน Docker Container ทั้งหมด

```

Dockerfile X
express-docker-app > Dockerfile > ...
You, 45 minutes ago | 1 author (You)
1 # ใช้ Official Node.js image (เวอร์ชัน Long Term Support) เป็น base image
2 FROM node:22-alpine
3
4 # กำหนด Working Directory ภายใน Container
5 WORKDIR /app
6
7 # Copy ไฟล์ package.json และ package-lock.json เข้าไปเก็บ
8 # เพื่อใช้ร่วมกับ Docker cache layer ที่เราเพิ่งตั้ง install dependencies ในไฟล์ dockerfile ที่แยกตัว
9 COPY package*.json .
10
11 # ติดตั้ง Dependencies
12 RUN npm install
13
14 # Copy ไฟล์ทั้งหมดในโปรเจกต์เข้าไปใน container
15 COPY .
16
17 # กำหนด Port ที่ Container จะทำงาน
18 EXPOSE 3000
19
20 # ค่าสีสำหรับ Express Application
21 CMD ["npm", "start:ts"]

Dockerfile X
flask-docker-app > Dockerfile > ...
You, 47 minutes ago | 1 author (You)
1 # ใช้ Official Python image เป็น base image
2 FROM python:3.13-slim
3
4 # กำหนด Working Directory ภายใน Container
5 WORKDIR /app
6
7 # Copy ไฟล์ requirements.txt เข้าไปเก็บ เพื่อใช้ cache layer ของ Docker
8 COPY requirements.txt .
9
10 # ติดตั้ง Dependencies ที่ระบุไว้
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # Copy ไฟล์ทั้งหมดในโปรเจกต์เข้าไปใน container
14 COPY .
15
16 # กำหนด Port ที่ Container จะทำงาน
17 EXPOSE 5000
18
19 # ค่าสีสำหรับ Flask Application
20 CMD ["python", "app.py"]

Dockerfile X
springboot-docker-app > Dockerfile > ...
You, 52 minutes ago | 1 author (You)
1 # STAGE 1: Build Stage - ใช้ Maven Image เพื่อ Build โปรเจกต์
2 FROM maven:3.8.5-openjdk-21 AS build
3 WORKDIR /app
4 COPY pom.xml .
5 RUN mvn dependency:go-offline
6 COPY src ./src
7 RUN mvn package -DskipTests
8
9 # STAGE 2: Run Stage - ใช้ JRE Image ขนาดเล็กเพื่อ Run ய่อย
10 FROM eclipse-temurin:21-jre-focal
11 WORKDIR /app
12 COPY --from=build /app/target/*.jar app.jar
13 EXPOSE 8080
14 ENTRYPOINT ["java", "-jar", "app.jar"]

```

เขียน Jenkinsfile ทำ Pipeline แยกแต่ละโปรเจกต์

```

Jenkinsfile X
express-docker-app > Jenkinsfile
You, 47 minutes ago | 1 author (You)
1 pipeline {
2   agent any
3
4   environment {
5     DOCKER_HUB_CREDENTIALS = credentials ('dockerhub-cred')
6     DOCKER_REPO = "your-dockerhub-username/express-docker-app"
7     APP_NAME = "express-docker-app"
8     DEPLOY_SERVER = "user@your-server-ip"
9   }
10
11   stages {
12     stage('Checkout') {
13       steps {
14         git branch: 'main', url: 'https://github.com/your-repo/express-docker-app.git'
15       }
16     }
17
18     stage('Install & Test') {
19       steps {
20         sh ...
21         npm install
22         npm test
23       }
24     }
25
26     stage('Build Docker Image') {
27       steps {
28         script {
29           dockerImage = docker.build (
30             "${DOCKER_REPO}#${BUILD_NUMBER}"
31           )
32         }
33       }
34     }
35
36     stage('Push Docker Image') {
37       steps {
38         script {
39           docker.withRegistry (
40             'https://index.docker.io/v1/',
41             "${DOCKER_HUB_CREDENTIALS}"
42           ) {
43             dockerImage.push()
44             dockerImage.push ("latest")
45           }
46         }
47       }
48     }
49
50     stage('Deploy to Server') {
51       steps {
52         sshagent (
53           ['deploy-server-cred']
54         ) {
55           sh """
56             You, 48 minutes
57           """
58         }
59       }
60     }
61   }
62 }

```

เขียน Github Actions ทำ Pipeline แยกแต่ละโปรเจกต์

```

main.yml X
express-docker-app > _github > workflows > main.yml
You, 51 minutes ago | 1 author (You)
1 # ชื่อของ Workflow ที่จะแสดงในหน้า Actions ของ GitHub
2 name: CI/CD - Express Docker App
3
4 # กำหนด Trigger: ให้ Workflow นี้ทำงานทุกครั้งที่มีการ push ไปยัง branch 'main'
5 on:
6   push:
7     branches: [ "main" ]
8
9 # กำหนด Jobs ที่จะให้ทำงาน
10 jobs:
11   build-and-push:
12     # กำหนดสภาพแวดล้อมที่จะรัน Job นี้ (ใช้ Ubuntu ของ GitHub)
13     runs-on: ubuntu-latest
14
15     # กำหนดขั้นตอนการทำงาน (Steps)
16     steps:
17       # 1. Checkout Code: ลีดโค้ดจาก Repository ลงมาที่ Runner
18       - name: Checkout repository
19         uses: actions/checkout@v4
20
21       # 2. Login to Docker Hub: ล็อกอินเข้า Docker Hub โดยใช้ Secrets
22       - name: Log in to Docker Hub
23         uses: docker/login-action@v3
24         with:
25           username: ${{ secrets.DOCKERHUB_USERNAME }}
26           password: ${{ secrets.DOCKERHUB_TOKEN }}
27
28       # 3. Build and Push Docker Image: สร้างและ Push Image
29       - name: Build and push Docker image
30         uses: docker/build-push-action@v5
31         with:
32           context: .
33           push: true
34           tags: ${{ secrets.DOCKERHUB_USERNAME }}/
35           my-express-api:latest, ${{ secrets.DOCKERHUB_USERNAME }}}/my-express-api:${{ secrets.github.sha }}
36
37       # 4. Notify N8N on Success: แจ้งเตือน N8N เมื่อสำเร็จ
38       - name: Notify N8N on Success
39         if: success() # ทำงานเมื่อ Step ก่อนหน้าทั้งหมดสำเร็จ
40         run:
41           curl -X POST -H "Content-Type: application/json" \
42             -d '{
43               "status": "SUCCESS",
44               "project": "${{ github.repository }}"
45             }',
46             You, 51 minutes

```

● LIVE

อบรมออนไลน์



ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins และ GitHub Actions ร่วมกับ n8n



มีวิดีโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน

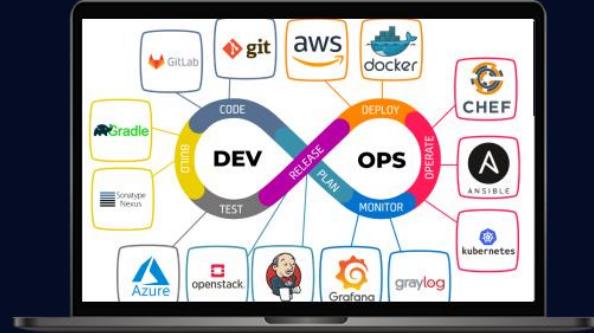


สอนสดผ่าน Zoom
รับจำนวนจำกัด



Samit Koyom
สถาบันไอทีจีเนียส

Day 1



1. พื้นฐานและการติดตั้งเครื่องมือ DevOps CI/CD
2. พื้นฐาน Git



Jenkins
และ GitHub Actions
ร่วมกับ n8n

ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins

และ GitHub

Actions

ร่วมกับ **n8n**



มีวิดีโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน



สถาบันไอทีจีเนียส

วัน
5 15
ซั่วโมงเต็ม



การเตรียม
เครื่องมือ



Samit Koyom
สถาบันไอทีจีเนียส

โปรแกรม (Tool and Editor) ที่ใช้อบรม

1. Visual Studio Code
2. Java JDK 21.x
3. Node.js 22.x
4. Python 3.1x
5. Docker Desktop
6. Git

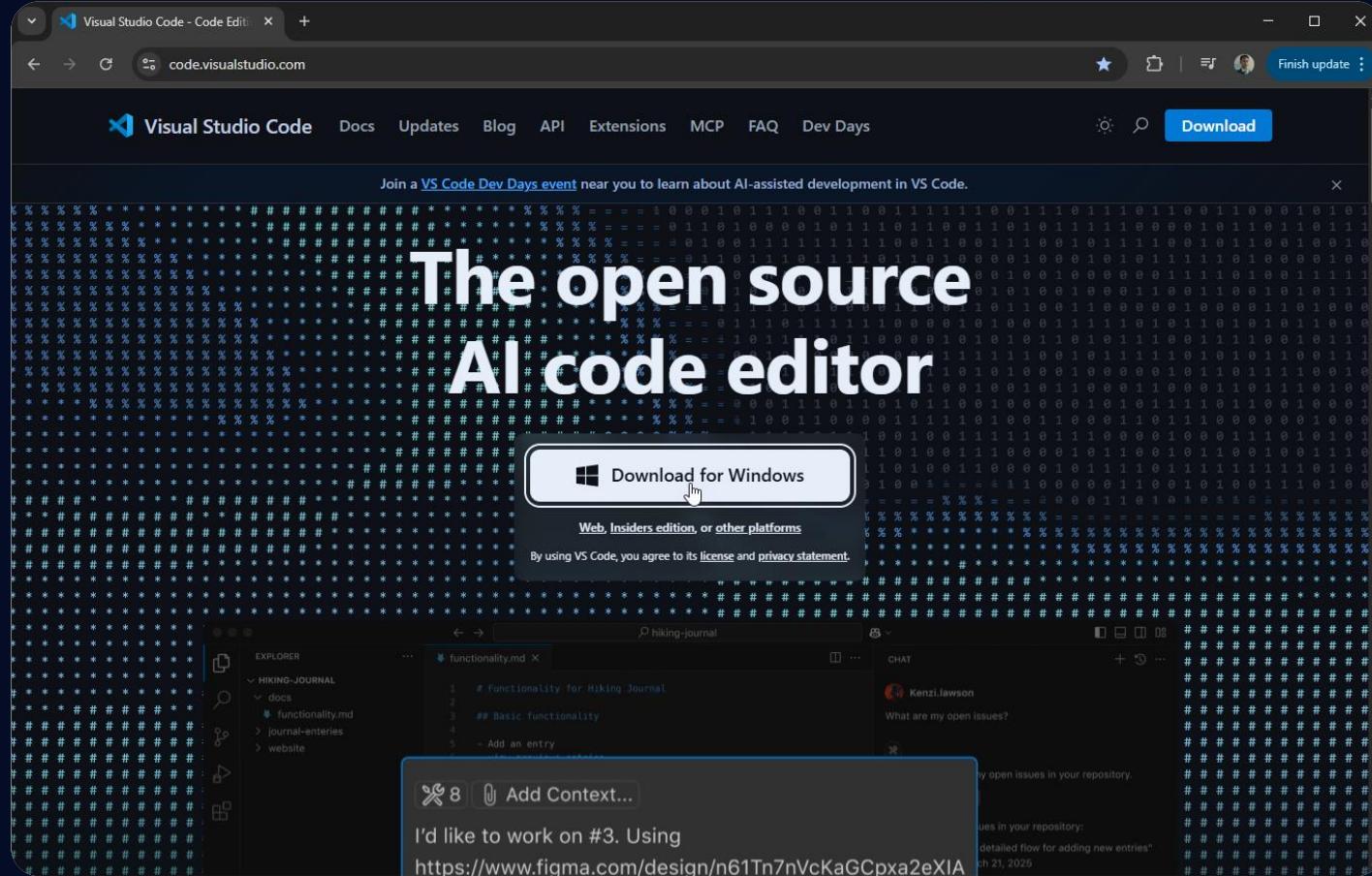




1. ติดตั้ง Visual Studio Code



ຕັດຕັ້ງ Visual Studio Code ວຽກເສຍເວັບໄຊທີ່ຈຳເປັນ



ເຂົ້າໄປດາວໂຫລດ Visual Studio Code ໄດ້ກີ່ <https://code.visualstudio.com>



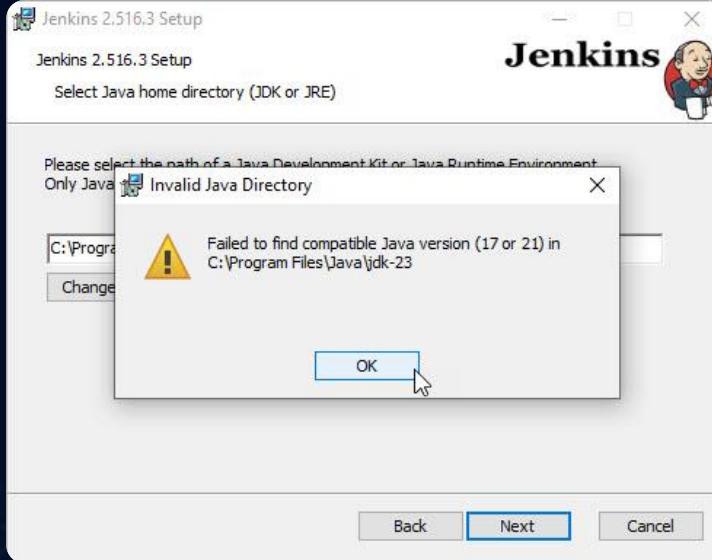
การติดตั้งส่วนเสริม (Extension) ของ Visual Studio Code



รายชื่อ Extensions ที่แนะนำสำหรับ VS Code

- 1. Material Icon Theme** by Philipp Kief
- 2. Python** by Microsoft
- 3. Language Support for Java(TM)** by Red Hat by Red Hat
- 4. Docker** by Microsoft
- 5. Jenkins** by p1c2u
- 6. GitHub Actions** by Github
- 7. One Dark Pro** by binaryify





2. ติดตั้ง Java JDK 21.x



หมายเหตุ ในคอร์สนี้ Jenkins ต้องการ Java JDK **17** หรือ **21** เท่านั้น (เก่าหรือใหม่กว่านี้ไม่รองรับ)



The screenshot shows the Oracle Java Downloads page at oracle.com/java/technologies/downloads/#jdk21-windows. The 'Windows' tab is selected, highlighted with a red box. A large orange button labeled 'Windows เลือกดังนี้' is positioned to the right of the download links. The table lists three download options:

Product/file description	File size	Download
x64 Compressed Archive	186.05 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	164.42 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256)
x64 MSI Installer	163.16 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)

Below the table, there is a 'Documentation Download' button and a 'Release information' section with links to 'Online Documentation' and 'Installation Instructions'. A URL link is also present at the bottom of the section.

ดาวน์โหลด Java JDK 21 ได้ที่ <https://www.oracle.com/java/technologies/downloads/#jdk21-windows>



The screenshot shows the Oracle Java Downloads page for macOS. The URL in the address bar is oracle.com/java/technologies/downloads/#jdk21-mac. The page title is "Java SE Development Kit 21.0.8 downloads". It states that JDK 21 binaries are free to use in production and redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions \(NFTC\)](#). It also mentions that JDK 21 will receive updates under the NFTC until September 2026, after which updates will be licensed under the [Java SE OTN License \(OTN\)](#).

The "macOS" tab is selected, highlighted with a red box. Other tabs available are "Linux" and "Win". An orange box highlights the "MacOS เลือกดังนี้" section.

Product/file description	File size	Download
ARM64 Compressed Archive	181.46 MB	https://download.oracle.com/java/21/latest/jdk-21_macos-aarch64_bin.tar.gz (sha256)
ARM64 DMG Installer	180.79 MB	https://download.oracle.com/java/21/latest/jdk-21_macos-aarch64_bin.dmg (sha256)
x64 Compressed Archive	183.65 MB	https://download.oracle.com/java/21/latest/jdk-21_macos-x64_bin.tar.gz (sha256)
x64 DMG Installer	183.00 MB	https://download.oracle.com/java/21/latest/jdk-21_macos-x64_bin.dmg (sha256)

Buttons at the bottom include "Documentation Download" and "Release information".

Annotations on the right side highlight the download links for "MacOS CPU Arm" and "MacOS CPU Intel" with orange boxes.

ดาวน์โหลด Java JDK 21 ได้ที่ <https://www.oracle.com/java/technologies/downloads/#jdk21-mac>



The screenshot shows the Oracle Java Downloads page for JDK 21 on a Linux system. The 'Linux' tab is selected, highlighted with a red border. The page lists five download options:

Product/file description	File size	Download
ARM64 Compressed Archive	186.07 MB	https://download.oracle.com/java/21/latest/jdk-21_linux-aarch64_bin.tar.gz (sha256)
ARM64 RPM Package	185.76 MB	https://download.oracle.com/java/21/latest/jdk-21_linux-aarch64_bin.rpm (sha256) (OL 9 GPG Key)
x64 Compressed Archive	187.89 MB	https://download.oracle.com/java/21/latest/jdk-21_linux-x64_bin.tar.gz (sha256)
x64 Debian Package	159.73 MB	https://download.oracle.com/java/21/latest/jdk-21_linux-x64_bin.deb (sha256)
x64 RPM Package	187.55 MB	https://download.oracle.com/java/21/latest/jdk-21_linux-x64_bin.rpm (sha256) (OL 9 GPG Key)

Three download links are highlighted with orange boxes and red borders: the ARM64 RPM Package, the x64 Compressed Archive, and the x64 RPM Package.

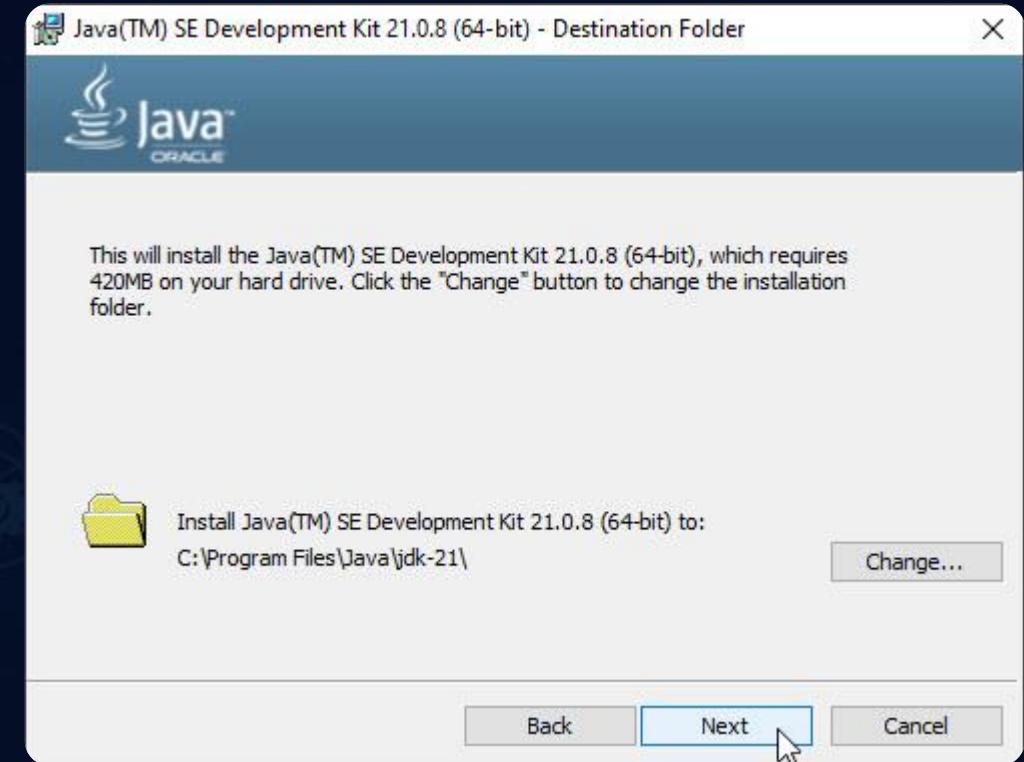
ดาวน์โหลด Java JDK 21 ได้ที่ <https://www.oracle.com/java/technologies/downloads/#jdk21-linux>



หลังดาวน์โหลดไฟล์ทำการคลิกเริ่มติดตั้งกัน



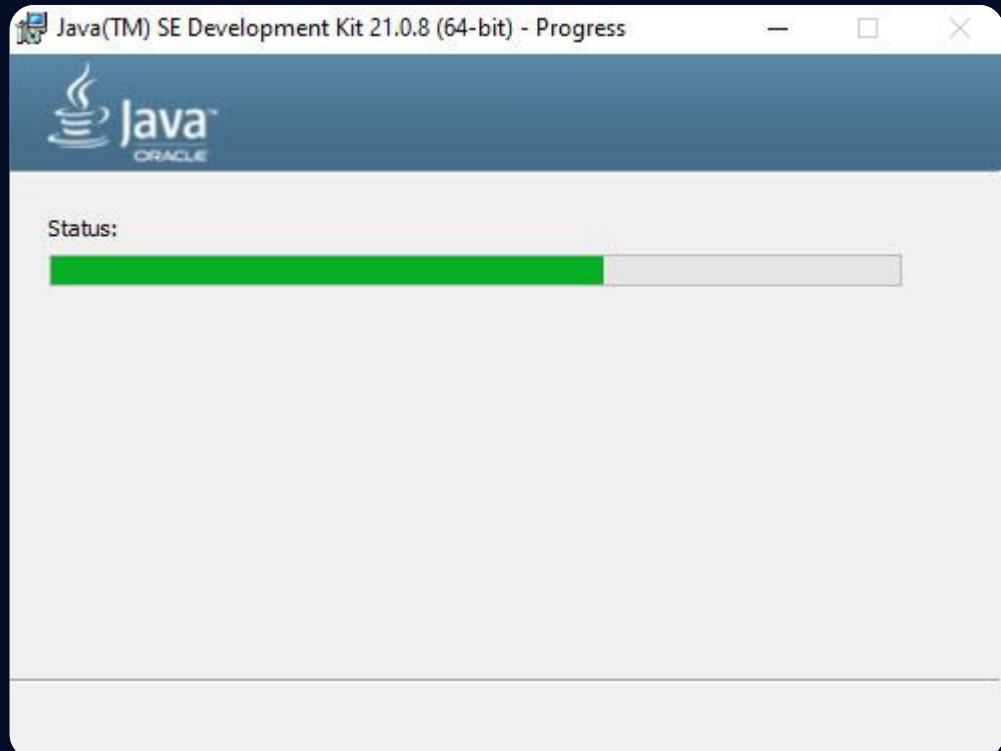
คลิก Next



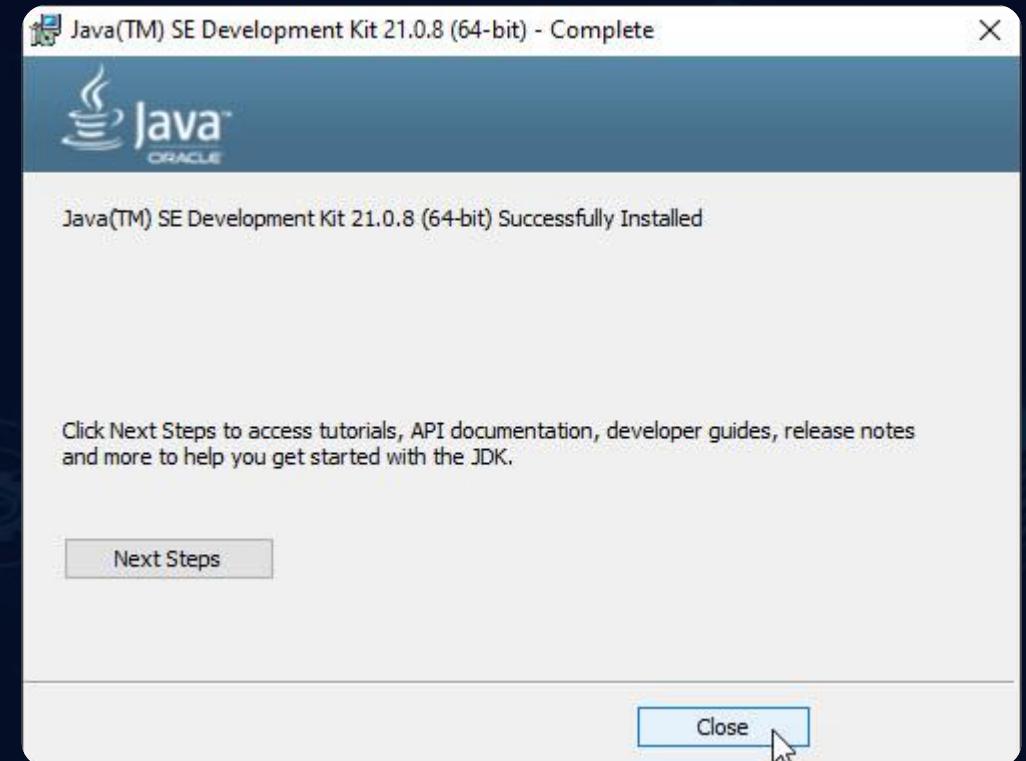
บน Windows จะติดตั้งไว้ตาม path นี้ คลิก Next



รอการติดตั้งจบแล้วเสร็จ...



