Basic Docker Admin

OKay Training

The Best solution for seminar training

Day 1:

Basic Concepts of Container and Docker

By Sommanut Ketpong

Introduction to Container

By Sommanut Ketpong

What is a Container?

- Container เป็นเทคโนโลยีที่ใช้สร้างหน่วยย่อย ๆ ของโปรแกรมหรือแอปพลิเคชัน ซึ่งมีทุกอย่างที่แอปพลิเคชันต้องการในการทำงาน เช่น ไฟล์ระบบ ไลบรารี และ dependencies ต่าง ๆ
- **ข้อดีของ Container**:
  - ลดความยุ่งยากในการจัดการ เพราะแต่ละ Container สามารถรันได้อย่างอิสระ
  - ใช้งานทรัพยากรระบบอย่างมีประสิทธิภาพและประหยัด
  - พกพาง่าย สามารถรันได้ทุกที่ (cross-platform)

Containers vs Virtual Machines (VMs)

- VM: มีระบบปฏิบัติการเต็มรูปแบบ, ใช้ทรัพยากรจำนวนมาก, และรันหลายแอปพลิเคชันพร้อมกันในเครื่องเดียว
- Container: รันได้บน OS หลัก (host OS), มีขนาดเล็กกว่า, ใช้ทรัพยากรน้อยกว่า และเน้นการพกพาไปใช้ได้ทุกที่
- สรุป: Container คือหน่วยย่อยที่เบากว่า VM ช่วยลดทรัพยากรที่ใช้และสะดวกต่อการจัดการมากกว่า VM

Why Container

By Sommanut Ketpong

Key Benefits of Using Containers

- Resource Efficiency:
  - Container ใช้ทรัพยากรน้อยกว่า VM ช่วยลดการใช้ CPU, Memory และ Storage ทำให้การบริหารจัดการระบบง่ายขึ้น
- Portability:
  - เนื่องจาก Container ทำงานใน environment ที่กำหนดไว้ แอปพลิเคชันใน Container จึงสามารถย้ายไปทำงานได้ทุกที่ เช่น จาก Local ไป Production
- Isolation:
  - Container แยกแอปพลิเคชันและ dependencies ออกจากกัน ทำให้มั่นใจว่าแอปจะไม่รบกวนกัน และรองรับหลายแอปพลิเคชันในระบบเดียว

Containers in Development and Testing

- Flexible Development:
  - นักพัฒนาสามารถสร้างสภาพแวดล้อมจำลองที่ใกล้เคียงกับ Production ได้ ทำให้การทดสอบแอปพลิเคชันทำได้แม่นยำและง่าย
- Consistent Environments:
  - เนื่องจาก Container มี environment แบบเดียวกันไม่ว่าจะรันที่ใด จึงทำให้มั่นใจว่าแอปพลิเคชันจะทำงานได้เหมือนกันในทุกระบบ
- Quick Setup:
  - นักพัฒนาสามารถ deploy แอปพลิเคชันใน Container ได้รวดเร็ว ไม่ต้องตั้งค่า environment หลายครั้ง

Deep Dive: Docker and Docker Compose

By Sommanut Ketpong

What is Docker

- Docker:
  - เป็นแพลตฟอร์มสำหรับสร้าง, จัดการ, และรัน Containers ซึ่งรวม dependencies ทั้งหมดของแอปพลิเคชันให้อยู่ในหน่วยเดียว ช่วยให้แอปสามารถทำงานได้ในทุกสภาพแวดล้อม
- ความสำคัญของ Docker:
  - ลดความซับซ้อนในการติดตั้งและจัดการแอปพลิเคชัน
  - ทำให้แอปพลิเคชันมีความยืดหยุ่นและสามารถ deploy ได้รวดเร็ว

Introduction to Docker Compose

- Docker Compose:
  - เป็นเครื่องมือที่ช่วยให้เราจัดการ multi-container application ได้ง่ายขึ้น โดยการกำหนดคอนฟิกทั้งหมดในไฟล์ docker-compose.yml ไฟล์เดียว
- การทำงาน:
  - Docker Compose จะช่วยตั้งค่าและรัน Containers หลายตัวพร้อมกันด้วยคำสั่งเดียว เช่น docker-compose up

Structure of docker-compose.yml

- ตัวอย่างไฟล์ docker-compose.yml:

yaml Copy code

```yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
```

Structure of docker-compose.yml

- คำอธิบาย:
  - services: ระบุ Containers ที่ต้องการใช้ เช่น web และ db
  - image: ชี้ไปที่ Docker image ที่ต้องการใช้ เช่น nginx และ mysql
  - ports และ environment: ตั้งค่าการเชื่อมต่อระหว่าง Containers

Docker Image

By Sommanut Ketpong

What is a Docker Image

- Docker Image: เป็นแม่แบบ (template) ที่เก็บทุกอย่างที่แอปพลิเคชันต้องการในการรัน เช่น โค้ด, ไลบรารี, และ dependencies ต่าง ๆ
- ลักษณะเฉพาะของ Image:
  - Images สามารถถูกสร้างขึ้น, ดึงลง, และแชร์ได้
  - เป็นไฟล์แบบ read-only และสามารถนำมาสร้าง Container ได้หลายตัว