รอการติดตั้ง



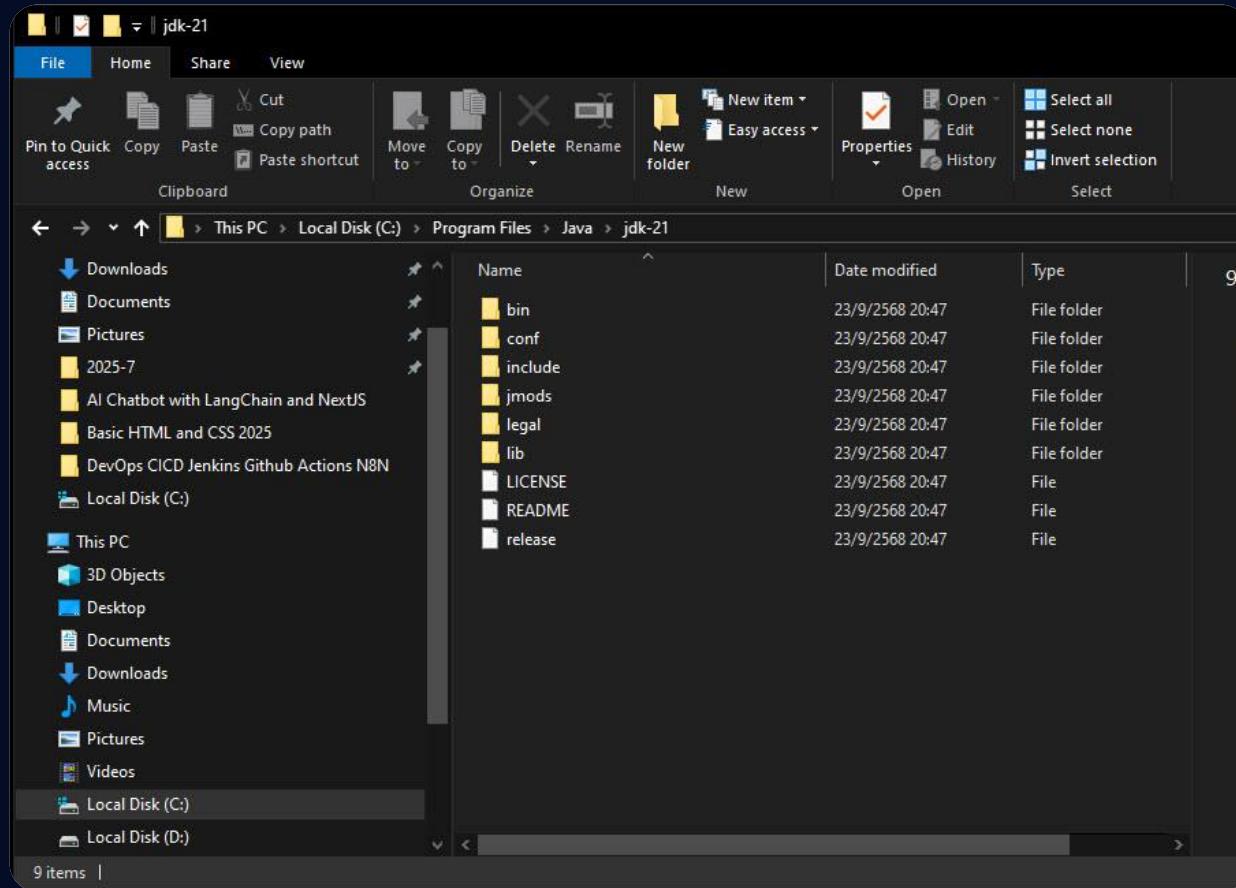
ติดตั้งเสร็จแล้ว คลิก Close



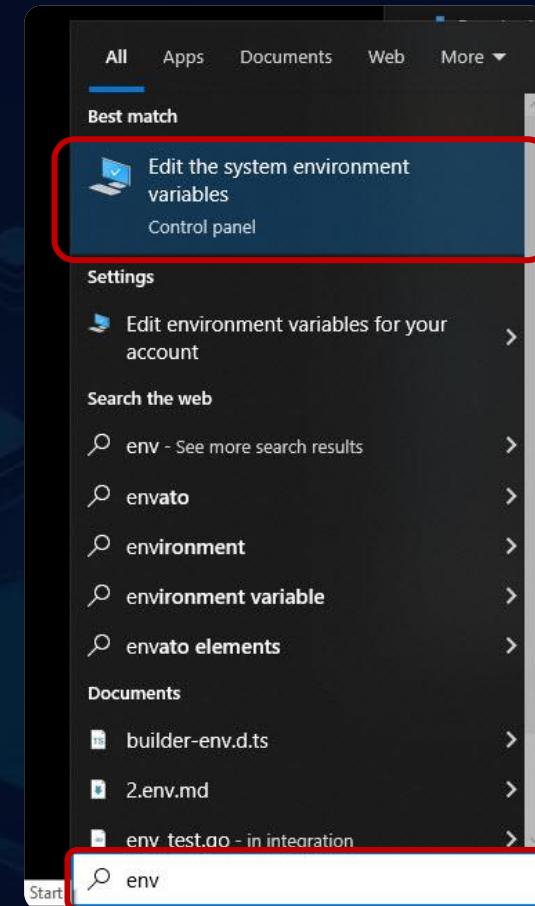
สถาบันไอทีเกี้ยนส์

www.itgenius.co.th

หลังติดตั้ง Java JDK เสร็จเรามากำหนด path ให้ Windows เรียกใช้งานได้



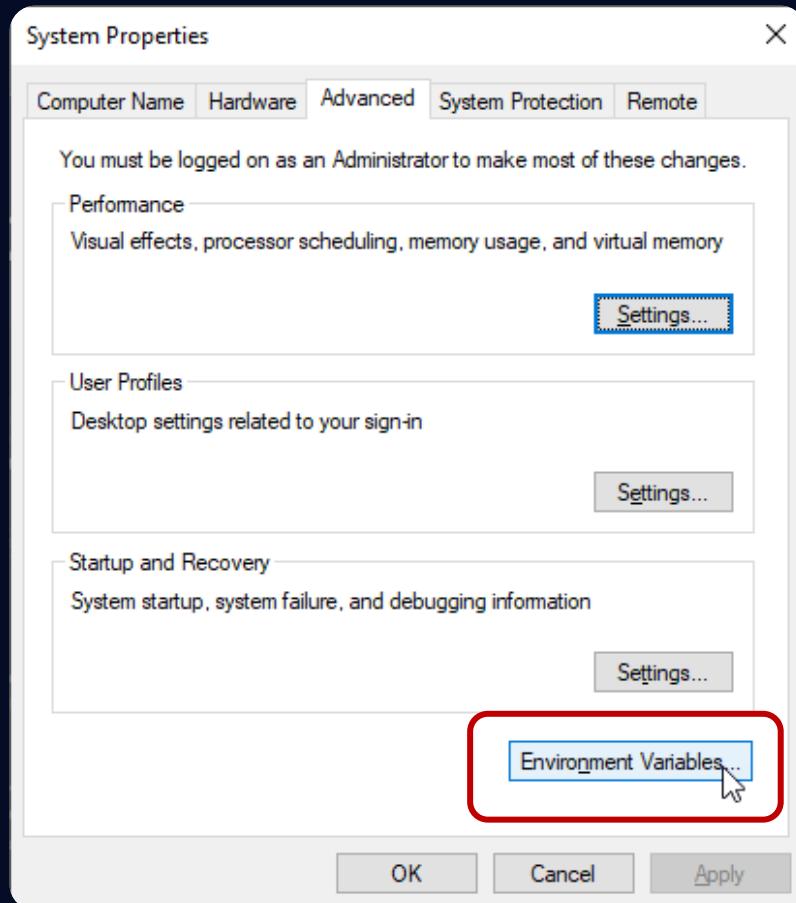
Windows เข้าไปเช็ค path ที่ C:\Program Files\Java\jdk-21
(คัดลอก path นี้ไว้)



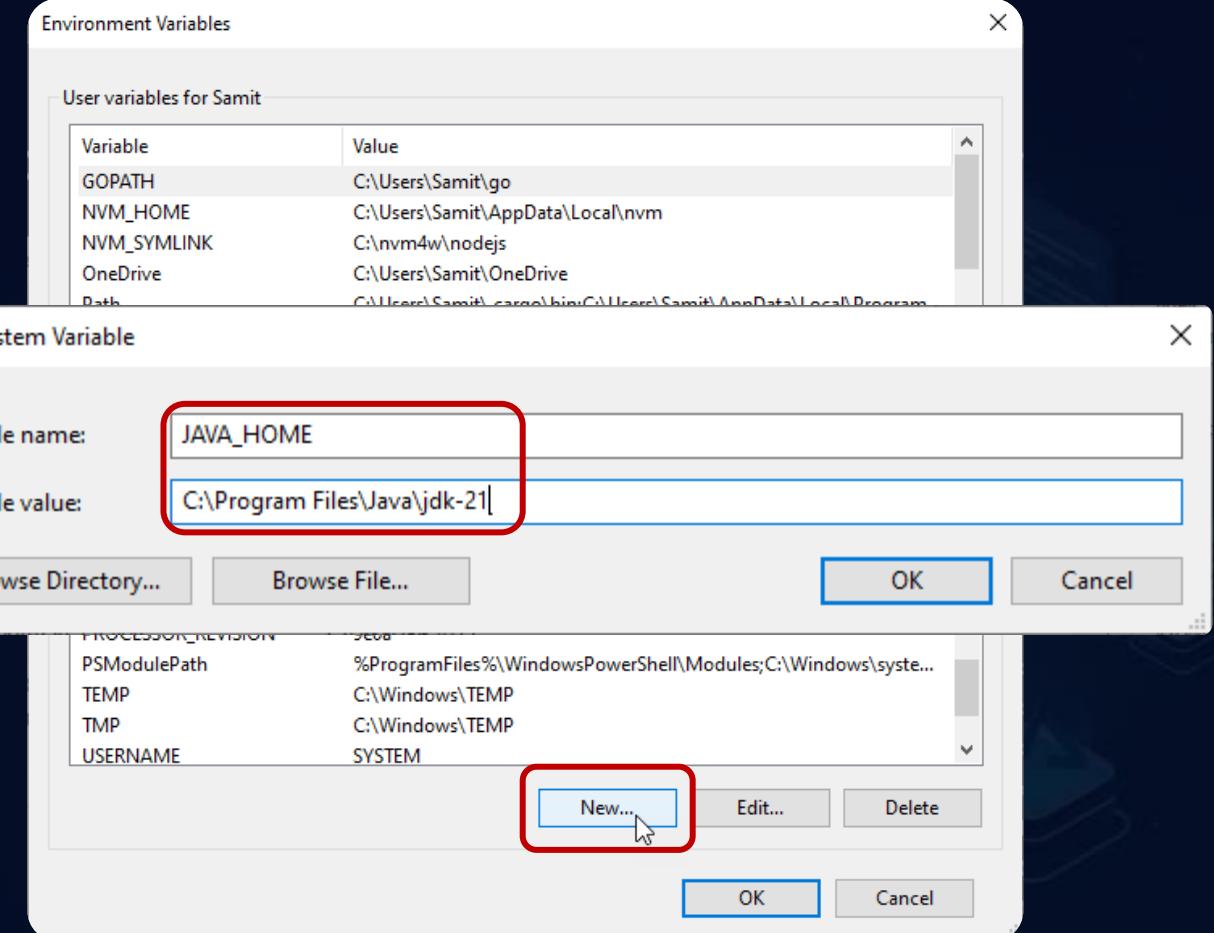
คลิก start ของ windows ค้นหา “env”
แล้วคลิกเข้าไปตั้งค่าดังรูป



หลังติดตั้ง Java JDK เสร็จเรามากำหนด path ให้ Windows เรียกใช้งานได้



คลิกที่ Environment Variables...



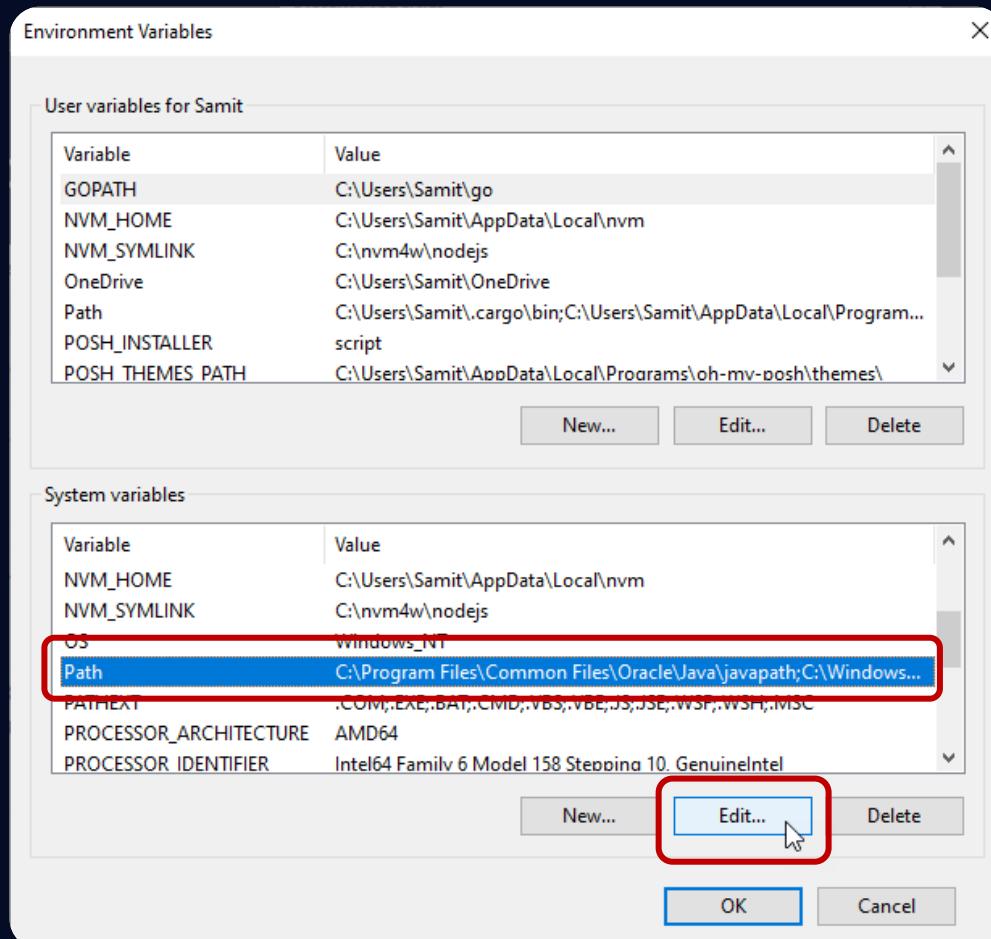
คลิกที่ปุ่ม New... ดังภาพ และกำหนดค่าดังนี้

Variable name: **JAVA_HOME**

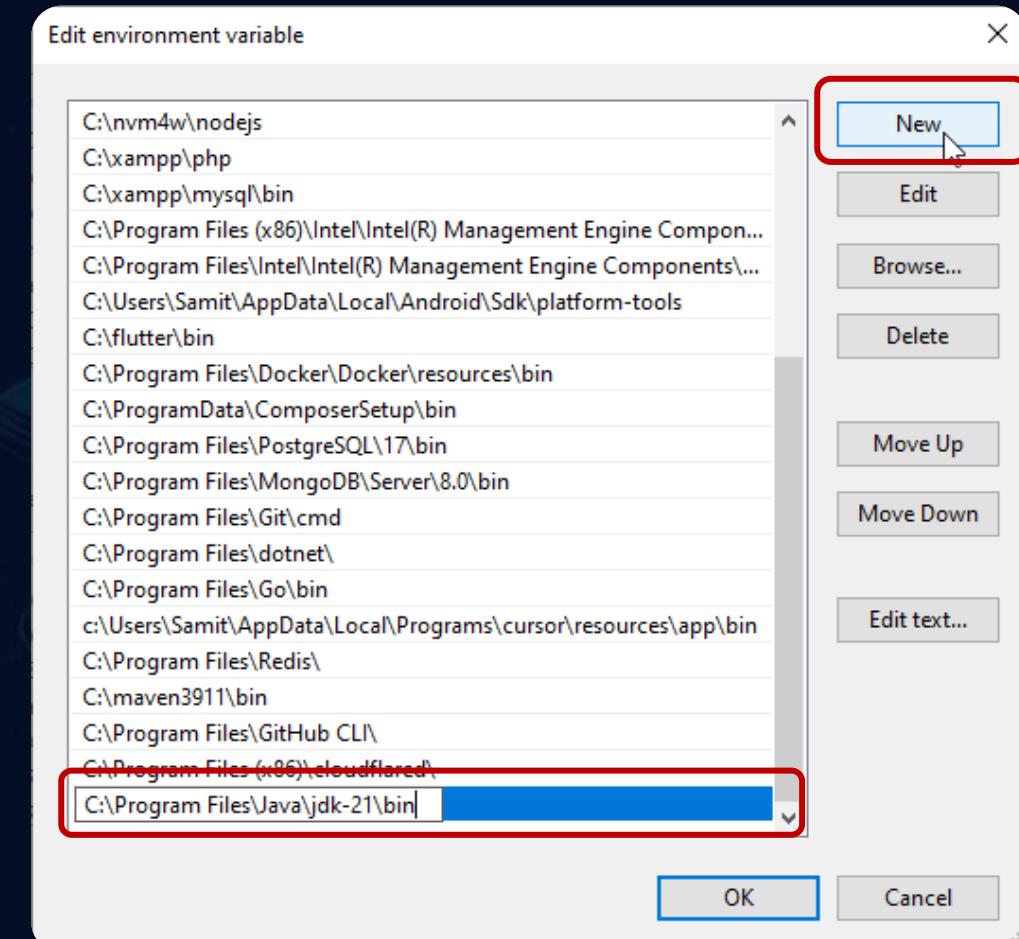
Variable value: **C:\Program Files\Java\jdk-21** www.itgenius.co.th



หลังติดตั้ง Java JDK เสร็จเรามากำหนด path ให้ Windows เรียกใช้งานได้



คลิกที่ Path ดังภาพ และคลิกปุ่ม Edit...



คลิกที่ปุ่ม New... ดังภาพ และกำหนดค่าดังนี้
C:\Program Files\Java\jdk-21\bin



หลังกำหนดค่าเรียบร้อยลงแล้ว ให้เปิด Command Prompt / Terminal ขึ้นมา ตรวจสอบด้วยคำสั่งดังนี้ หากได้ผลลัพธ์ดังภาพถือว่าเรียบร้อย

java -version

set JAVA_HOME

```
Command Prompt
C:\Users\Samit>java -version
java version "21.0.8" 2025-07-15 LTS
Java(TM) SE Runtime Environment (build 21.0.8+12-LTS-250)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.8+12-LTS-250, mixed mode, sharing)
```

```
Command Prompt
C:\Users\Samit>set JAVA_HOME
JAVA_HOME=C:\Program Files\Java\jdk-21

C:\Users\Samit>
```



ส

คอร์สนี้ Jenkins ต้องการ Java JDK **17** หรือ **21** เก่าที่สุด (เก่าหรือใหม่กว่านี้ไม่รองรับ) th



3. ติดตั้ง Node JS



Download Node.JS V.22.x

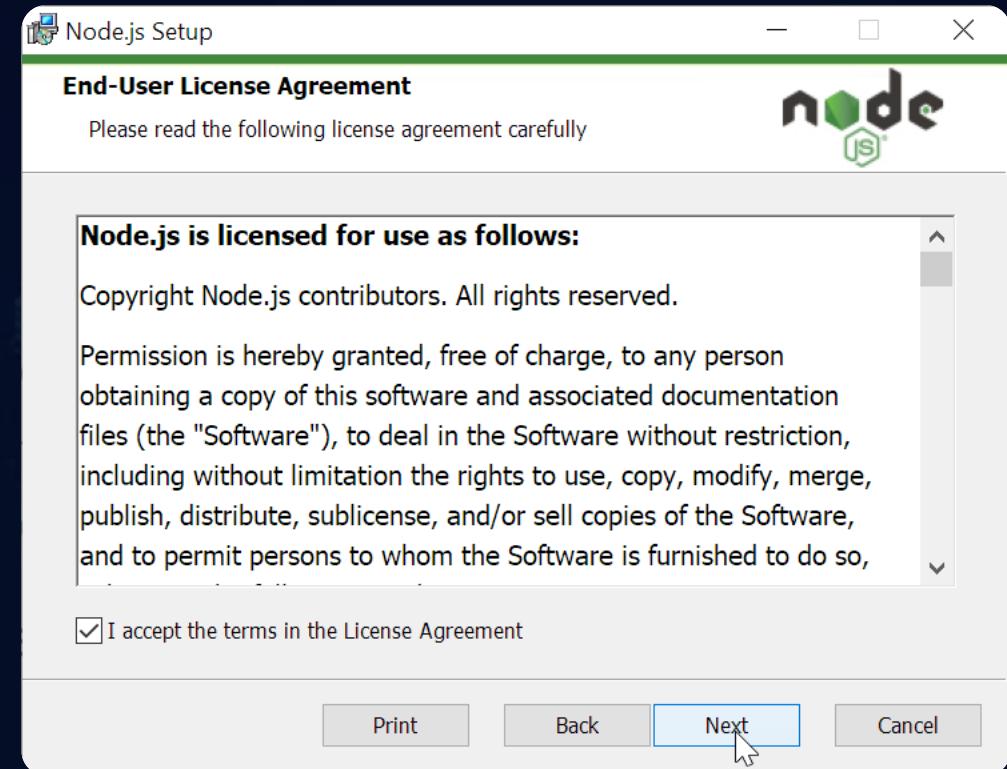
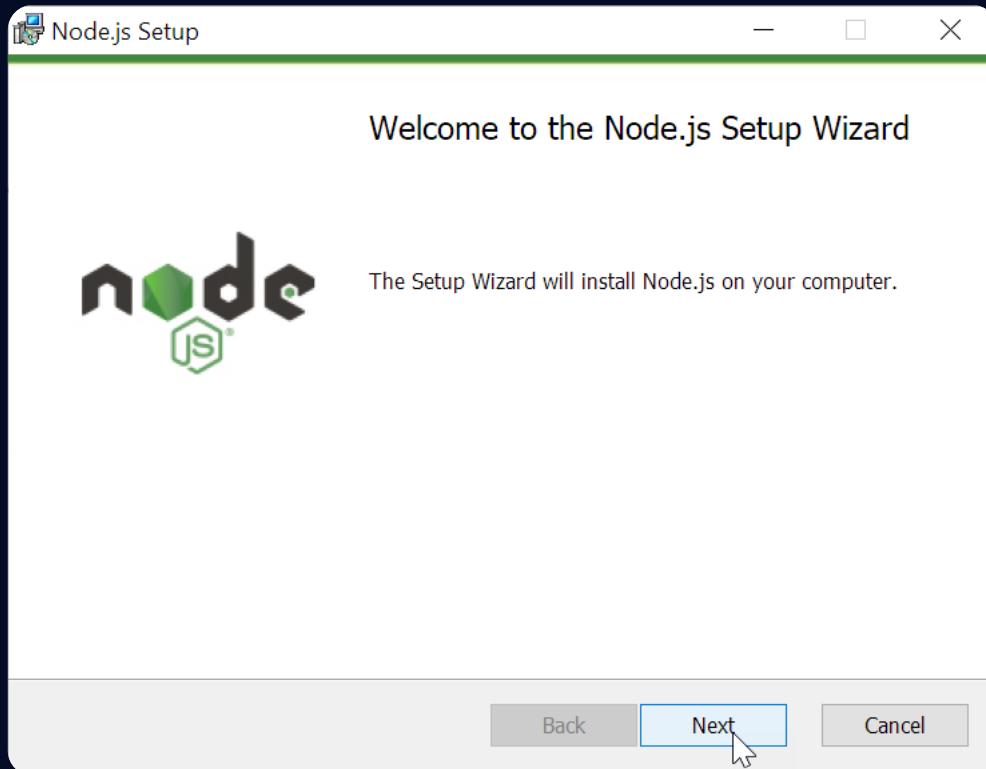
<https://nodejs.org/en/>

The screenshot shows the official Node.js download page at <https://nodejs.org/en/>. The top navigation bar includes links for Learn, About, Download (which is highlighted in green), Blog, Docs, Contribute, Certification, and a search bar. A banner at the top states "New security releases to be made available Wednesday, May 14, 2025". The main section is titled "Download Node.js®" and shows the selection "Get Node.js® v22.15.0 (LTS) for Windows using fpm with npm". Below this, a code block displays the following PowerShell commands:

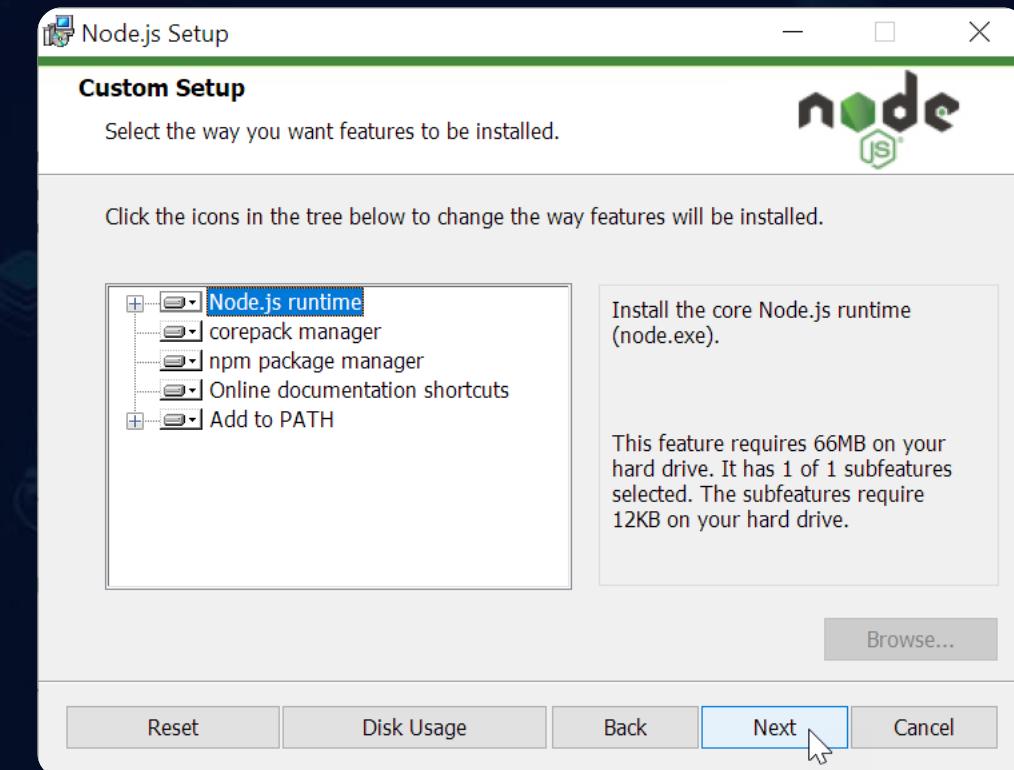
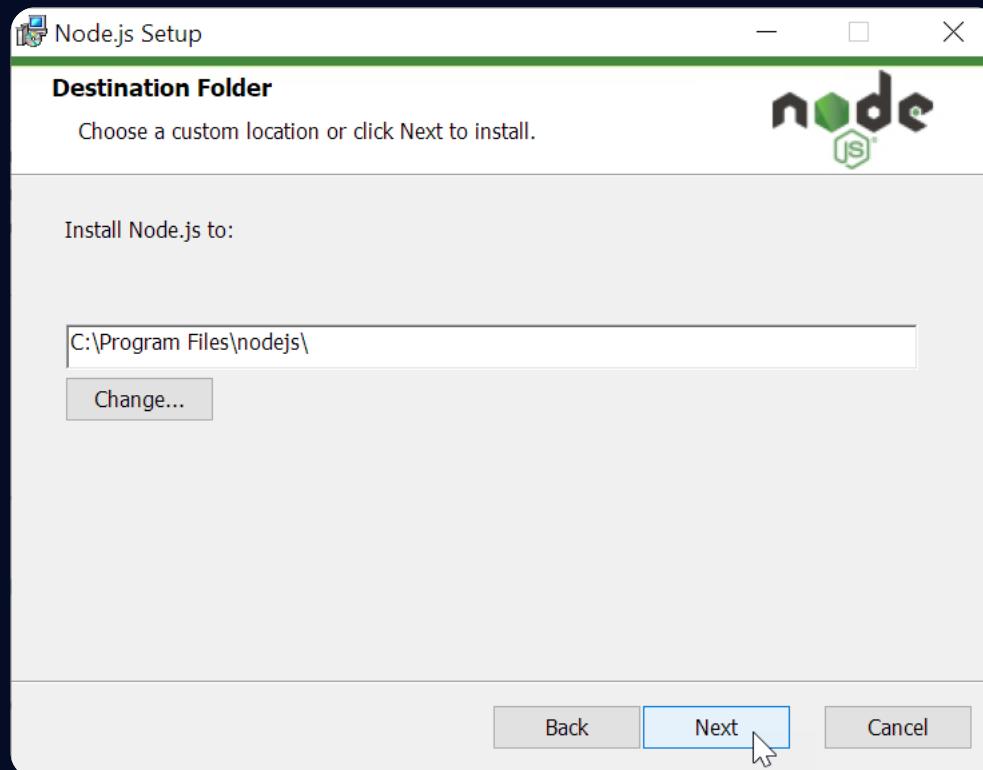
```
1 # Download and install fpm:  
2 winget install Schniz.fpm  
3  
4 # Download and install Node.js:  
5 fpm install 22  
6  
7 # Verify the Node.js version:  
8 node -v # Should print "v22.15.0".  
9  
10 # Verify npm version:  
11 npm -v # Should print "10.9.2".
```

A "Copy to clipboard" button is located below the code block. Further down, it says "Or get a prebuilt Node.js® for Windows running a x64 architecture." with two download buttons: "Windows Installer (.msi)" and "Standalone Binary (.zip)". At the bottom, there's a link to the changelog and blog post for this version.

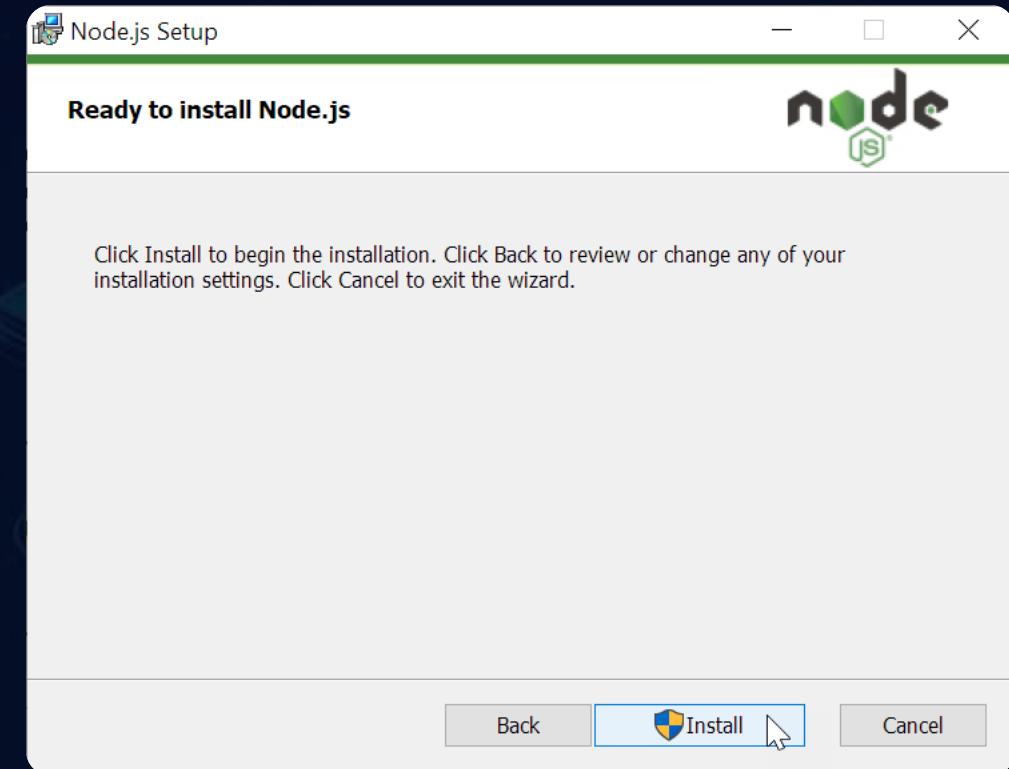
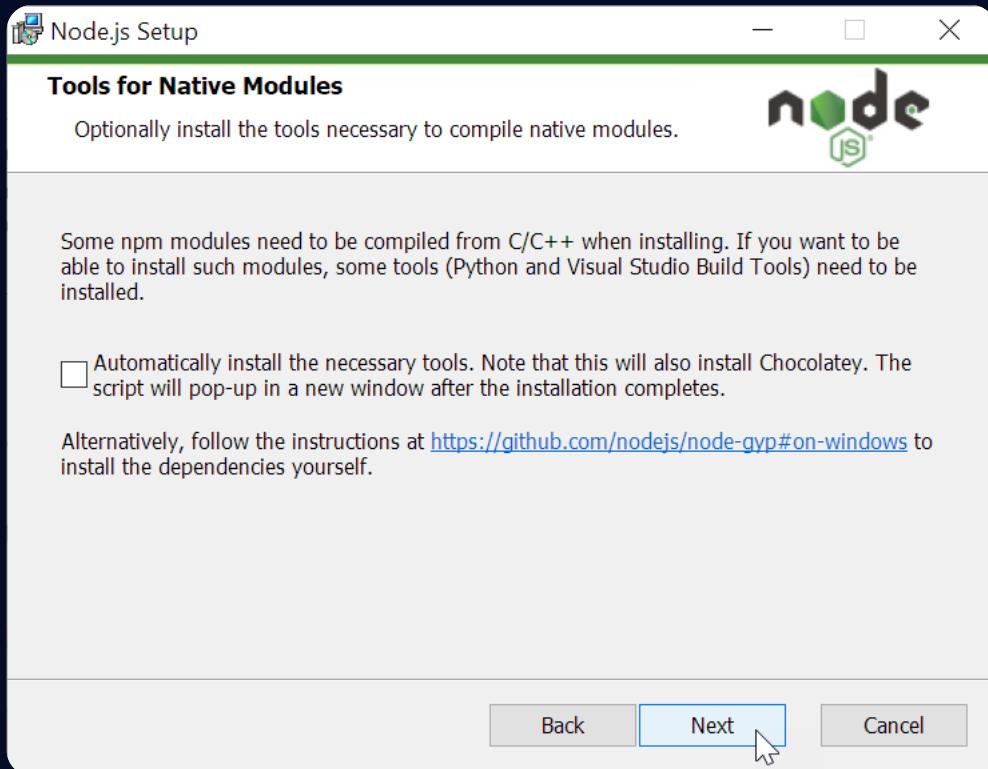
ติดตั้ง Node.js V.22.x



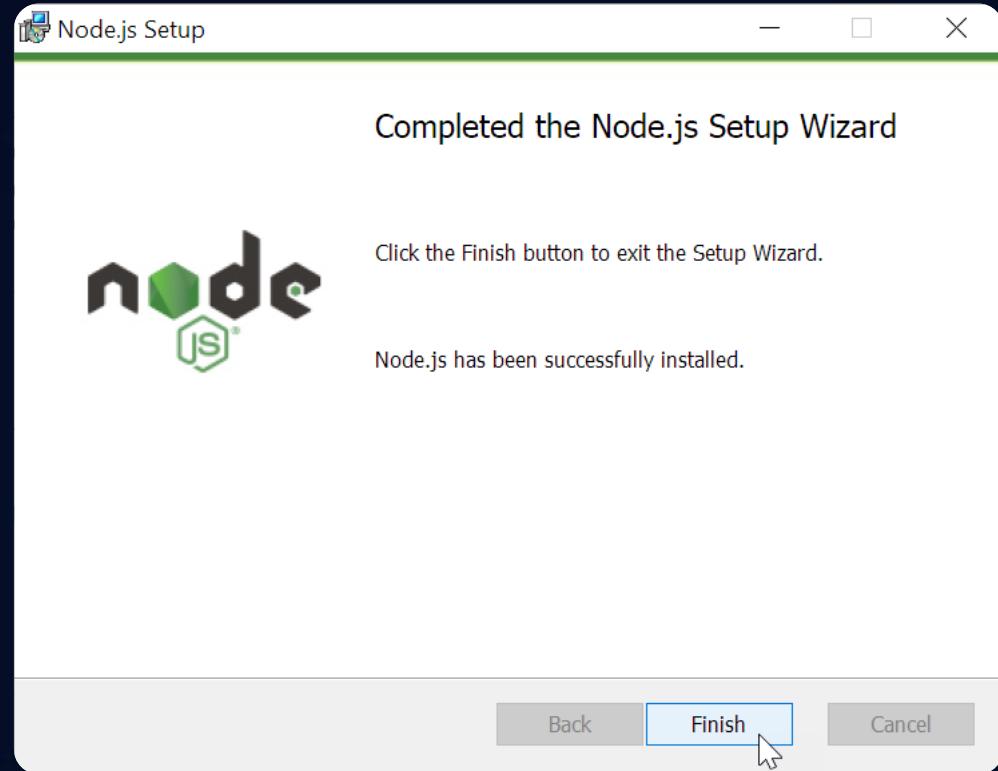
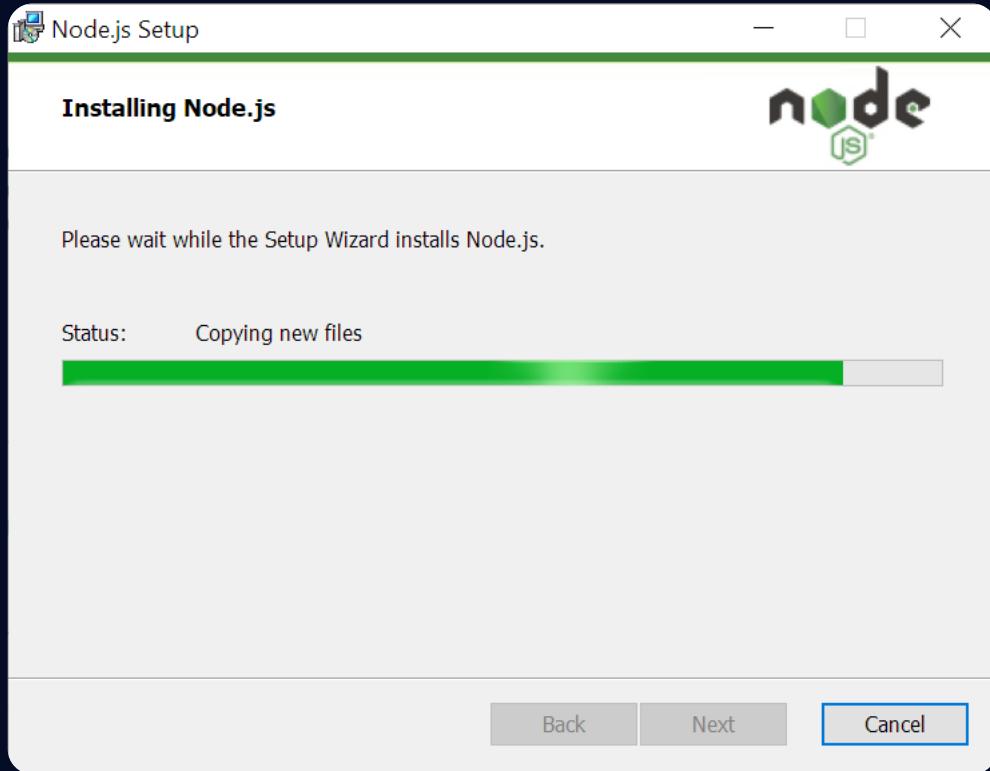
ติดตั้ง Node.js V.22.x



ติดตั้ง Node.js V.22.x



ติดตั้ง Node.js V.22.x



ກດສອບអລັງຕິດຕັ້ງເສັ່ງ

```
node -v
```

```
C:\Users\Samit>node -v  
v22.14.0
```

```
C:\Users\Samit>
```

```
npx -v
```

```
C:\Users\Samit>npx -v  
10.9.2
```

```
C:\Users\Samit>
```

```
npm -v
```

```
C:\Users\Samit>npm -v  
10.9.2
```

```
C:\Users\Samit>
```

ໝາຍເຫດ ກາວອບມອງຮັບຕັ້ງແຕ່ Node.JS 20 ຂຶ້ນໄປ ສາມາຄໃ້ Node.JS 21 , 22, 23 ມີເຊື້ອ 24 ກີດໄດ້





4. ติดตั้ง Python



The screenshot shows the Python Releases for Windows page. The 'Downloads' tab is selected, highlighted with a red box. A dropdown menu is open under 'Windows', also highlighted with a red box. The menu items include 'All releases', 'Source code', and 'Windows'. The 'Windows' item is the current selection. To the right, there is a large orange callout box with the following text:

ก่อนดาวน์โหลดติดตั้งลงเครื่อง cmd/terminal มาเช็คด้วยคำสั่ง

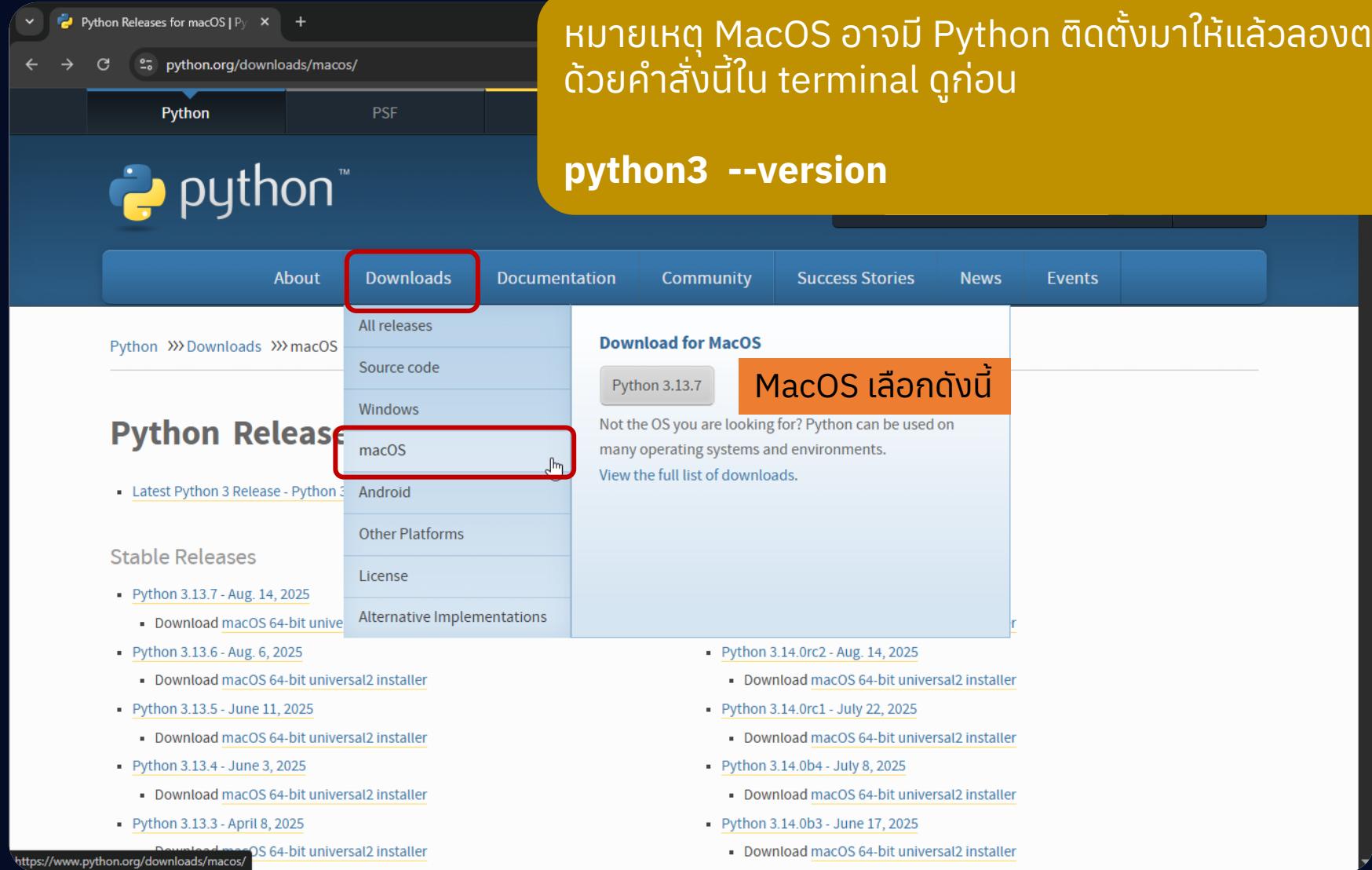
python --version

ถ้าพบ version python 3.x ขึ้นไป ก็ถือว่าใช้ได้ ถ้าต่ำกว่านี้ ก่อนอุปกรณ์แล้วดาวน์โหลดหน้านี้มาติดตั้งใหม่ได้เลย

https://www.python.org/ftp/python/3.13.7/python-3.13.7-amd64.exe

ดาวน์โหลด Python 3.x ได้ที่ <https://www.python.org/downloads/windows/>

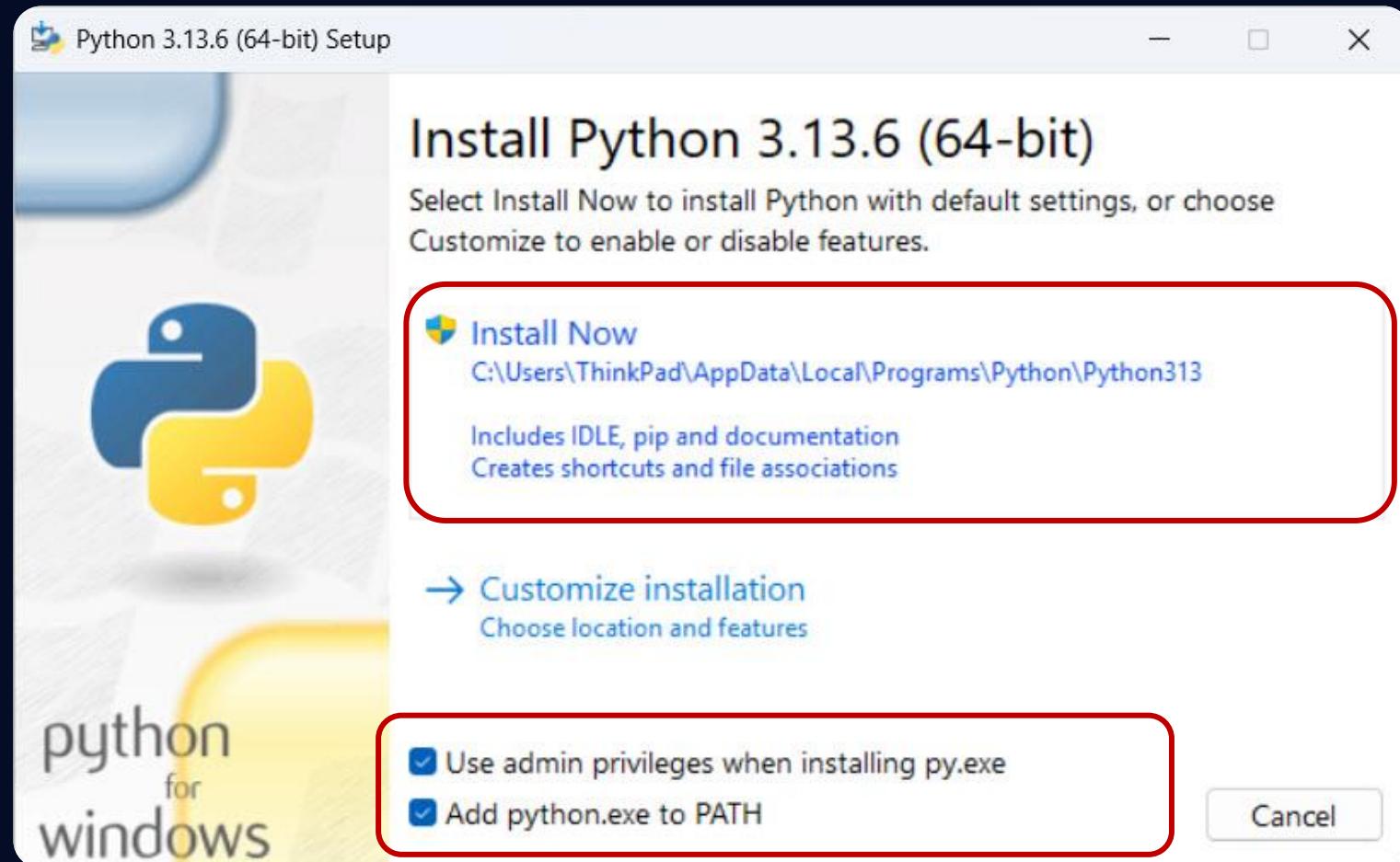




ดาวน์โหลด Python 3.x ได้ที่ <https://www.python.org/downloads/macos/>



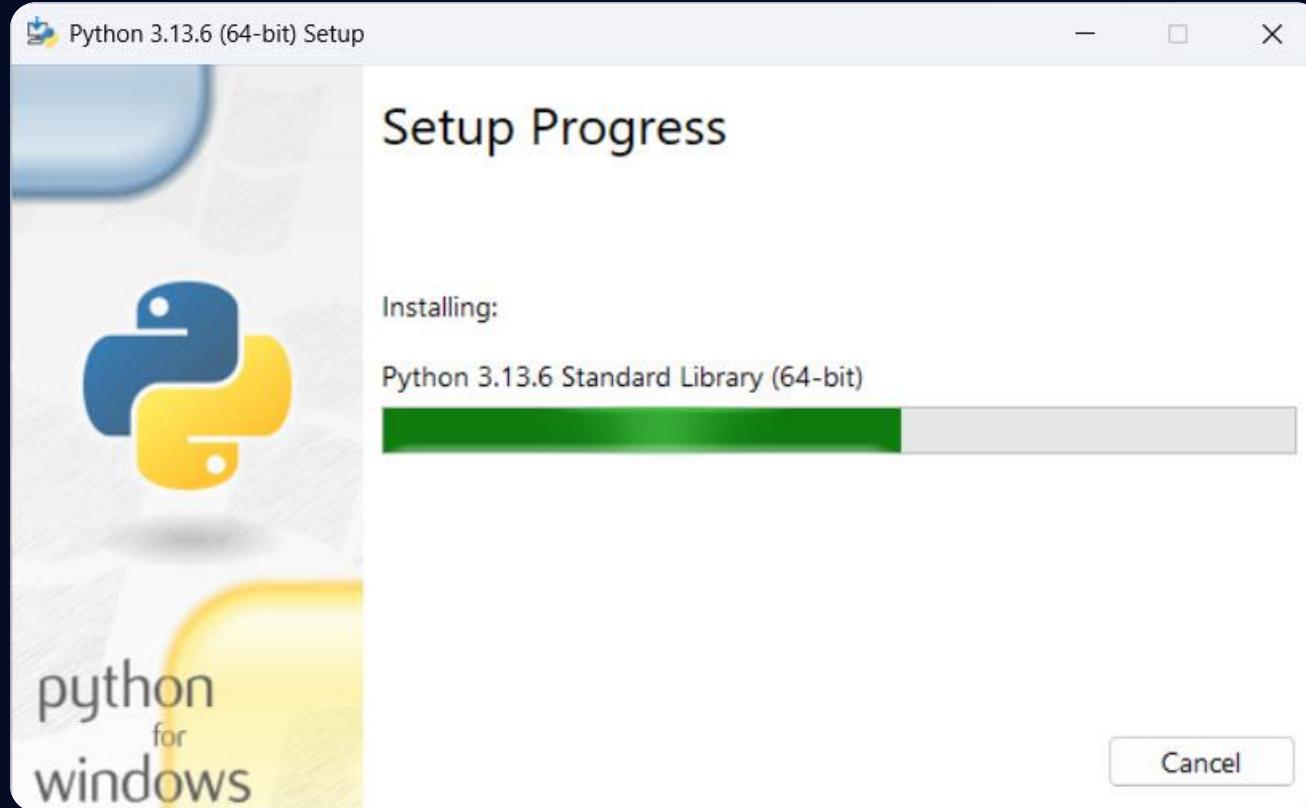
หลังดาวน์โหลดไฟล์มาแล้วทำการติดตั้ง เลือกดังภาพ



อย่าลืม! เช็คบ็อกเลือก 2 รายการด้านล่างก่อนเริ่มทำการติดตั้งนะครับ



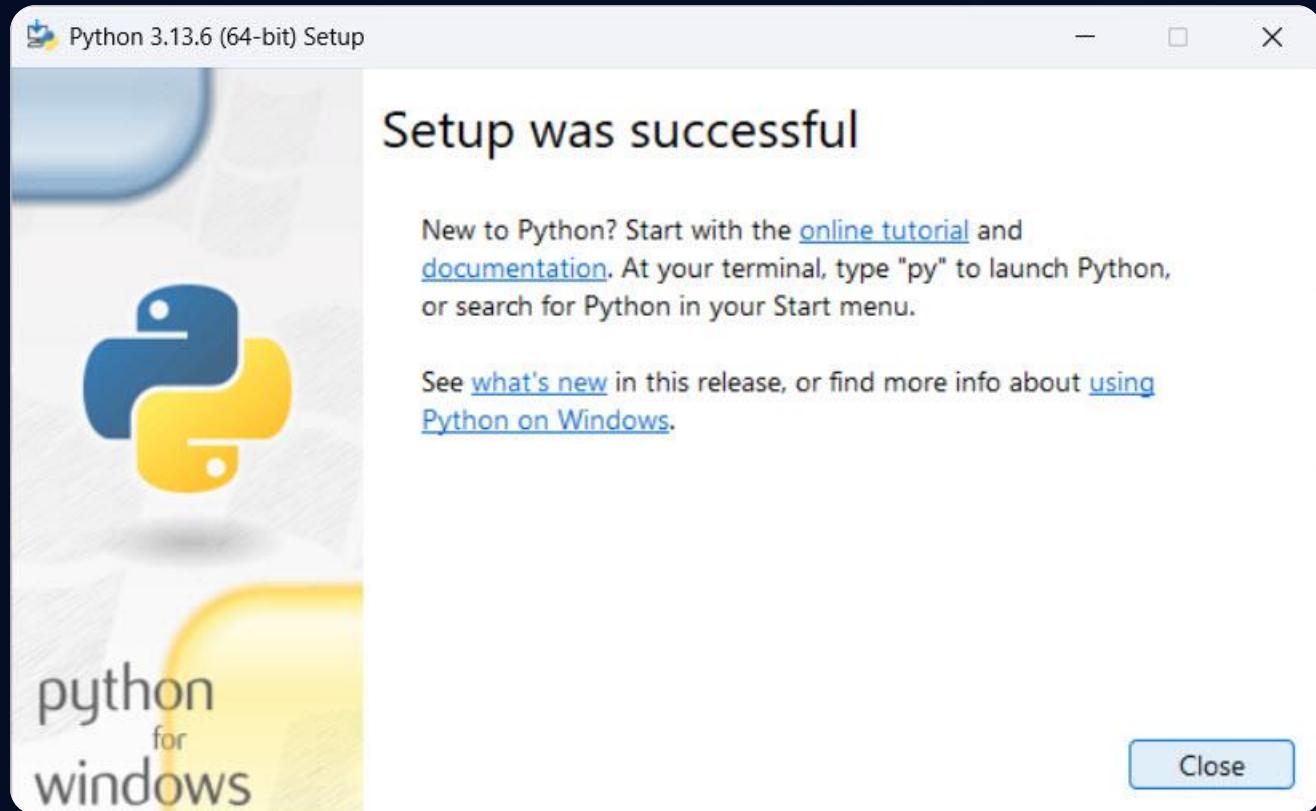
รอการติดตั้ง



รอจัดตั้งแล้วเสร็จ...



ติดตั้งเสร็จเรียบร้อย



เมื่อถึงหน้านี้แสดงว่าติดตั้งเรียบร้อยแล้วคลิกปุ่ม Close



ກດສອບჩັດຕິທີ່ເສື້ຈ

Windows

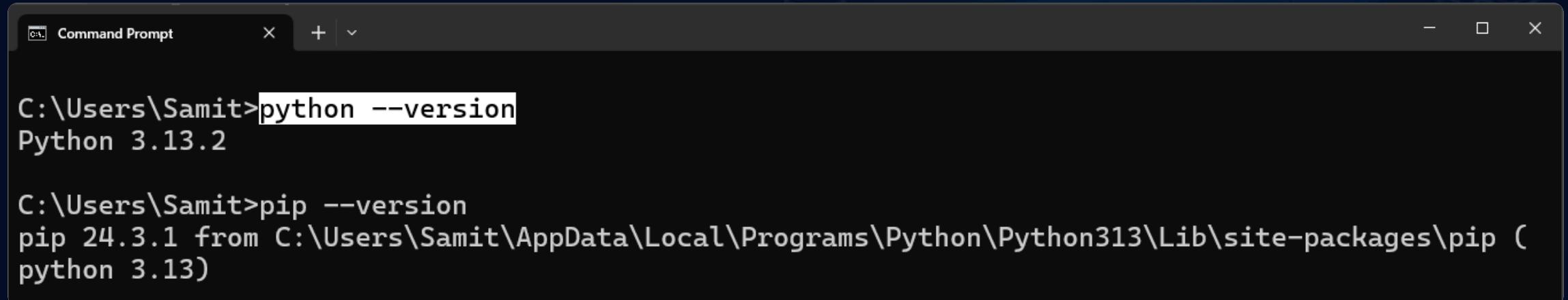
python --version

pip --version

MacOS

python3 --version

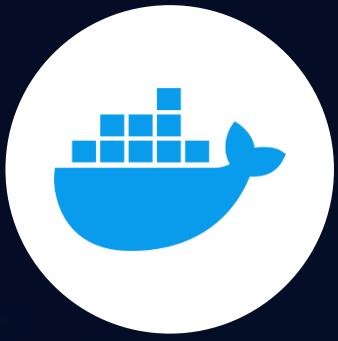
pip3 --version



```
C:\Users\Samit>python --version
Python 3.13.2

C:\Users\Samit>pip --version
pip 24.3.1 from C:\Users\Samit\AppData\Local\Programs\Python\Python313\Lib\site-packages\pip (python 3.13)
```





5. ติดตั้ง Docker Desktop



System Requirements Windows



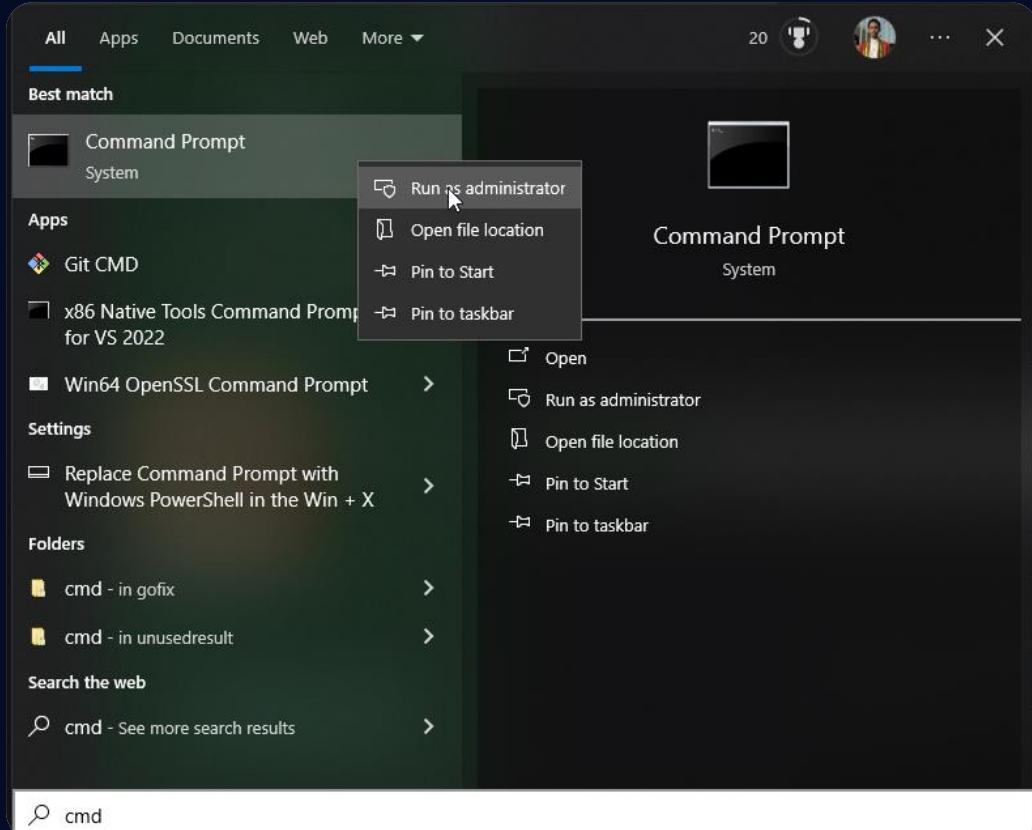
Docker Engine 20.10.9
Docker Desktop 4.10

- Windows 10 64-bit: Pro 2004 (build 19041) or higher, or Enterprise or Education 1909 (build 18363) or higher.
- For Windows 10 Home, see System requirements for WSL 2 backend.
- Hyper-V and Containers Windows features must be enabled.
- The following hardware prerequisites are required to successfully run Client Hyper-V on Windows 10:
- 64 bit processor with Second Level Address Translation (SLAT)
- 4GB system RAM
- BIOS-level hardware virtualization support must be enabled in the BIOS settings

<https://docs.docker.com/desktop/windows/install/>



สำหรับ Windows 10/11 แนะนำให้ติดตั้ง wsl ก่อน



คลิก start พิมพ์ cmd คลิกเปิดด้วย Run as Administrator

```
Administrator: C:\Windows\System32\cmd.exe - wsl --install
Microsoft Windows [Version 10.0.19043.2006]
(c) Microsoft Corporation. All rights reserved.

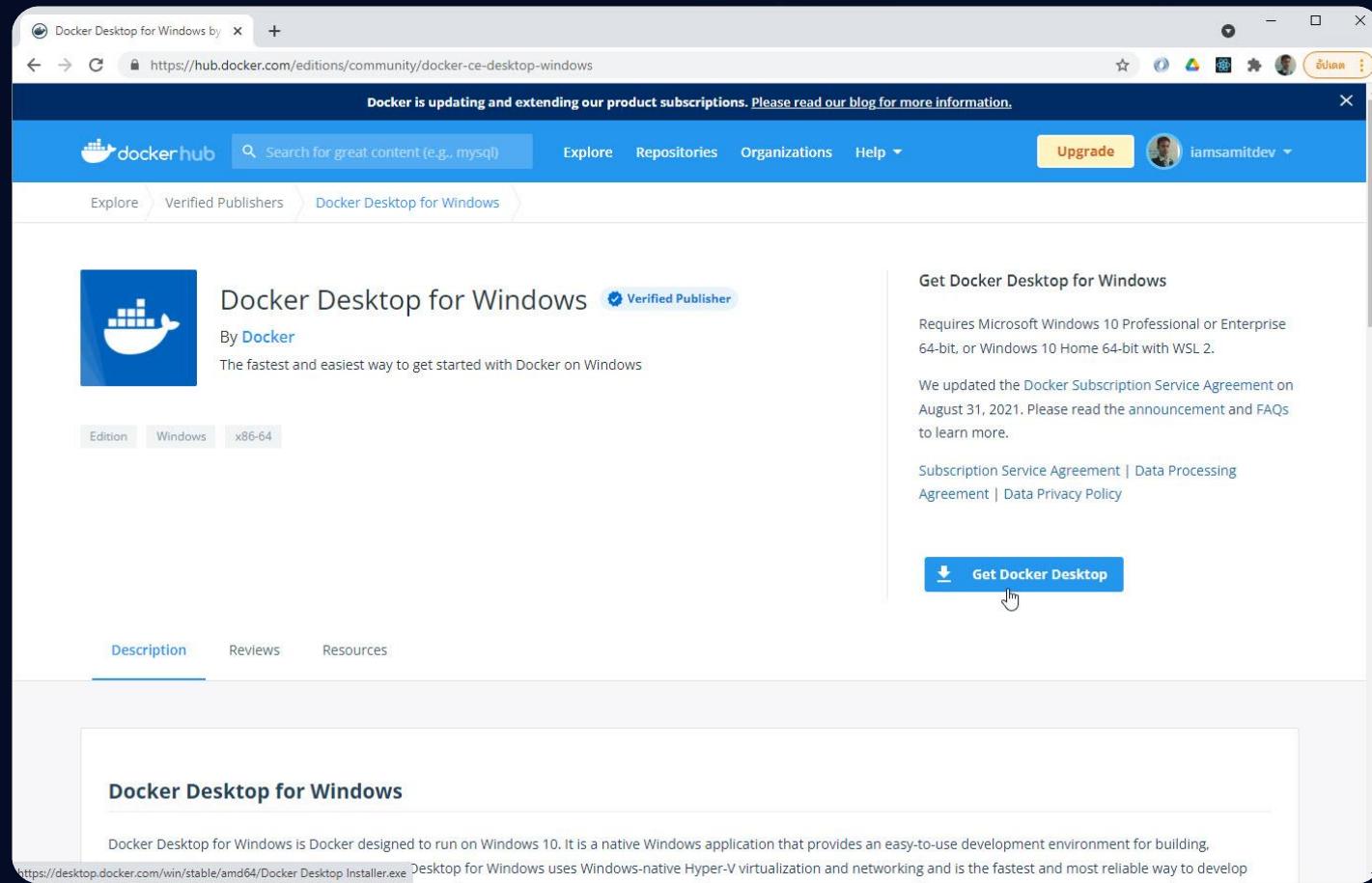
C:\WINDOWS\system32>wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Downloading: WSL Kernel
[=====] 27.2%
```

พิมพ์คำสั่งติดตั้ง wsl ด้วย **wsl --install**



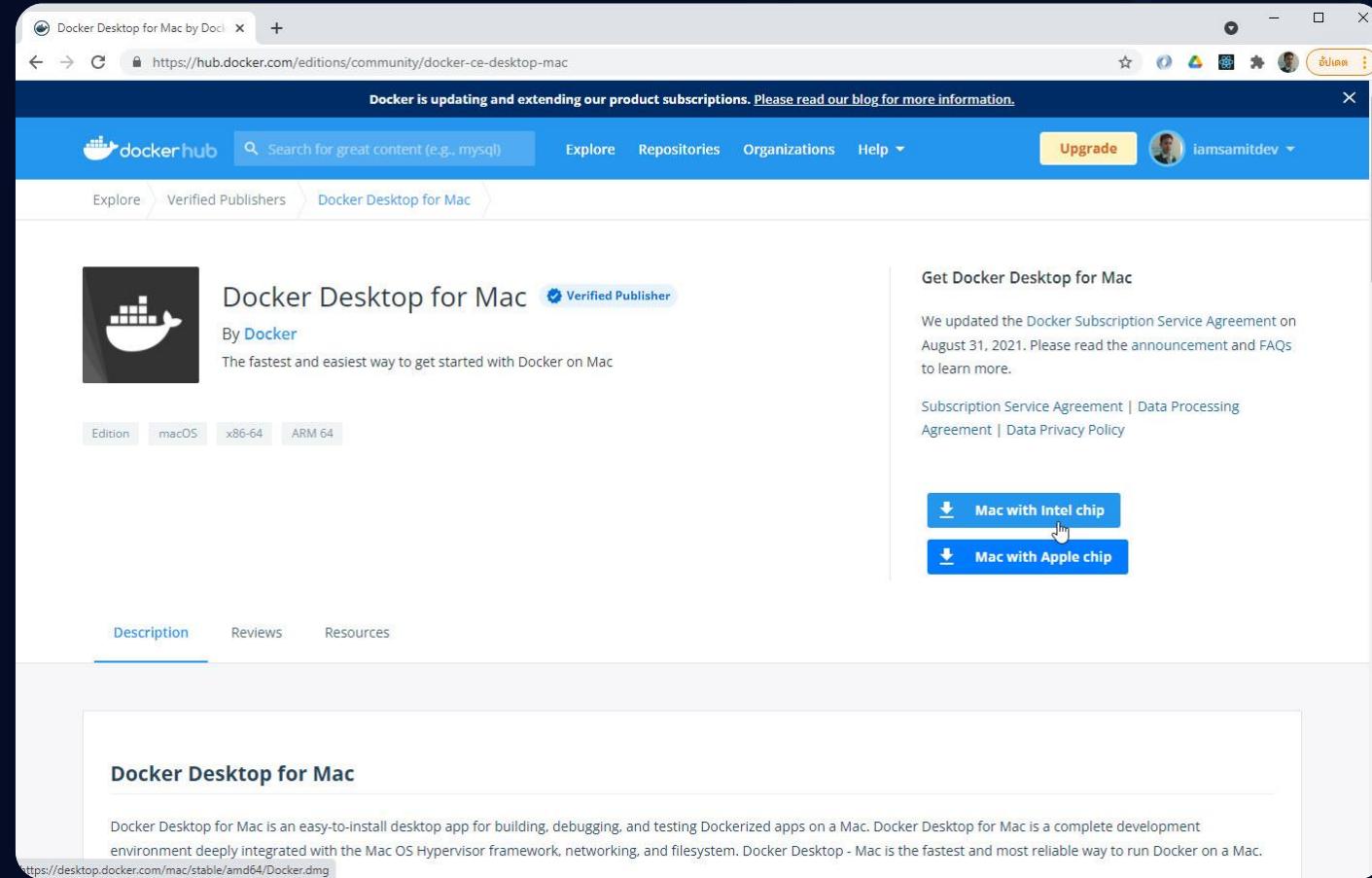
Download Docker Desktop for Windows

<https://hub.docker.com/editions/community/docker-ce-desktop-windows>

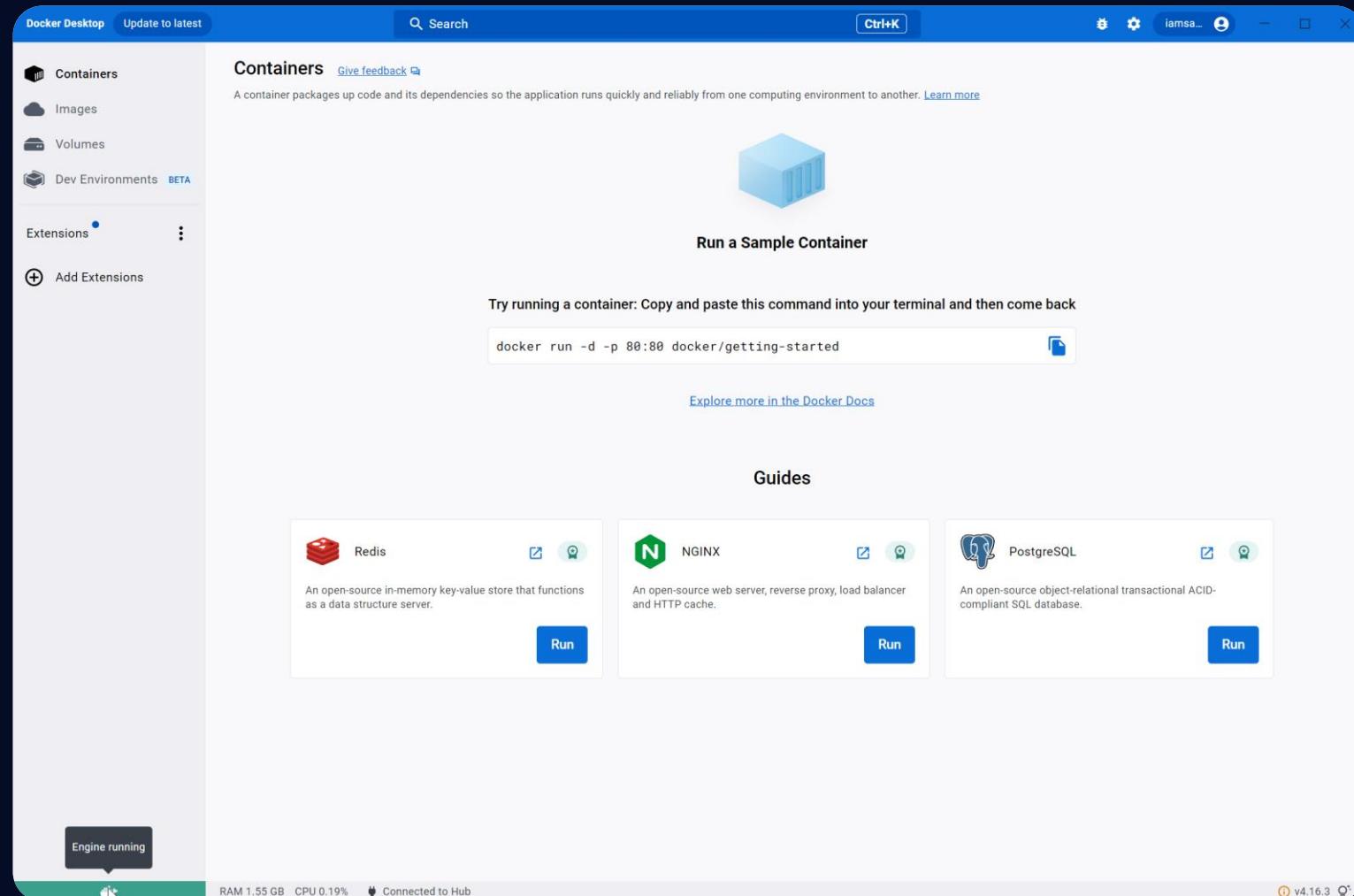


Download Docker Desktop for MacOS

<https://hub.docker.com/editions/community/docker-ce-desktop-mac>



ตัวอย่าง Docker Desktop หลังจากติดตั้งเรียบร้อยแล้ว





6. ติดตั้ง Git



ดาวน์โหลดไฟล์ติดตั้ง Git ได้ที่ <https://git-scm.com/>

The screenshot shows the official website for Git (<https://git-scm.com/>). The page features a dark background with a central diagram illustrating a distributed version control system where multiple repositories are interconnected by bidirectional arrows. At the top, there's a search bar and a 'Relaunch to update' button. Below the header, two main descriptive paragraphs highlight Git's benefits: it's a free and open source distributed version control system designed for efficiency, and it's easy to learn with lightning fast performance, outclassing other tools like Subversion, CVS, Perforce, and ClearCase.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

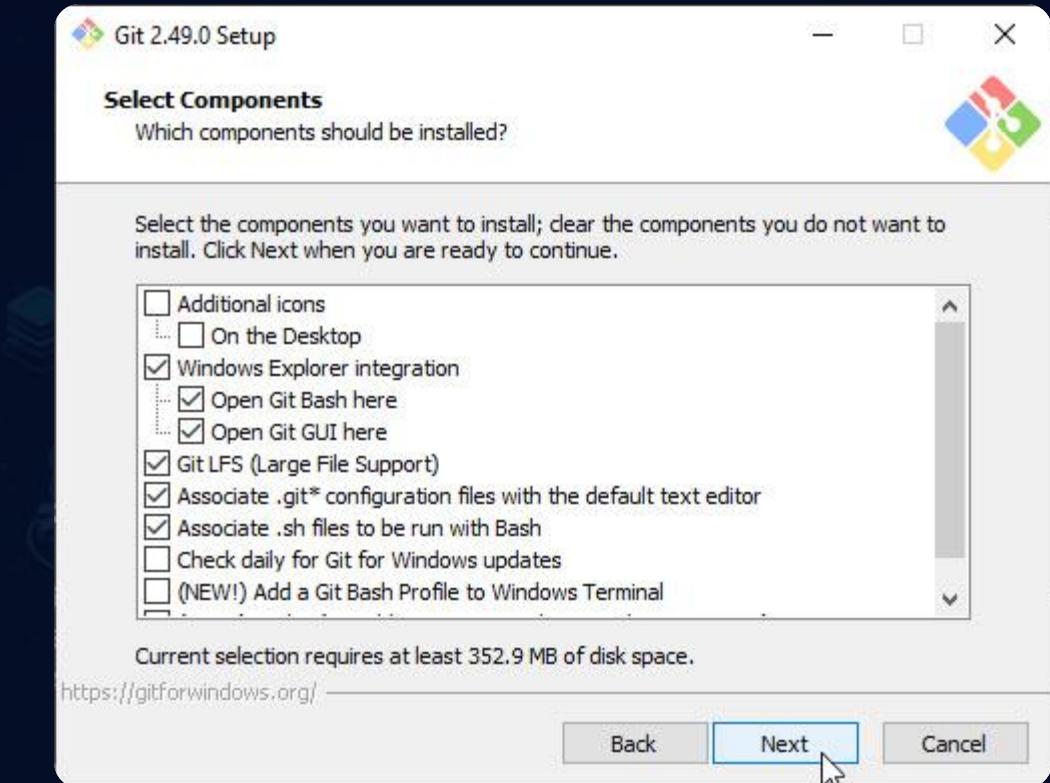
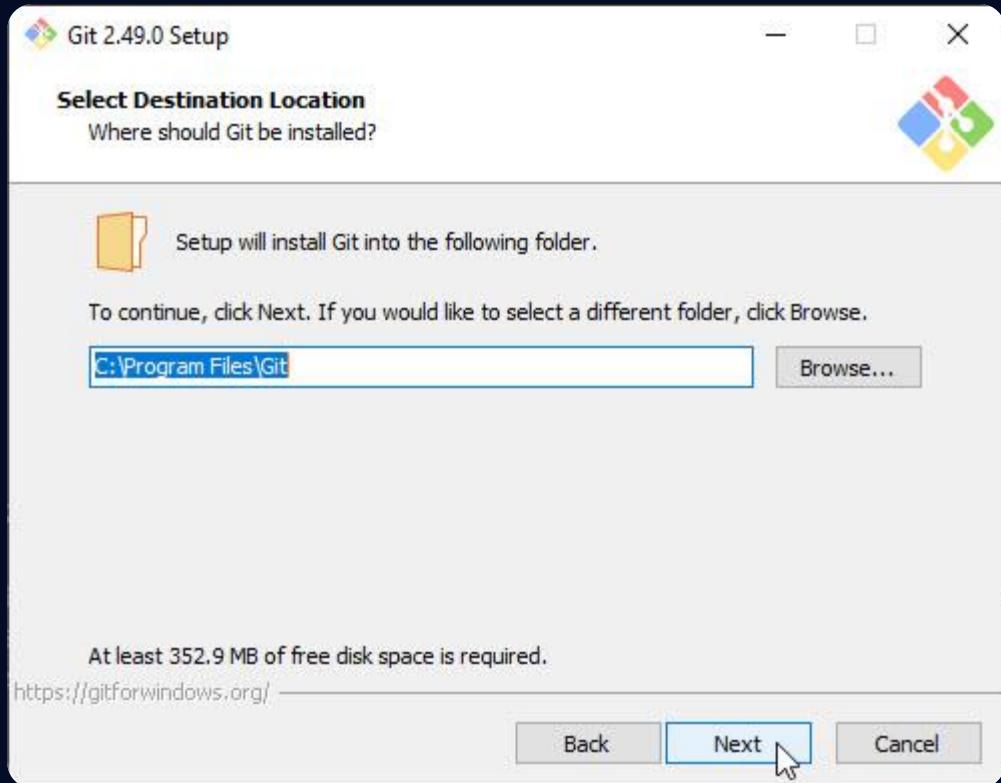
Latest source Release
2.49.0
[Release Notes \(2025-03-14\)](#)
[Download for Windows](#)

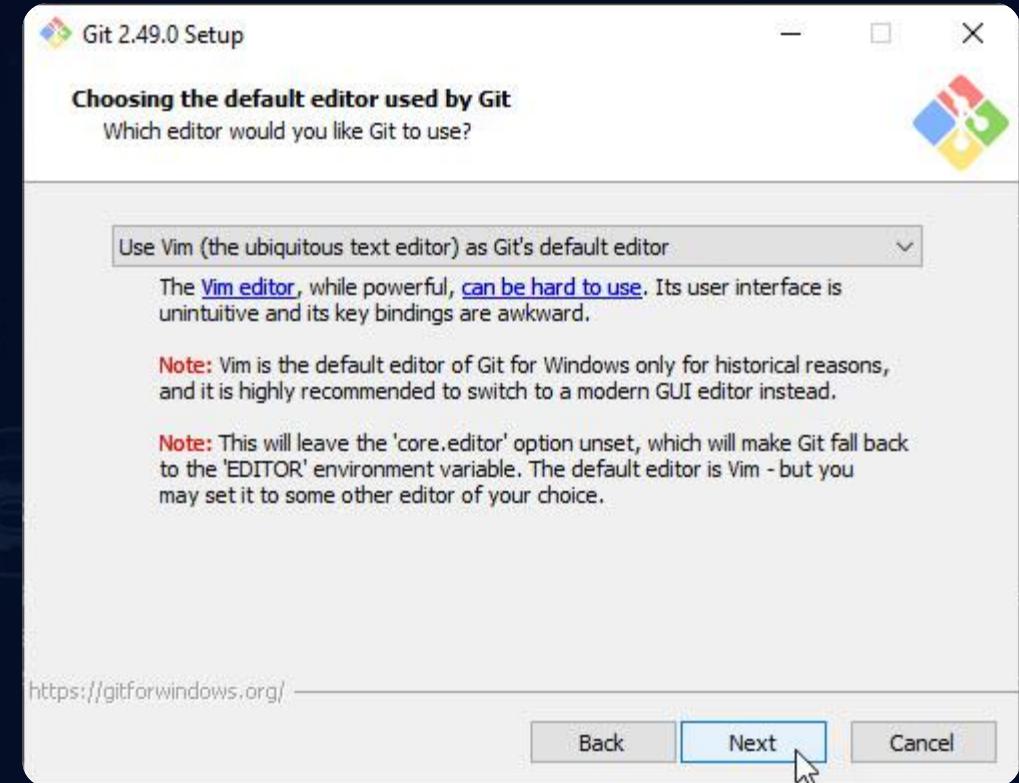
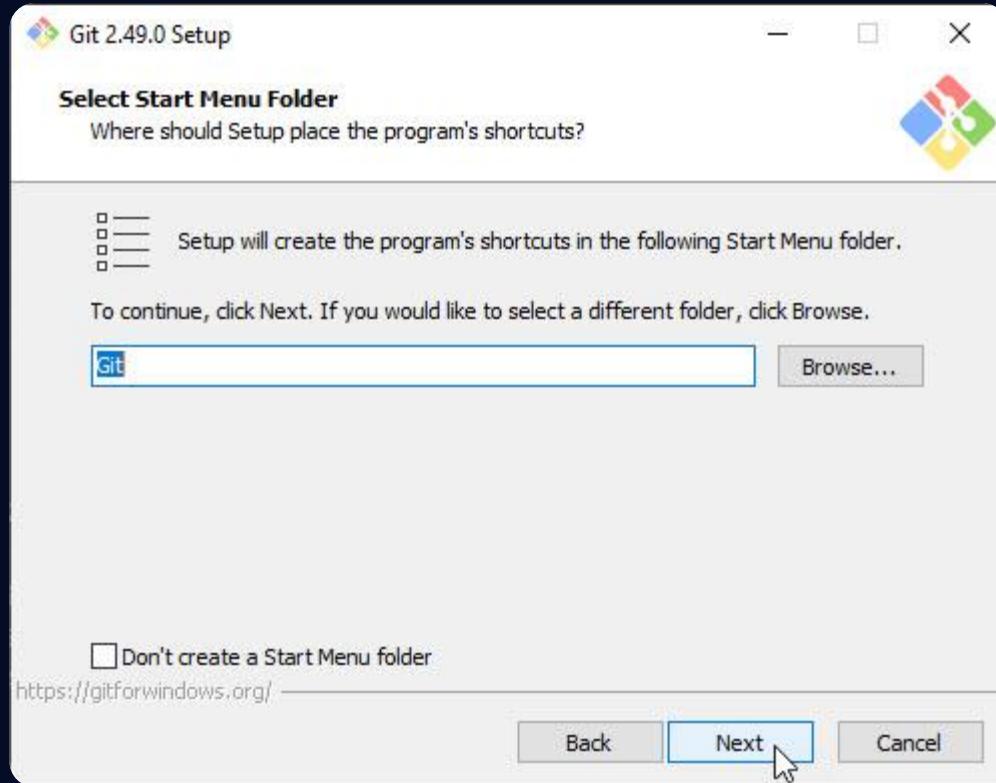
Products providing Git hosting

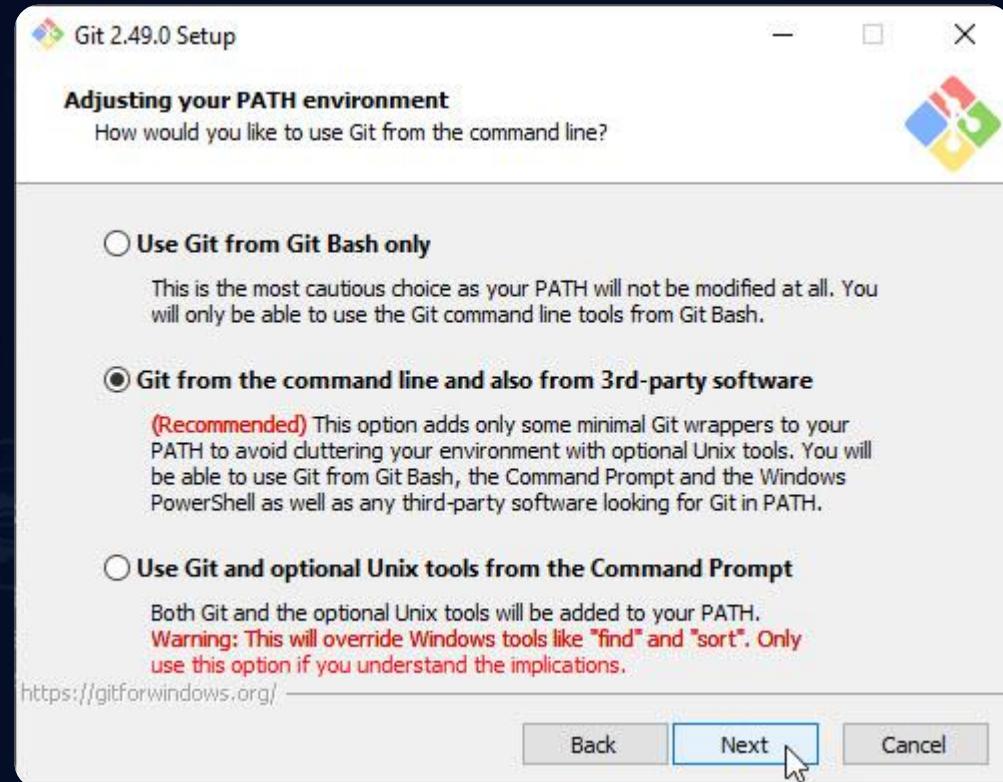
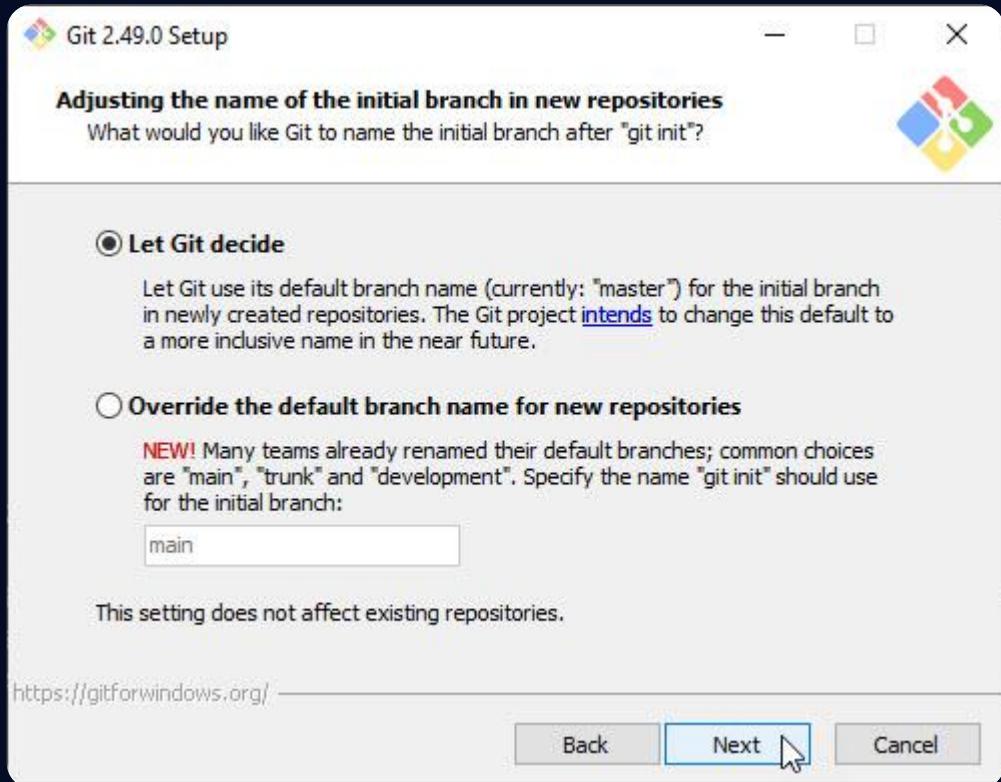
[Windows GUIs](#) [Tarballs](#)
[Mac Build](#) [Source Code](#)

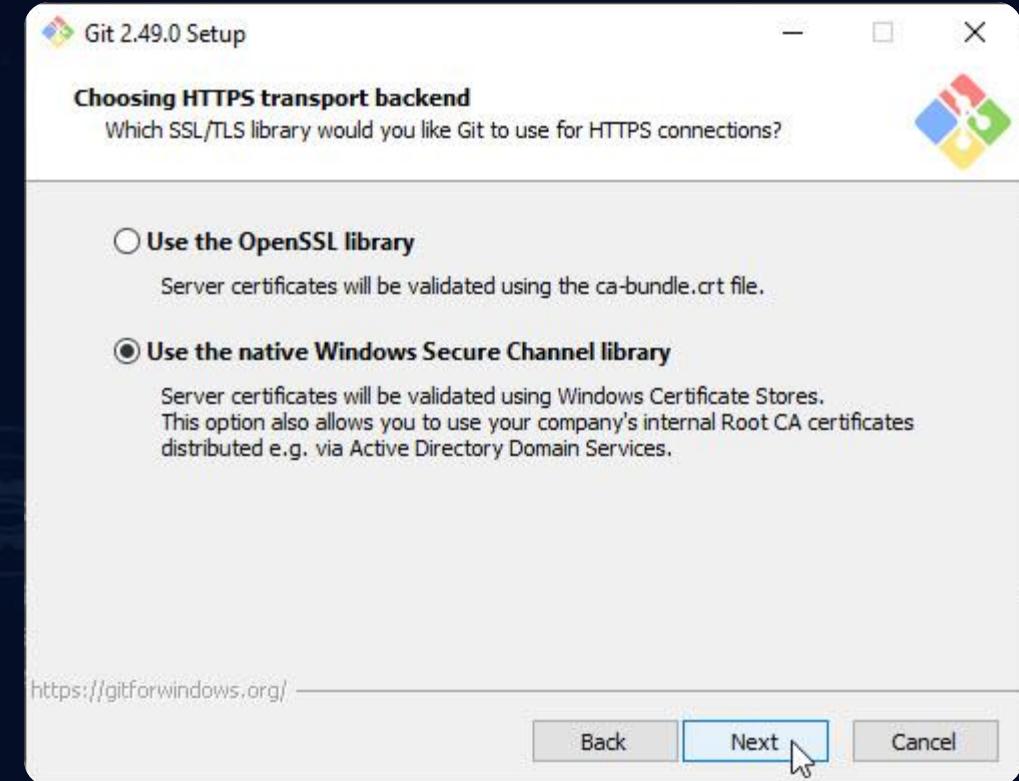
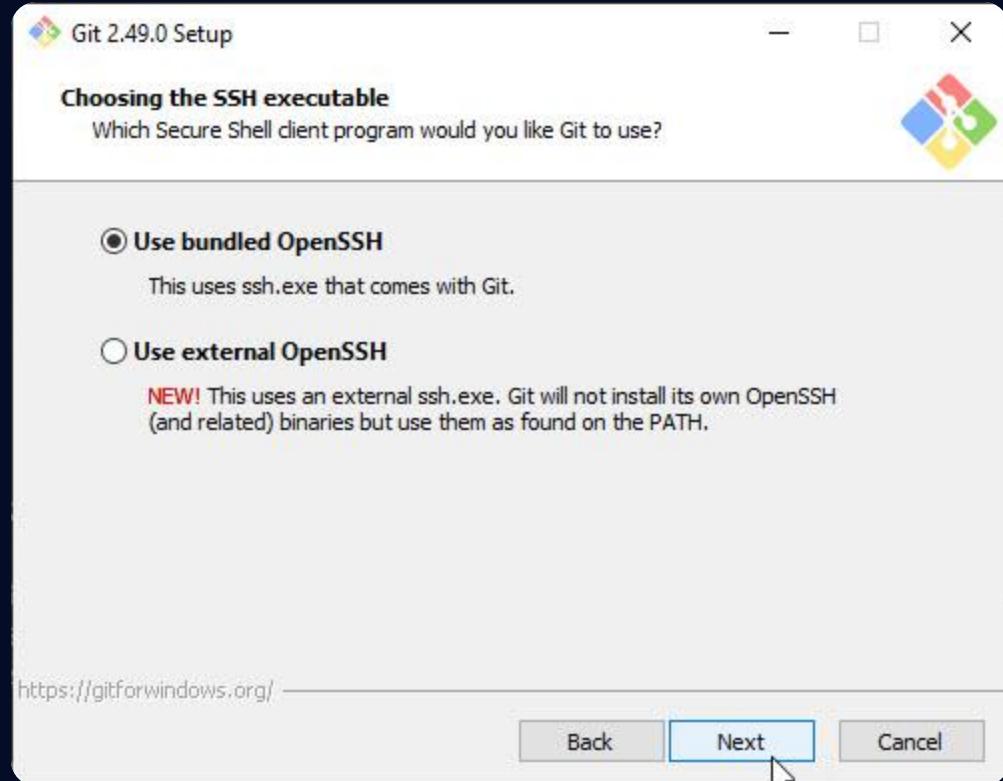


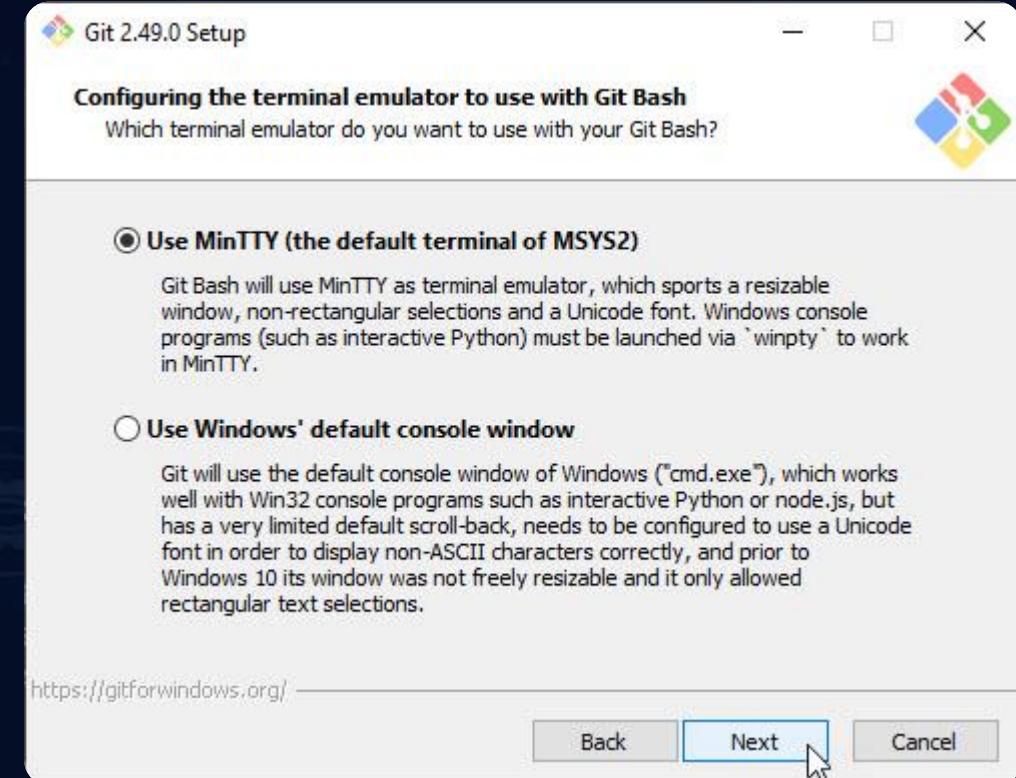
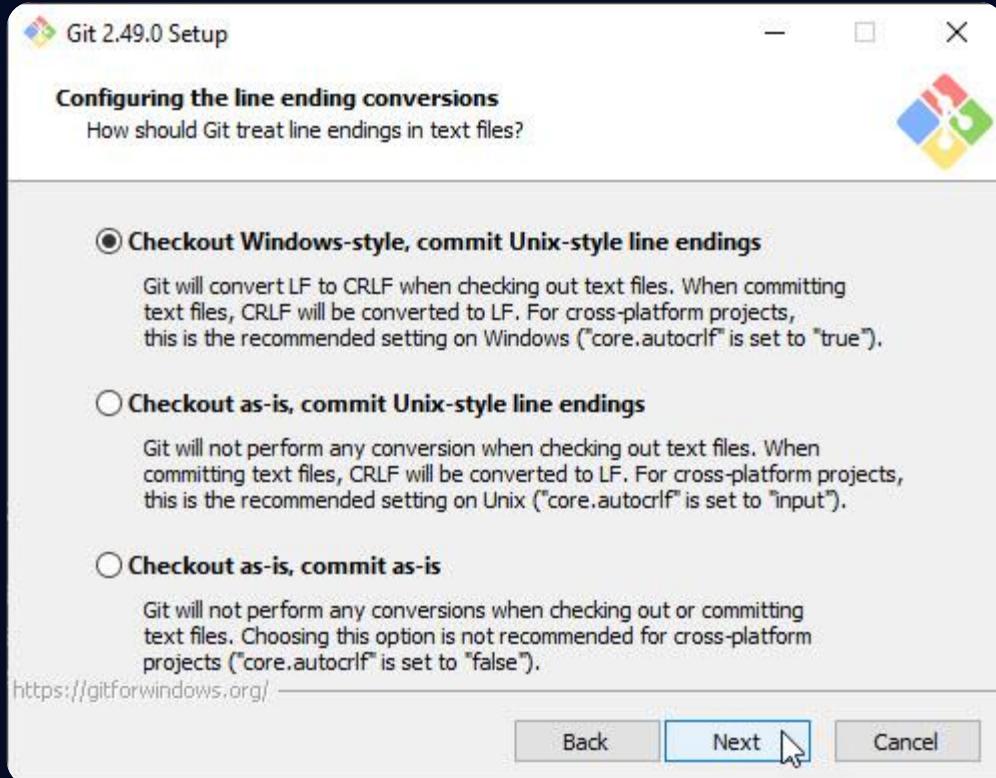


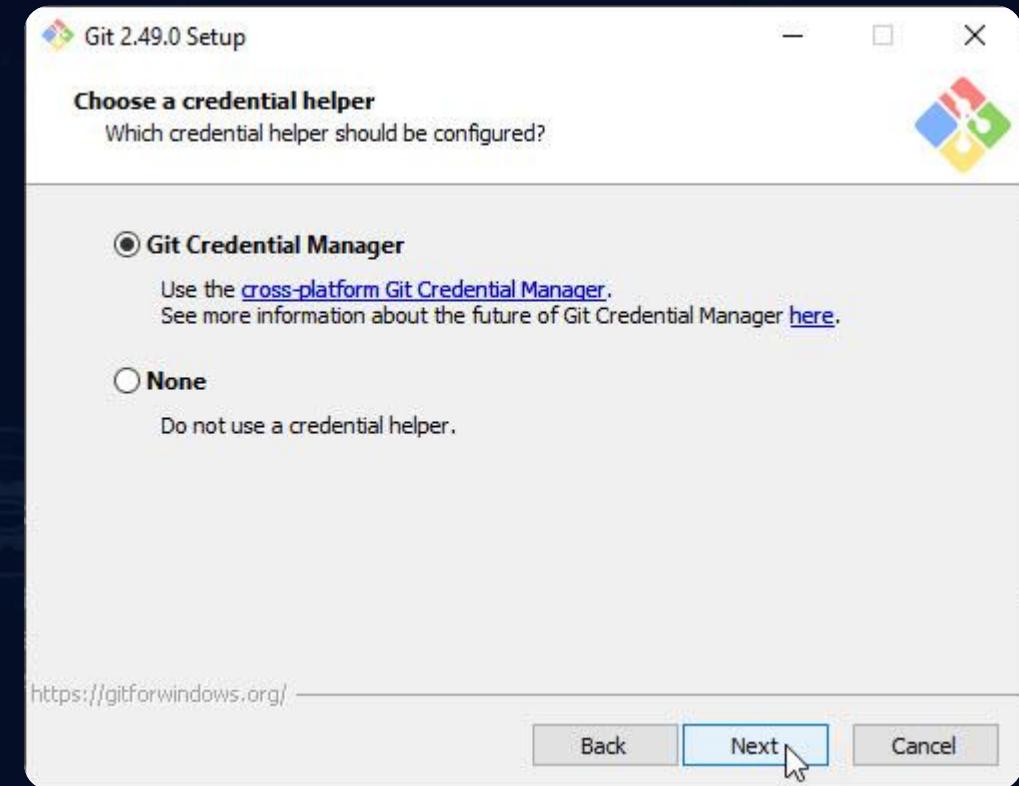
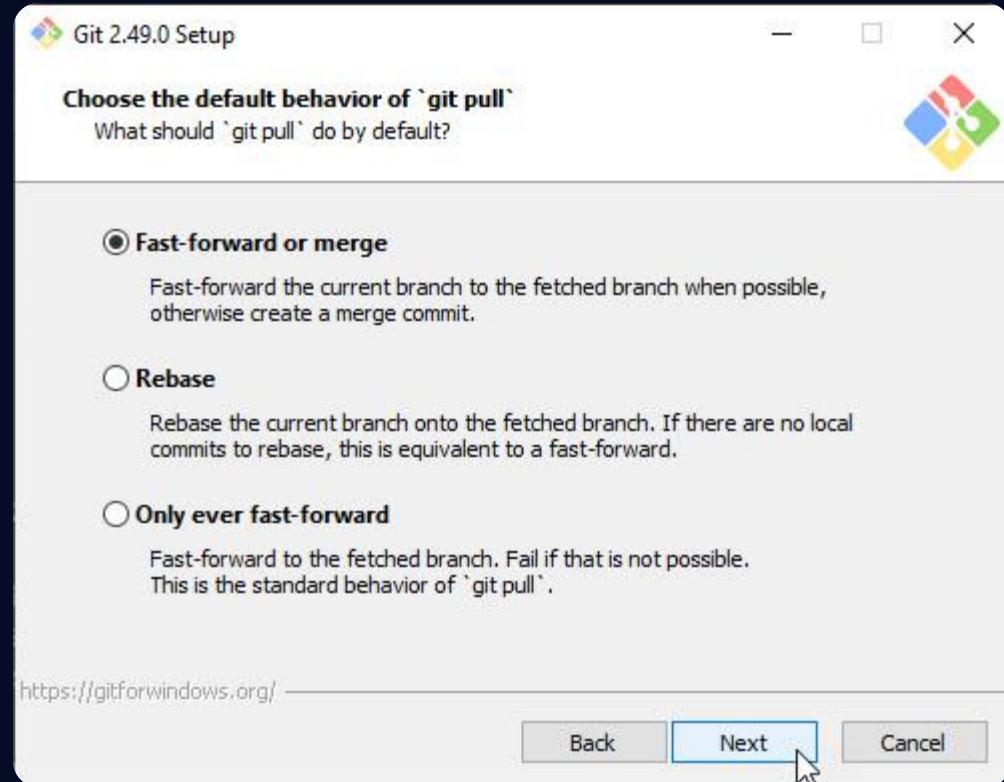


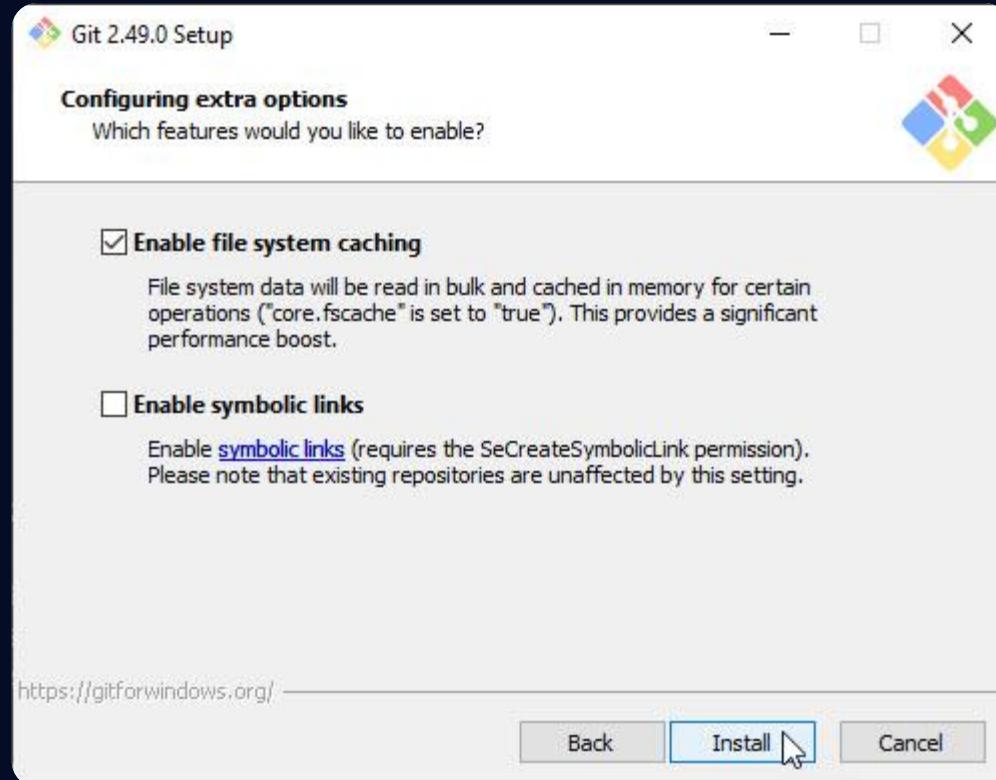








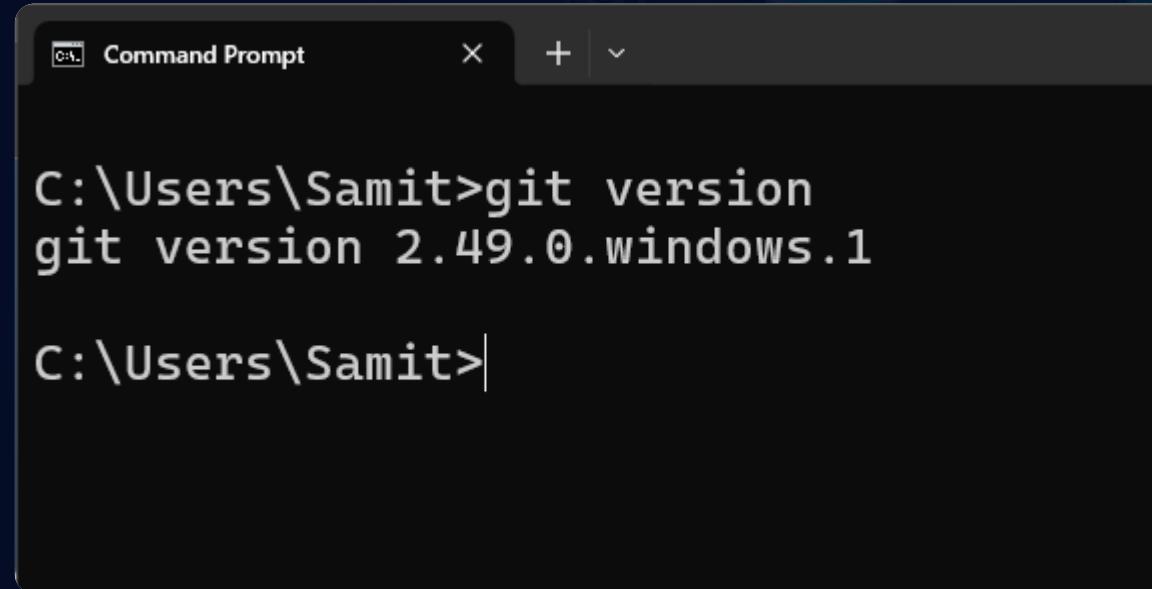




หลังติดตั้งสำเร็จกดสอบด้วยคำสั่ง

git version

หากพบเวอร์ชันดังภาพ ถือว่าติดตั้งเรียบร้อยพร้อมใช้งาน



```
Command Prompt
C:\Users\Samit>git version
git version 2.49.0.windows.1
C:\Users\Samit>
```



การตรวจสอบความเรียบร้อยของเครื่องมือที่ติดตั้งบน Windows / Mac OS / Linux

เปิด Command Prompt บน Windows หรือ Terminal บน Mac ขึ้นมาป้อนคำสั่งดังนี้

Visual Studio Code

```
code --version
```

Node JS

```
node -v  
npm -v  
npx -v
```

Java

```
java -version
```

Python (windows)

```
python --version  
pip --version
```

Python (macos)

```
python3 --version  
pip3 --version
```

Docker

```
docker --version
```

Git

```
git version
```





พื้นฐานการใช้งาน Git

- เริ่มต้นใช้งาน Git ในโปรเจกต์
- การตั้งค่า , การ clone , คำสั่งพื้นฐาน



Git First time setup

กำหนดข้อมูลผู้ใช้

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@example.com
```

คำสั่งเช็คข้อมูลที่กำหนดไว้

```
git config --list  
git config --global --list
```



Set Default branch “main”

ตั้งค่าให้ branch หลักชื่อ “main”

```
git config --global init.defaultBranch main
```

ตรวจสอบหลังตั้งค่า

```
git config --list
```



Set Default branch “main”

ตรวจสอบหลังตั้งค่า

```
pwsh in Samit
Samit ➤ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=schannel
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Samit Koyom
user.email=samit90daytalk@hotmail.com
core.pager=cat
```

git config --list --show-origin

```
Samit ➤ git config --list --show-origin
file:C:/Program Files/Git/etc/gitconfig diff.astextplain.textconv=astextplain
file:C:/Program Files/Git/etc/gitconfig filter.lfs.clean=git-lfs clean -- %f
file:C:/Program Files/Git/etc/gitconfig filter.lfs.smudge=git-lfs smudge -- %f
file:C:/Program Files/Git/etc/gitconfig filter.lfs.process=git-lfs filter-process
file:C:/Program Files/Git/etc/gitconfig filter.lfs.required=true
file:C:/Program Files/Git/etc/gitconfig http.sslbackend=schannel
file:C:/Program Files/Git/etc/gitconfig core.autocrlf=true
file:C:/Program Files/Git/etc/gitconfig core.fscache=true
file:C:/Program Files/Git/etc/gitconfig core.symlinks=false
file:C:/Program Files/Git/etc/gitconfig pull.rebase=false
file:C:/Program Files/Git/etc/gitconfig credential.helper=manager
file:C:/Program Files/Git/etc/gitconfig credential.https://dev.azure.com.usehttppath=true
file:C:/Program Files/Git/etc/gitconfig init.defaultbranch=master
file:C:/Users/Samit/.gitconfig user.name=Samit Koyom
file:C:/Users/Samit/.gitconfig user.email=samit90daytalk@hotmail.com
file:C:/Users/Samit/.gitconfig core.pager=cat
file:C:/Users/Samit/.gitconfig init.defaultbranch=main
```

Git มีการตั้งค่าได้ 3 ระดับ ซึ่งจะถูกอ่านเรียงต่อกันตามลำดับนี้ครับ:

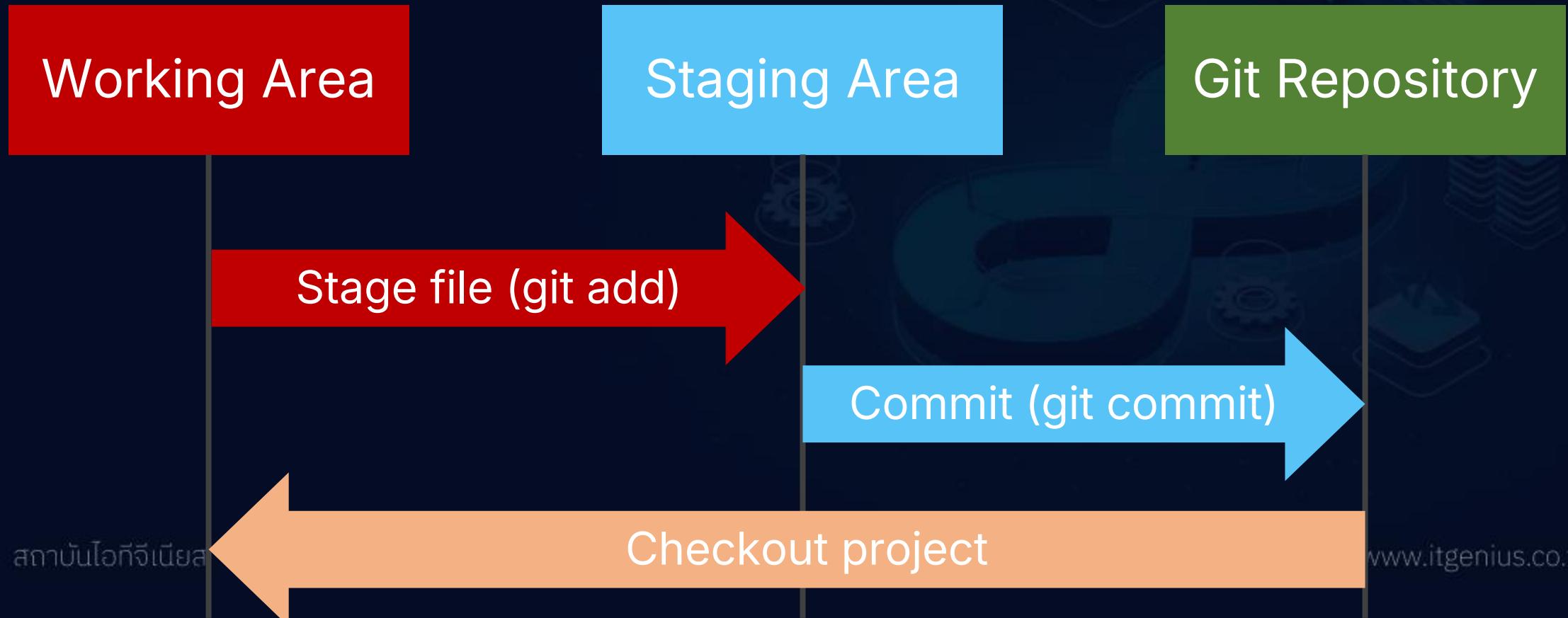
1. **System:** การตั้งค่าสำหรับทุก User บนเครื่องคอมพิวเตอร์นี้
2. **Global:** การตั้งค่าสำหรับ User ของคุณคนเดียว (ไฟล์ `~/.gitconfig`)
3. **Local:** การตั้งค่าสำหรับโปรเจกต์นั้นๆ โปรเจกต์เดียว (ไฟล์ `.git/config` ในโฟลเดอร์โปรเจกต์)

คำสั่ง `git config --list` ก็คุณใช้ จะแสดงค่าจาก ทุกระดับ ที่มีอยู่ ทำให้คุณเห็นค่าซึ่กันได้หากมีการตั้งค่าซื้อเดียวกันไว้ในไฟล์ต่างระดับกัน

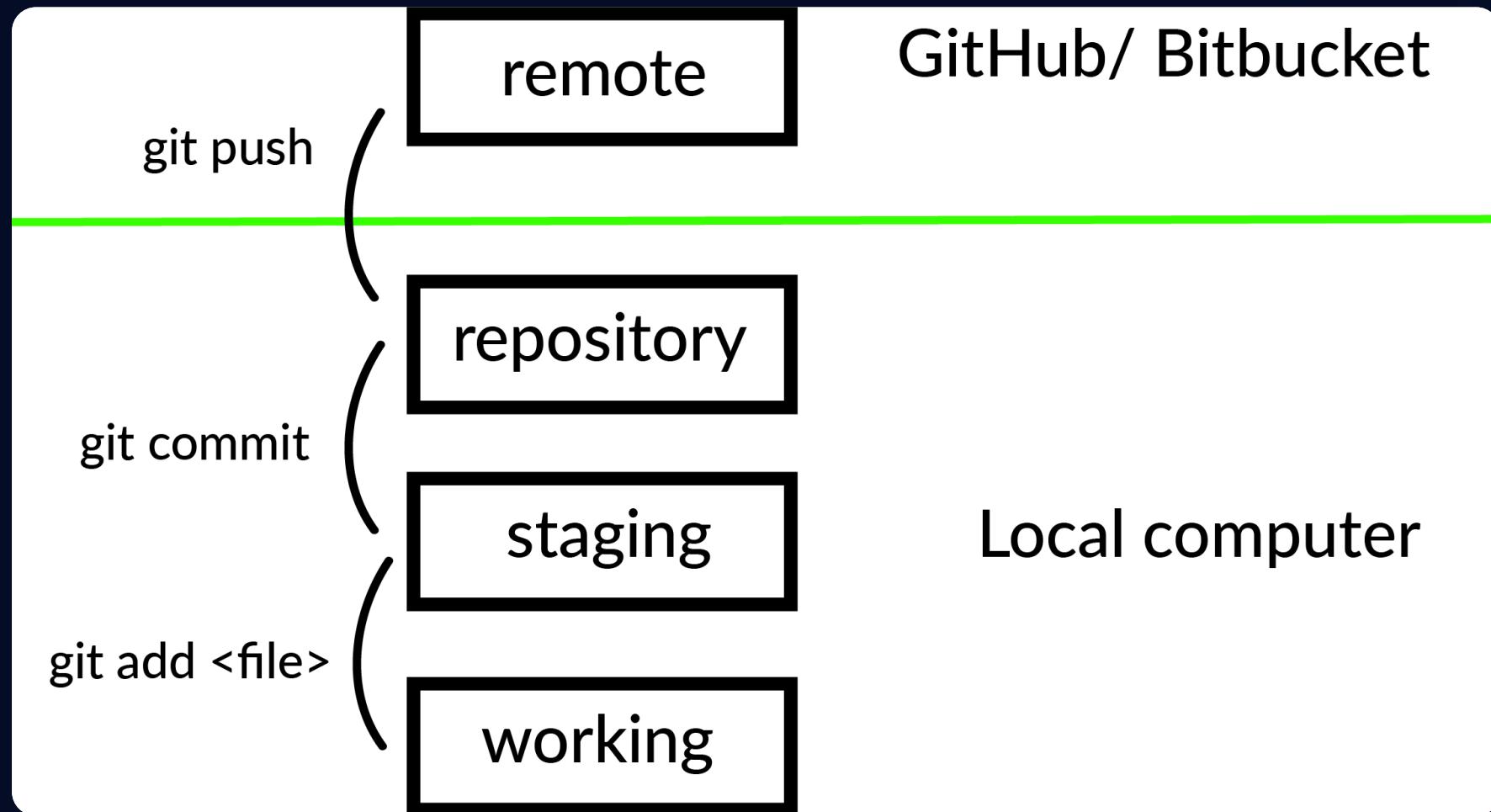


Git Workflow

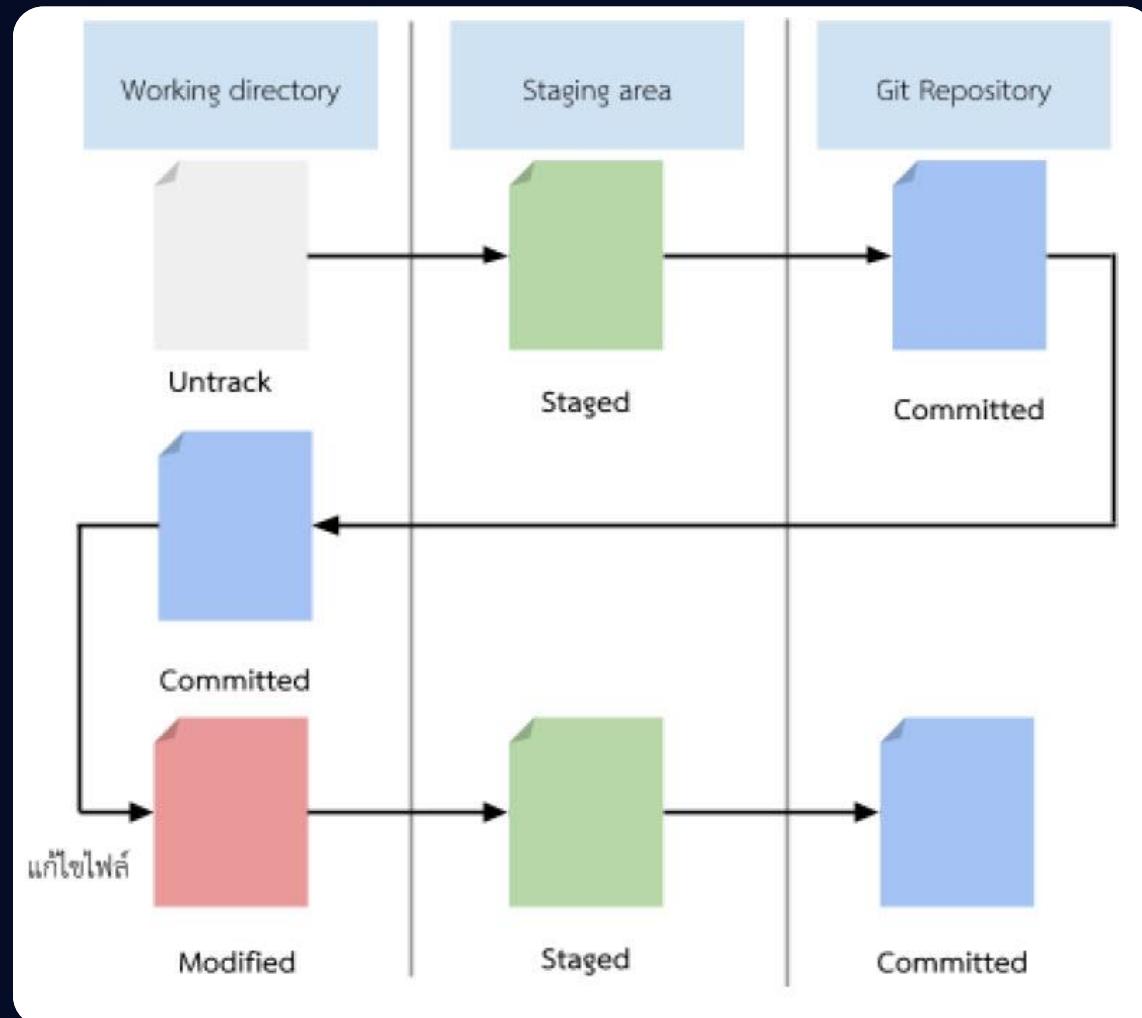
- สร้างไฟล์ใหม่
- เปลี่ยนแปลง แก้ไขไฟล์
- ลบไฟล์
- นำไฟล์ที่เปลี่ยนแปลงเข้า Staging Area
- บันทึกการเปลี่ยนแปลง ไปเก็บไว้ใน Repository อย่างถาวร



Git Work Flow



Git File Life Cycle



Example Workflow Exercise

ดูประวัติการ commit

git log

git log --oneline

เริ่มสร้างระบบ Git ในโปรเจกต์

git init

1

ดูสิ่งที่เปลี่ยนแปลง

git diff master
git diff HEAD

เพิ่ม/แก้ไขไฟล์

2

เรียกดูสถานะไฟล์

git status

3

คำสั่ง track และเพิ่มเข้า Staging area

git add

4

บันทึกประวัติเข้า Repository

git commit

5



คำสั่งพื้นฐาน Git ที่ใช้บ่อยในโปรเจกต์จริง





1. การตั้งค่าเบื้องต้น (Initial Setup)

ใช้ครั้งแรกในเครื่องหรือโปรเจกต์ใหม่

bash

Copy code

```
git config --global user.name "Samit Koyom"      # ตั้งชื่อผู้ใช้  
git config --global user.email "your@email.com" # ตั้งอีเมล  
git config --global core.editor "code --wait"    # ตั้ง VS Code เป็น editor  
git config --list                                # ตรวจสอบการตั้งค่า
```



📁 2. เริ่มต้นโปรเจกต์

bash

```
git init  
git clone <url>
```

เริ่มต้นโปรเจกต์ใหม่
คัดลอก repo จาก remote (GitHub, GitLab)

 Copy code





3. ตรวจสอบสถานะและการเปลี่ยนแปลง

bash



Copy code

```
git status          # ตรวจสอบสถานะไฟล์
git diff           # ดูรายละเอียดการเปลี่ยนแปลง
git log --oneline --graph    # ดูประวัติ commit แบบย่อและเป็นกราฟ
git show <commit_id>      # ดูรายละเอียด commit ได้
```



+ 4. จัดการไฟล์ใน staging area

bash

 Copy code

```
git add .          # เพิ่มไฟล์ทั้งหมด  
git add <file>    # เพิ่มไฟล์เฉพาะ  
git restore --staged <file> # เอาไฟล์ออกจาก staging
```



💬 5. การ commit

bash

Copy code

```
git commit -m "ข้อความ commit"          # commit พร้อมข้อความ  
git commit -am "commit โดยไม่ต้อง add" # เพิ่มและ commit ไฟล์ที่เคย track แล้ว
```





6. การเชื่อมต่อ กับ remote repository

bash

Copy code

```
git remote add origin <url>      # เชื่อมกับ remote  
git remote -v                      # แสดง remote ทั้งหมด  
git push -u origin main            # push ครึ่งแรก กำหนด branch  
git push                          # push ครึ่งต่อไป  
git pull                           # ดึงข้อมูลล่าสุดจาก remote
```



7. การจัดการ Branch

bash

 Copy code

```
git branch          # แสดง branch ทั้งหมด  
git branch <name>      # สร้าง branch ใหม่  
git switch <name>      # สลับไป branch อื่น (หรือใช้ git checkout)  
git merge <name>       # รวม branch เข้ากับ branch ปัจจุบัน  
git branch -d <name>    # ลบ branch ที่ merge แล้ว  
git push origin --delete <name> # ลบ branch บน remote
```





8. การย้อนกลับและแก้ไข

bash



Copy code

```
git restore <file>          # คืนไฟล์กลับเป็นเหมือนล่าสุด  
git reset --hard HEAD        # ย้อนกลับทุกไฟล์เป็น commit ล่าสุด  
git reset --hard <commit_id> # ย้อนกลับไป commit ที่ต้องการ  
git revert <commit_id>       # สร้าง commit ใหม่เพื่อยกเลิกการเปลี่ยนแปลง
```





9. การดูและเลือก commit

bash

Copy code

```
git log --oneline          # แสดง commit ย่อ  
git checkout <commit_id>    # ไปยัง commit นั้น (detached HEAD)  
git switch -c newbranch <commit_id> # สร้าง branch ใหม่จาก commit เก่า
```



💡 10. การ Sync โปรเจกต์

bash

```
git fetch
```

ดึงข้อมูลจาก remote โดยไม่ merge

```
git pull origin main
```

ดึงและรวมจาก remote main

```
git push origin main
```

อัปเดตชิ้น remote main

 Copy code





11. คำสั่งอื่นที่มีประโยชน์

bash

 Copy code

```
git stash                      # เก็บการเปลี่ยนแปลงชั่วคราว  
git stash pop                 # ดึงกลับมาใช้งาน  
git tag <version>            # สร้าง tag  
git push origin --tags        # ส่ง tag ขึ้น remote
```



📌 ตัวอย่าง Workflow พื้นฐาน

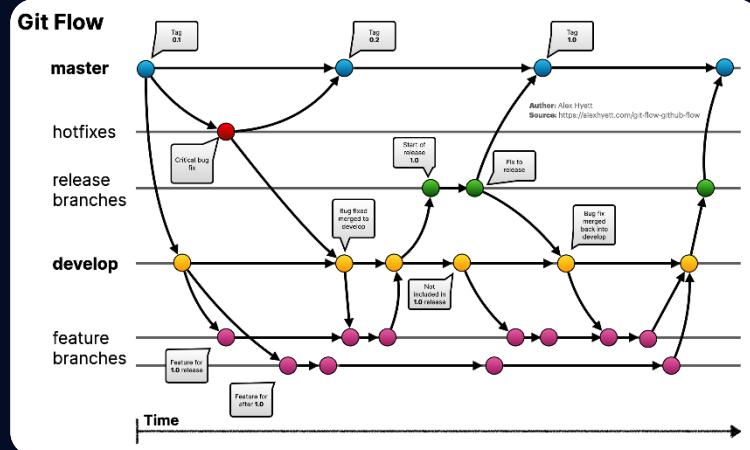
bash

 Copy code

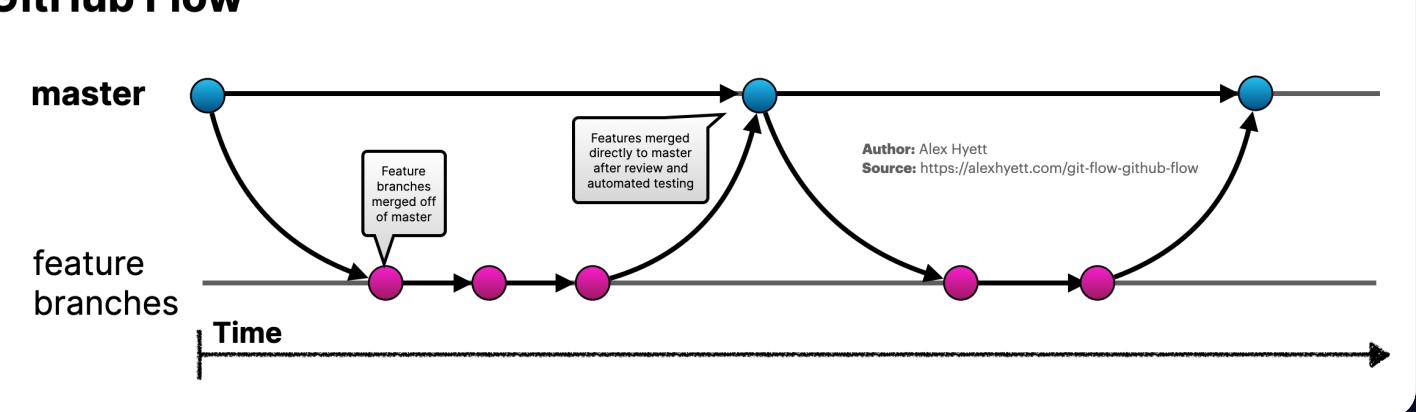
```
git init
git add .
git commit -m "init project"
git branch -M main
git remote add origin https://github.com/iamsamitdev/project.git
git push -u origin main
```



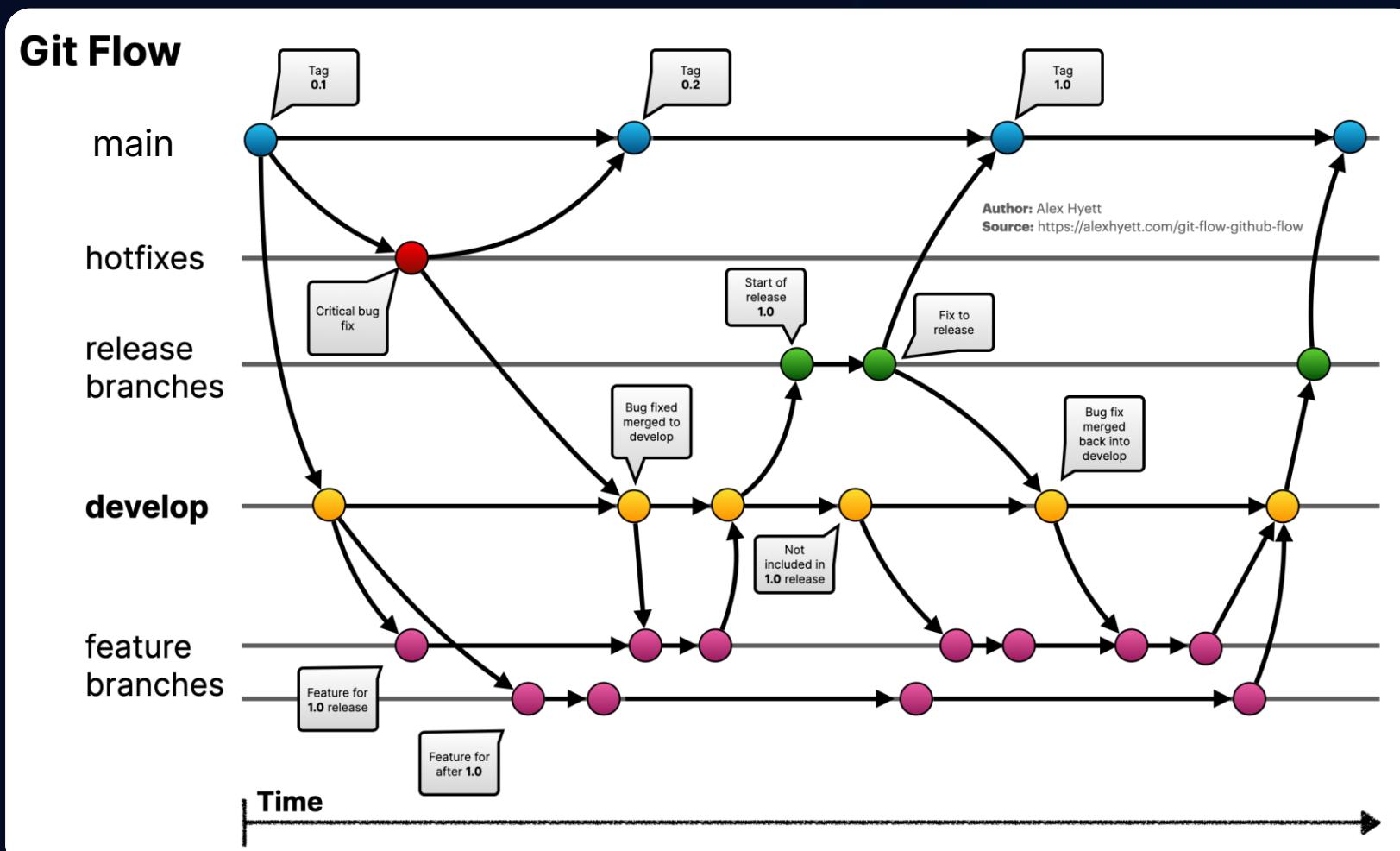
Git Flow vs GitHub Flow



GitHub Flow

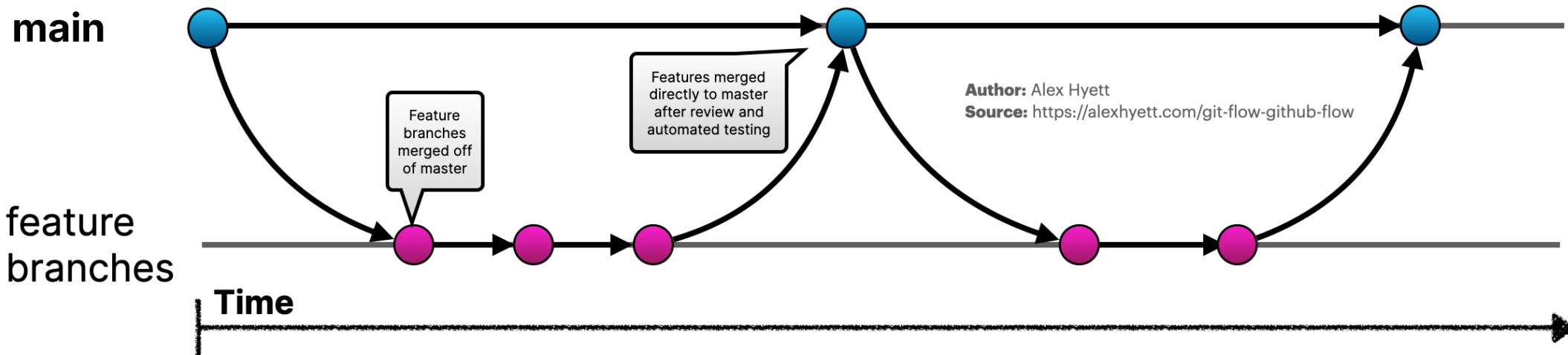


Git Flow



GitHub Flow

GitHub Flow



• LIVE

อบรมออนไลน์



ปั้นระบบอัตโนมัติอย่างมือโปรดด้วย

Jenkins และ GitHub Actions ร่วมกับ n8n



มีวิดีโอบันทึกการอบรม
ย้อนหลังให้ทุกวัน



สอนสดผ่าน Zoom
รับจำนวนจำกัด

วันที่

2



Samit Koyom
สถาบันไอทีจีเนียส

Day 2



2. พื้นฐาน Docker



Jenkins
และ GitHub
Actions
ร่วมกับ n8n





พื้นฐาน Docker

- รู้จัก Docker และแนวคิด Container
- รู้จัก Docker Images และ Containers
- การสร้าง Custom Images (Dockerfile)
- รู้จัก Docker Compose และการจัดการ

WHAT IS DOCKER?



รู้จัก Virtual Machine



Lenovo ThinkServer RD340
70AB001UUX 1U Rack Server -
1 x Intel Xeon E5-2407 v2 2.40
GHz Ram 16GB

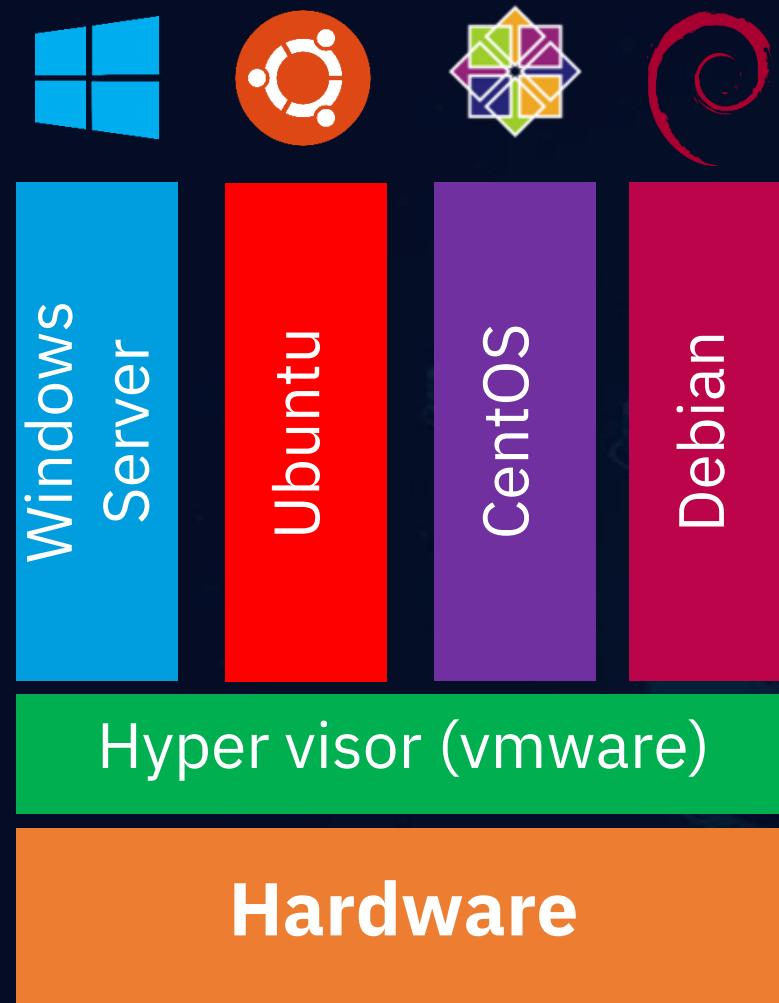
VMware ESXi

Windows Server
2 CPU
4 GB Ram

Linux Ubuntu
1 CPU
2 GB Ram



รู้จัก Virtual Machine





คือ **Software Container** ถูกออกแบบมาเพื่อสร้างสภาพแวดล้อมให้ Application ของเราทำงานร่วมกันได้บน OS เดียวกัน ช่วยให้การ Setup และจัดการ Application ต่าง ๆ ทำได้ง่ายขึ้น



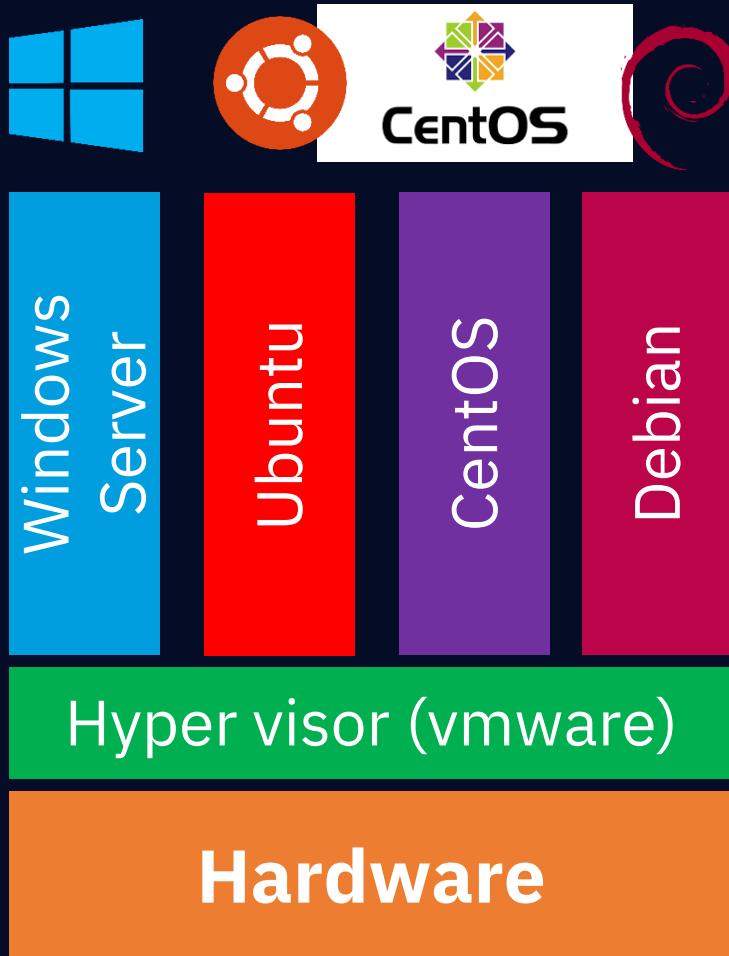
Container คืออะไร



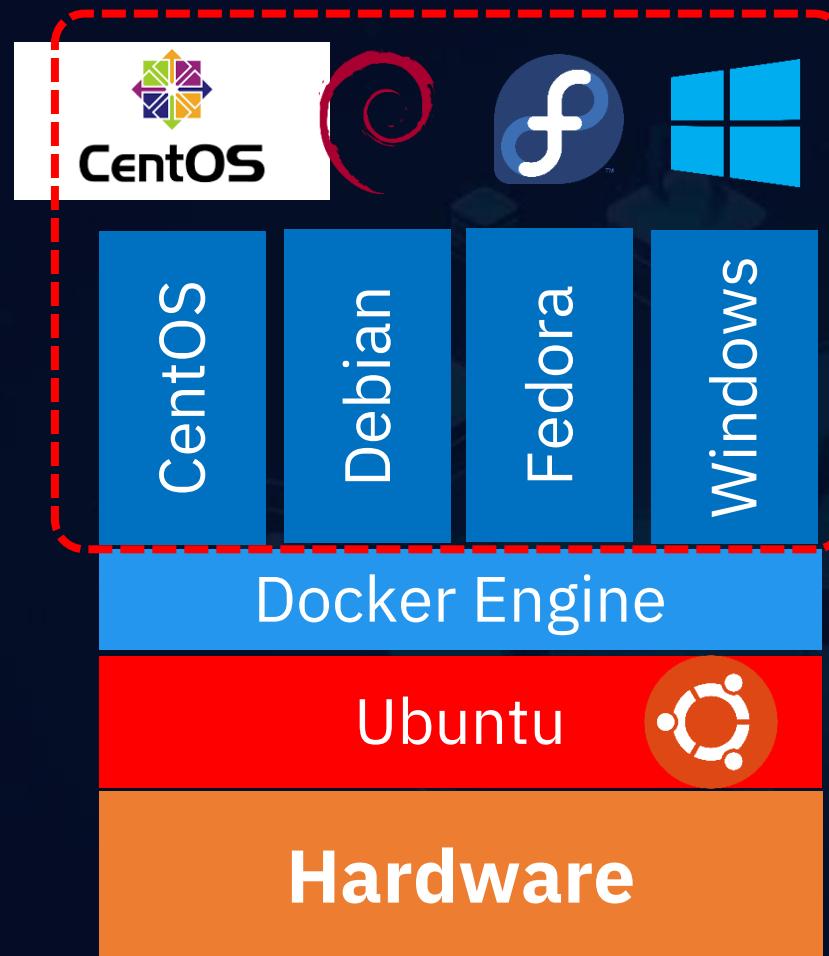
เปรียบเสมือนการ Pack รวม App/Bins/Libs ต่างๆ เป็นก้อนเดียวกัน แล้วทำการกำหนดสภาพแวดล้อมการทำงานไว้ภายในตัวมันเอง

เมื่อต้องการนำไปใช้งานที่อื่นหรือบน Production จริง ก็นำไปกั้งก้อน Container นี้เลย ทำให้สภาพแวดล้อม เมื่อกันทุกประการ

ความแตกต่างระหว่าง Virtual Machine และ Docker Container



VM



Container

- Lightweight
- Fast!
- Isolated

- Container Micro Computer
 - OS
 - CPU
 - Mem
 - Network

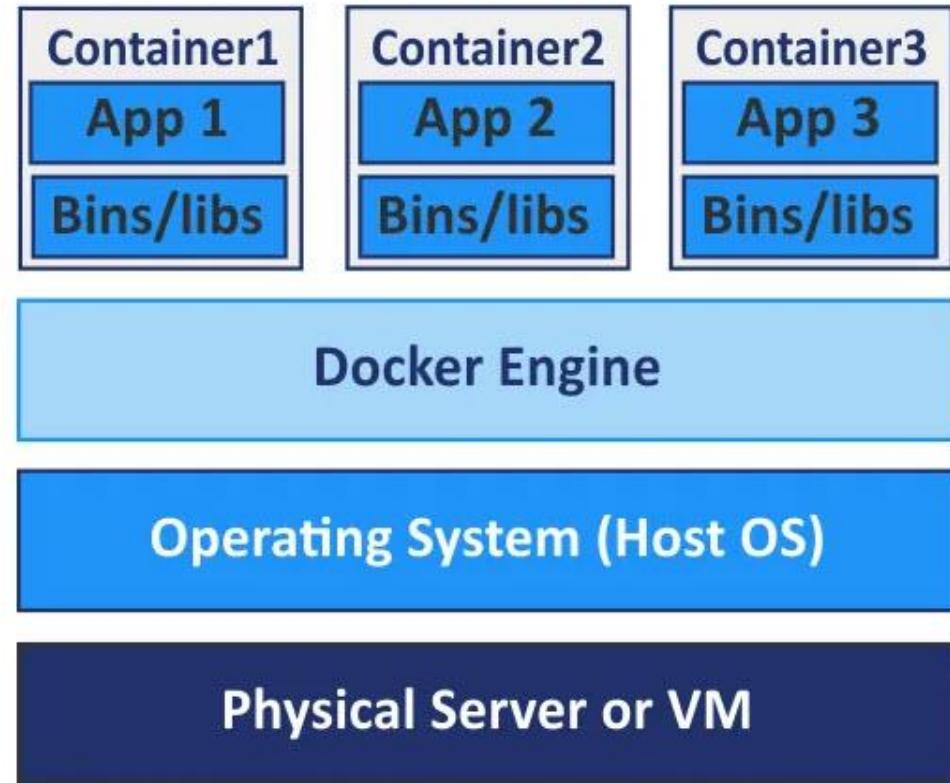


ความแตกต่างระหว่าง Virtual Machine และ Docker Container

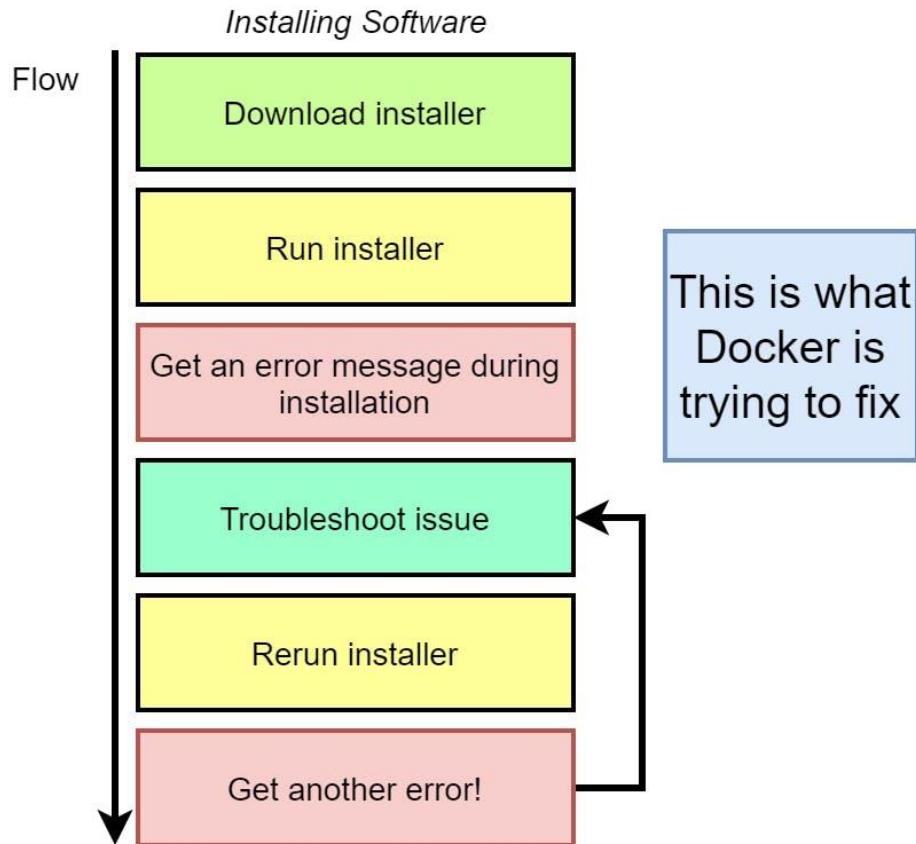
Virtual Machines



Containers

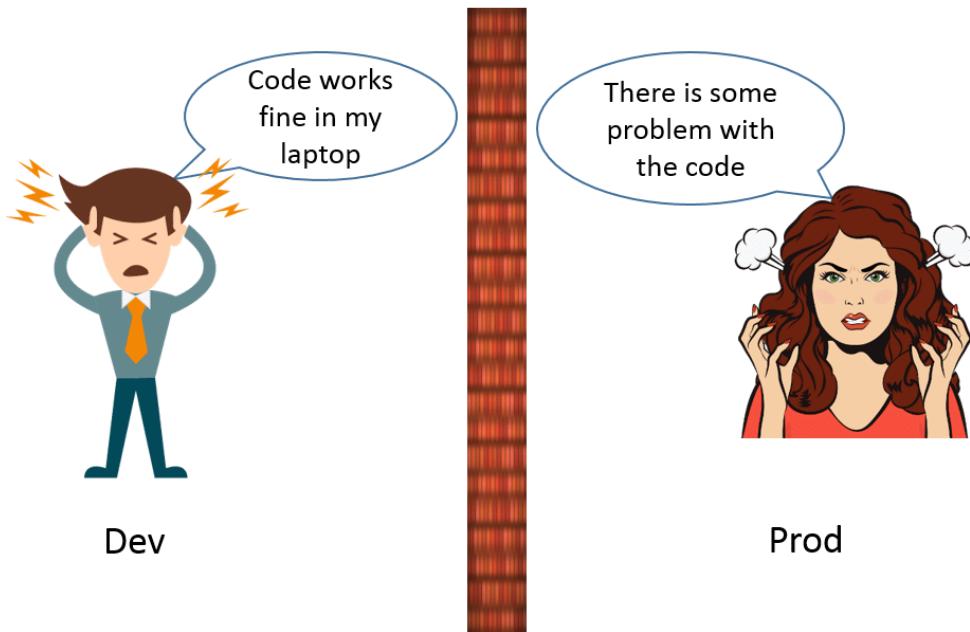


ทำไมต้องใช้ Docker และ Containers



ทำไมต้องใช้ Docker และ Containers

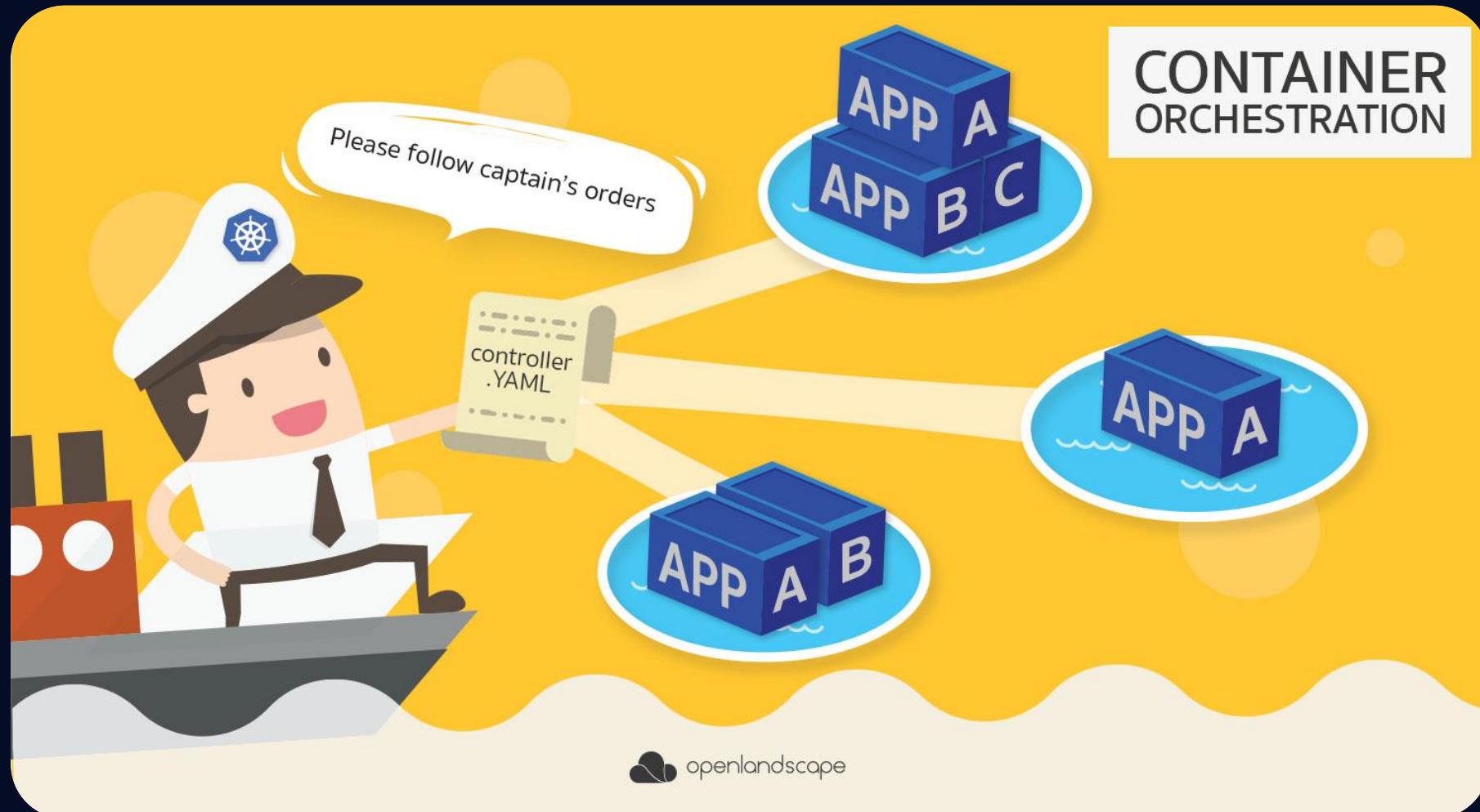
In Dev there can be a software that is upgraded and in Prod the old version of that software might be present



KEEP
CALM
IT
WORKS ON
ALL MACHINES



Kubernetes คุณางสู่การทำระบบที่ไม่มีวันล่ม



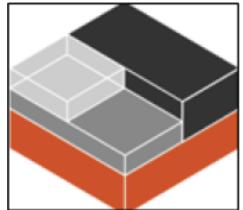
 openlandscape



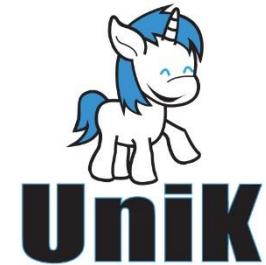
สถาบันไอทีจีเนียส

www.itgenius.co.th

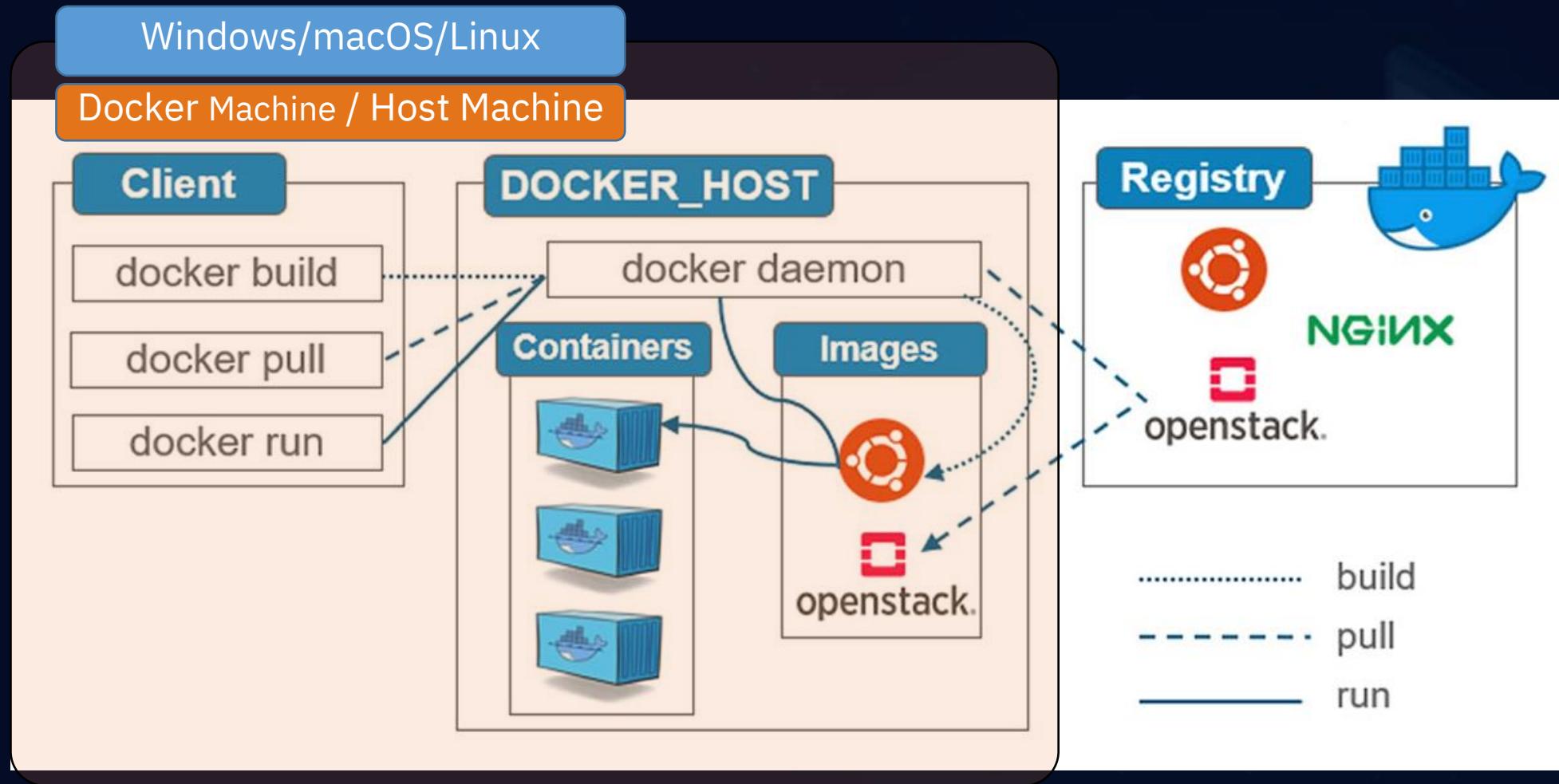
Other Software Container



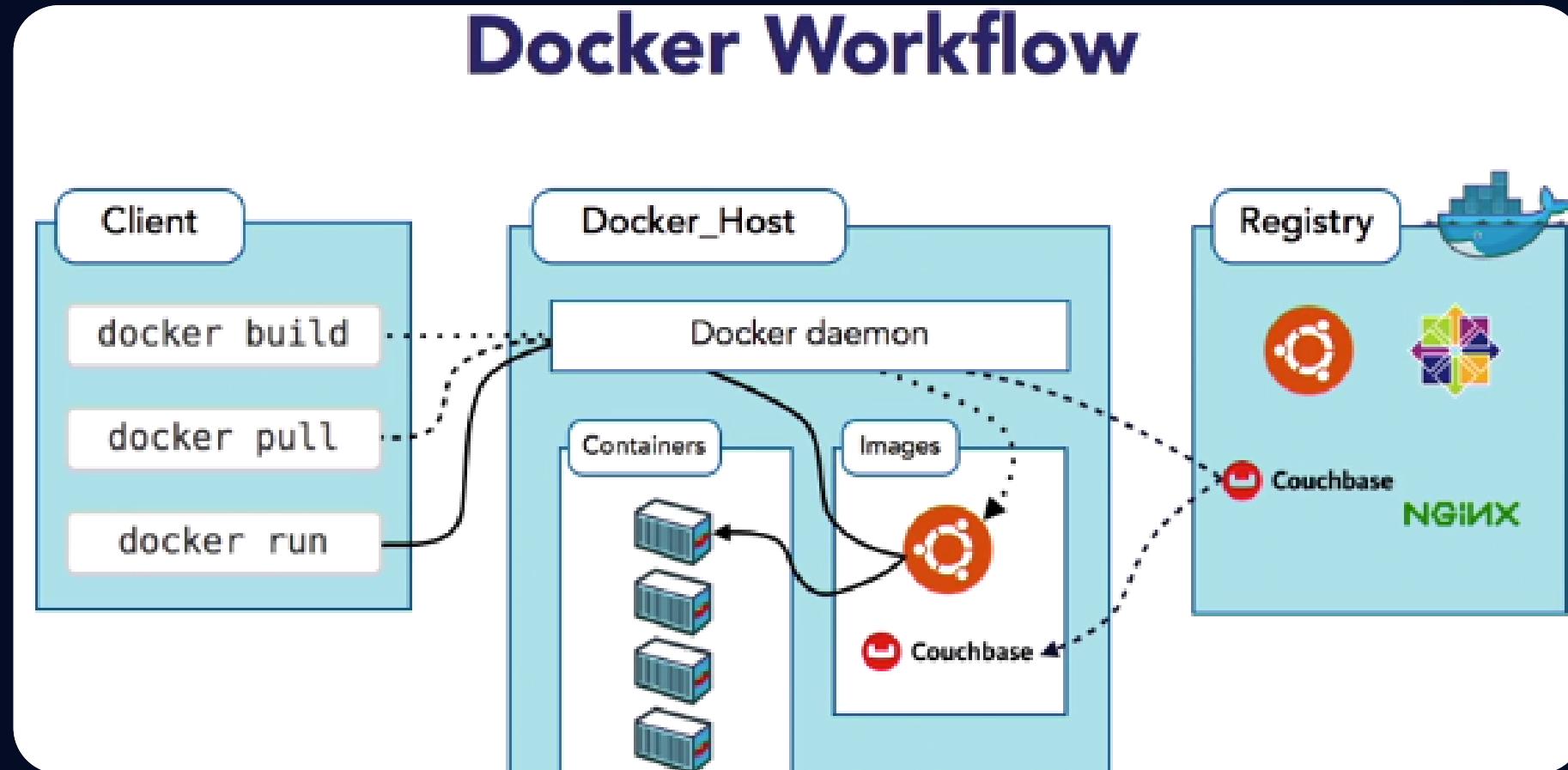
LXC (Linux)



โครงสร้างและสถาปัตยกรรมของ Docker



Workflow ในการทำงานกับ Docker



Workflow ในการทำงานกับ Docker

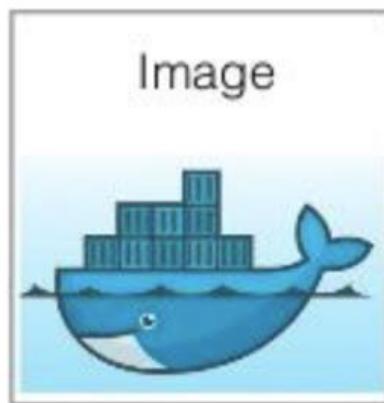


Workflow ในการทำงานกับ Docker



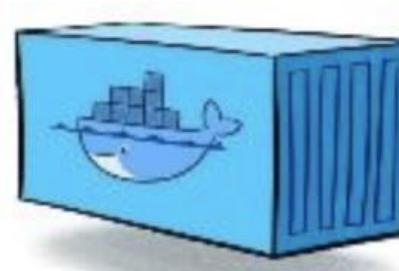
Dockerfile

build



Docker Image

run



Docker Container



Hello World

**docker**

```
docker run hello-world
```

```
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit

<https://docs.docker.com/get-started/>

```
$ docker images hello-world
```

REPOSITORY	TAG	IMAGE ID	SIZE
hello-world	latest	1b44b5a3e06a	10.07kB



```
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit

<https://docs.docker.com/get-started/>

```
$ docker images hello-world
REPOSITORY      TAG          IMAGE ID          SIZE
hello-world    latest        1b44b5a3e06a   10.07kB
```



คำสั่งพื้นฐาน Docker ที่ใช้บ่อยในโปรเจ็คต์จริง



Docker Basic Command



ลบรายการ image กึ่งหมด

```
$ docker image prune -a
```

แสดง version ของ docker

```
$ docker version
```

แสดงสถานะของ docker

```
$ docker stats
```

รายการ Images กึ่งหมด รายการ Container กึ่งหมด

```
$ docker images
```

```
$ docker image ls
```

รายการ Container ที่กำกันอยู่

```
$ docker ps
```

```
$ docker ps -a
```

ดึงรายการ image จาก registry

```
$ docker pull <imagename:tag>
```

ลบรายการ image ที่ไม่ต้องการ

```
$ docker rmi <imagename:tag>
```

Container Management



สร้าง Container

```
$ docker create <imagename|id>
```

สร้างและรัน Container (create + start)

```
$ docker run <imagename|id>
```

Start/Stop Container

```
$ docker [start|stop] <container_id>
```

Stop All Container

```
$ docker stop $(docker ps -a -q)
```

ลบรายการ Container (ที่ stop แล้ว)

```
$ docker rm <container_id>
```

ลบรายการ Container (ที่ start อยู่)

```
$ docker rm -f <container_id>
```

ลบรายการ Container ก็งหมดที่ stop อยู่

```
$ docker container prune
```



สถาบันไอทีเนยส์



1. การตรวจสอบและตั้งค่าระบบ (System Info & Setup)

คำสั่ง

```
docker --version
```

คำอธิบาย

ตรวจสอบเวอร์ชันของ Docker

```
docker info
```

แสดงข้อมูลระบบ Docker เช่น จำนวน container, image

```
docker login
```

เข้าสู่ระบบ Docker Hub

```
docker logout
```

ออกจากระบบ Docker Hub





2. การจัดการ Images (Image Management)

คำสั่ง

คำอธิบาย

`docker images`

แสดงรายการ image ทั้งหมดที่มีอยู่ในเครื่อง

`docker pull <image>`

ดาวน์โหลด image จาก Docker Hub

`docker rmi <image>`

ลบ image ออกจากเครื่อง

`docker tag <source> <target>`

เปลี่ยนชื่อหรือแท็ก image

`docker build -t <name> .`

สร้าง image จาก Dockerfile



คำแนะนำ: ก่อน deploy จริง ควรใช้ `docker image prune` เพื่อล้าง image ที่ไม่ได้ใช้





3. การจัดการ Containers (Container Lifecycle)

คำสั่ง

```
docker ps
```

```
docker ps -a
```

```
docker run <image>
```

```
docker run -d -p 8080:80 --name web nginx
```

```
docker start <container>
```

```
docker stop <container>
```

```
docker restart <container>
```

```
docker rm <container>
```

```
docker kill <container>
```

คำอธิบาย

แสดง container ที่กำลังทำงาน

แสดง container ทั้งหมด (รวมที่หยุดแล้ว)

สร้างและรัน container ในมี

รันแบบ background และกำหนดชื่อ พร้อม map port

เริ่ม container ที่หยุดไว้

หยุด container

รีสตาร์ท container

ลบ container ที่หยุดแล้ว

บังคับหยุด container





4. ตรวจสอบและเข้าใช้งาน Container (Inspection & Debug)

คำสั่ง

คำอธิบาย

```
docker logs <container>
```

ดู log ของ container

```
docker logs -f <container>
```

ดู log แบบ realtime

```
docker exec -it <container> bash
```

เข้า shell ภายใน container

```
docker inspect <container>
```

ดูรายละเอียดเต็มของ container (เช่น IP, volume)

```
docker top <container>
```

แสดง process ที่รันอยู่ใน container





5. การจัดการ Network และ Volume

คำสั่ง

```
docker network ls
```

คำอธิบาย

แสดง network ทั้งหมด

```
docker network inspect <name>
```

ดูรายละเอียด network

```
docker network create <name>
```

สร้าง network ใหม่

```
docker volume ls
```

แสดง volume ทั้งหมด

```
docker volume inspect <name>
```

ดูรายละเอียด volume

```
docker volume rm <name>
```

ลบ volume





6. การทำความสะอาดระบบ (Cleanup)

คำสั่ง

docker system prune

คำอธิบาย

ลบทุกอย่างที่ไม่ใช้งาน (container, image, volume, network)

docker image prune

ลบเฉพาะ image ที่ไม่ได้ใช้งาน

docker container prune

ลบ container ที่หยุดทำงาน

docker volume prune

ลบ volume ที่ไม่ได้ใช้งาน



คำเตือน: ควรตรวจสอบด้วย `docker ps -a` ก่อน prune เพื่อป้องกันการลบผิด



7. การใช้งาน Docker Compose (นโยบายบริการพร้อมกัน)

คำสั่ง

`docker-compose up`

คำอธิบาย

สร้างและรัน service จาก `docker-compose.yml`

`docker-compose up -d`

รันแบบ background

`docker-compose down`

หยุดและลบ service ทั้งหมด

`docker-compose ps`

แสดง service ที่กำลังทำงาน

`docker-compose logs -f`

ดู log ทั้งหมดแบบ realtime





ตัวอย่างไฟล์ docker-compose.yml

yaml

Copy code

```
version: '3'  
services:  
  web:  
    image: nginx  
    ports:  
      - "8080:80"  
  db:  
    image: postgres  
  environment:  
    POSTGRES_PASSWORD: example
```



⚡ 8. ตัวอย่าง Workflow พื้นฐาน (มือใหม่ควรรู้)

bash

 Copy code

```
# ดึง image
docker pull nginx

# สร้างและรัน container พร้อม port
docker run -d -p 8080:80 --name mynginx nginx

# ตรวจสอบสถานะ
docker ps

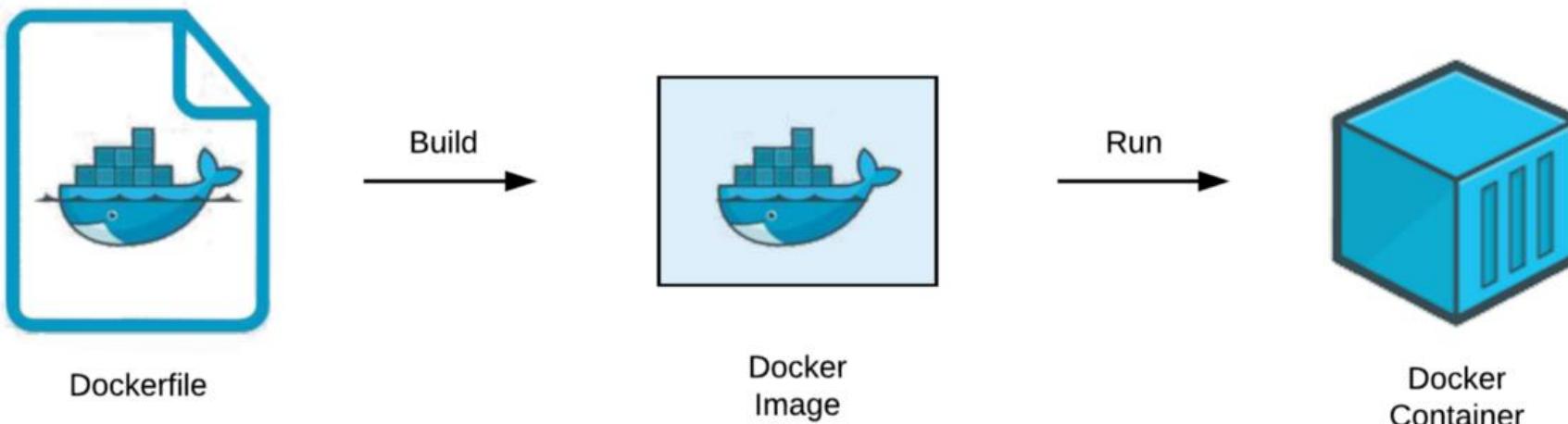
# ดู Log
docker logs mynginx

# เข้าไปใน container
docker exec -it mynginx bash

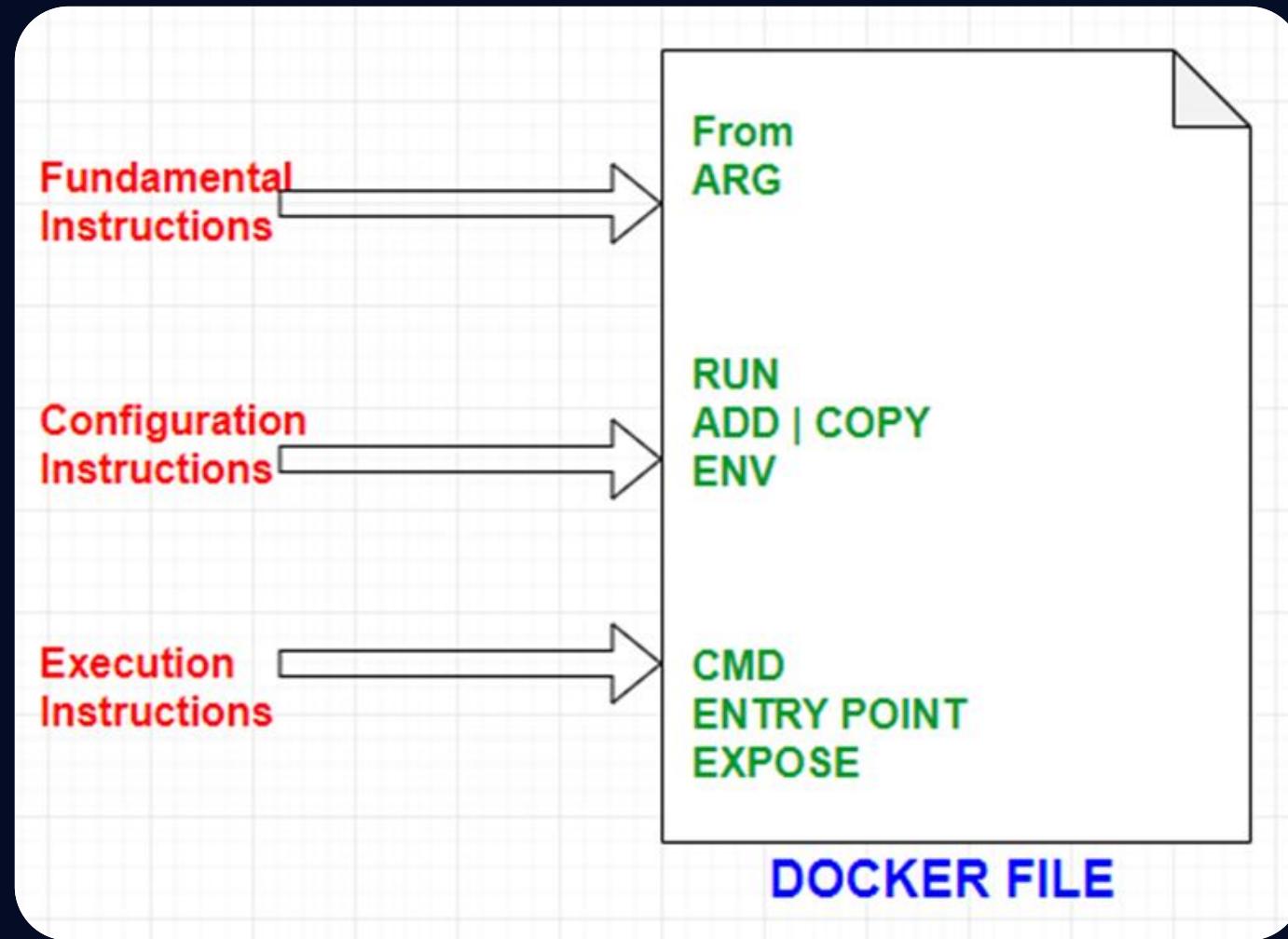
# หยุดและลบ container
docker stop mynginx
docker rm mynginx
```



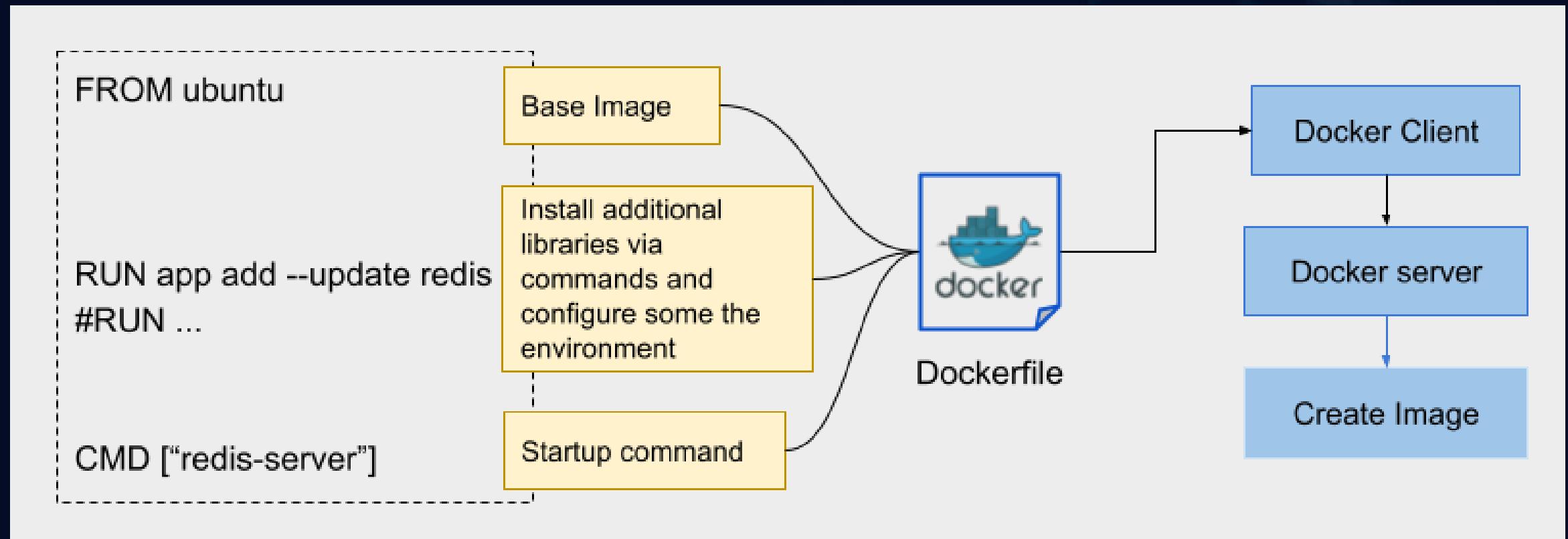
Docker File



Docker File Structure



Docker File Structure



Docker File Structure

```
FROM python:3
```

```
RUN pip3 install redis  
RUN pip3 install numpy  
RUN pip3 install pandas
```

dependencies

```
ADD executable.py /executable.py  
RUN ["chmod", "+x", "/executable.py"]
```

executable

```
ADD ProcessFileAdapterInput.py /ProcessFileAdapterInput.py  
RUN ["chmod", "+x", "/ProcessFileAdapterInput.py"]  
ADD ProcessFileAdapterOutput.py /ProcessFileAdapterOutput.py  
RUN ["chmod", "+x", "/ProcessFileAdapterOutput.py"]
```

adapter

```
ADD ProcessRedisConsumer.py /ProcessRedisConsumer.py  
RUN ["chmod", "+x", "/ProcessRedisConsumer.py"]  
ADD ProcessRedisPublisher.py /ProcessRedisPublisher.py  
RUN ["chmod", "+x", "/ProcessRedisPublisher.py"]
```

Redis communication

```
ADD ProcessRuntimeManagement.py /ProcessRuntimeManagement.py  
RUN ["chmod", "+x", "/ProcessRuntimeManagement.py"]
```

manager

```
ENTRYPOINT ["python3", "/ProcessRuntimeManagement.py"]
```



คำสั่งหลัก ๆ ของ Dockerfile

คำสั่ง	คำอธิบาย
FROM	กำหนด Base Image
LABEL	กำหนด Metadata เช่น ชื่อ version หรือเจ้าของ Image
ENV	กำหนด Environment Variable ภายใน Container
RUN	สำหรับติดตั้ง Packages ให้ Container
COPY	สำหรับคัดลอกไฟล์และโฟลเดอร์ไปยัง Container
ADD	สำหรับคัดลอกไฟล์และโฟลเดอร์ไปยัง Container โดยสามารถแตกไฟล์ .tar และคัดลอกจาก Host ภายนอกได้
CMD	สำหรับรับคำสั่งที่ต้องการขณะรัน Container
WORKDIR	กำหนด Working Directory ของ Container
ARG	กำหนด Variable ขณะสร้าง Image
ENTRYPOINT	สำหรับรับคำสั่งที่ต้องการขณะรัน Container
EXPOSE	กำหนด Port ที่เปิดให้ Container อีนติดต่อเข้ามา
VOLUME	สร้าง Folder เก็บข้อมูลแบบถาวรให้ Container



Docker File Structure

```
nodejs
  Dockerfile
  index.js
  package.json
```

Dockerfile

```
1 # Specify a base image
2 FROM node:alpine
3 WORKDIR /usr/app
4
5 # Install some dependencies
6 COPY ./package.json ./
7 RUN npm install
8 COPY ./ ./
```

```
9
```

```
10 # Default command
```

```
11 CMD ["npm", "start"]
```

```
12 |
```



Docker File Structure

Dockerfile x

```
1 # Specify a base image
2 FROM node:alpine
3 WORKDIR /usr/app
4
5 # Install some dependencies
6 COPY ./package.json .
7 RUN npm install
8 COPY ./ .
9
10 # Default command
11 CMD ["npm", "start"]
```

The diagram illustrates the build process of a Docker image. On the left, a dark gray rounded rectangle contains the Dockerfile code. An upward-pointing arrow points from this box to a vertical stack of seven rectangular cards on the right, each representing a layer in the image.

cf650ef85086	writeable container layer: docker run nodejsapp
995a21532fce	image layer: CMD ["npm", "start"]
ecf7275feff3	image layer: COPY ./ ./
334d93a151ee	image layer: RUN npm install
86c81d89b023	image layer: COPY ./package.json ./
7184cc184ef8	image layer: WORKDIR /usr/app
530c750a346e	base image: FROM node:alpine