What is a Docker Container

- Docker Container: คือ instance ของ Image ที่ถูกรันขึ้นเพื่อเป็นแอปพลิเคชันจริง ทำให้เราสามารถใช้งานแอปพลิเคชันนั้นได้
- ลักษณะสำคัญ:
  - Container ทำงานแยกเป็นเอกเทศจากระบบอื่น และสามารถสร้างได้หลายตัวจาก Image เดียว
  - มีความยืดหยุ่นสูงและสามารถหยุด ลบ หรือสร้างใหม่ได้โดยไม่กระทบกับระบบหลัก

What is a Docker Registry

- Docker Registry: เป็นแหล่งเก็บ Docker Images ซึ่งสามารถดึงลงมาสร้าง Container ได้ เช่น Docker Hub ที่เป็น Registry แบบสาธารณะ
- กระบวนการทำงาน:
  - นักพัฒนาสามารถ push (อัปโหลด) Image ขึ้น Registry และดึง (pull) Image ลงมาใช้ได้ตามต้องการ ทำให้การแบ่งปัน Image สะดวกและรวดเร็ว

Basic Docker Commands

By Sommanut Ketpong

Overview of Basic Docker Commands

- คำสั่งพื้นฐานใน Docker ที่จำเป็นในการเริ่มต้นใช้งานและจัดการ Containers
- คำสั่งหลัก:
  - docker run: สร้างและรัน Container จาก Image
  - docker ps: แสดงรายชื่อ Containers ที่กำลังรันอยู่
  - docker stop: หยุดการทำงานของ Container
  - docker rm: ลบ Container ที่หยุดทำงานแล้ว

คำสั่ง	คำอธิบาย
docker run	สร้างและเริ่ม Container จาก Image
docker ps	แสดง Containers ที่กำลังรัน
docker stop	หยุด Container ตามชื่อหรือ ID
docker rm	ลบ Container ที่หยุดการทำงานแล้ว

OKay Training
The Best solution for seminar training

Running and Managing Containers

- ตัวอย่างการใช้คำสั่งเพื่อสร้าง, รัน, และจัดการ Container:
  - สร้างและรัน Container: docker run -d --name myapp nginx
  - ตรวจสอบ Container ที่รันอยู่: docker ps
  - หยุดการทำงาน: docker stop myapp
  - ลบ Container: docker rm myapp
- คำอธิบายเพิ่มเติม: คำสั่งเหล่านี้ช่วยให้ผู้เรียนสามารถเริ่มต้นและจัดการแอปพลิเคชันใน Docker ได้ง่ายขึ้น

Hands-on: Deploying Docker Container

By Sommanut Ketpong

Steps to Deploy a Docker Container

- **คำสั่งการ deploy NGINX ใน Docker**:
  - เริ่มต้นการ deploy ด้วยคำสั่ง:

    bash

    ```
    docker run –d –name testnginx -p 8000:8000 nginx

    docker run -d --name tongapp -p 8000:8000 nginx
    ```

  - คำอธิบายคำสั่ง:
    - -d: รัน Container ในโหมด background (detached mode)
    - -p 80:80: ทำการแมปพอร์ต 80 ของเครื่องหลักกับพอร์ต 80 ของ Container
    - nginx: ชี้ไปที่ Docker Image ของ NGINX ที่จะใช้ในการรัน
- **การตรวจสอบ**: หลังจากรันคำสั่ง สามารถตรวจสอบได้โดยเปิดเบราว์เซอร์และเข้าไปที่ http://localhost

Verifying the Deployment on Browser

- การตรวจสอบผลลัพธ์:
  - เปิดเบราว์เซอร์และเข้าที่ http://localhost
  - หากการ deploy สำเร็จ หน้าตาของ NGINX welcome page จะแสดงขึ้นมา ซึ่งหมายความว่า NGINX Server ถูกติดตั้งและทำงานบน Docker อย่างถูกต้อง
- การตรวจสอบสถานะ Container:
  - ใช้คำสั่ง docker ps เพื่อตรวจสอบว่า Container กำลังรันอยู่

Introduction to Microservices

By Sommanut Ketpong

Microservices

- **Microservices คือรูปแบบการพัฒนาแอปพลิเคชันที่แบ่งระบบออกเป็น ชุดของบริการขนาดเล็ก (Small, Independent Services) ซึ่งแต่ละบริการทำงานได้อย่างอิสระ มีหน้าที่เฉพาะเจาะจง เช่น การจัดการผู้ใช้ (User Management) หรือการประมวลผลคำสั่งซื้อ (Order Processing)**
- **ลักษณะสำคัญของ Microservices:**
  - **อิสระ (Independence):**
    - **แต่ละบริการสามารถพัฒนา ทดสอบ ปรับปรุง หรือปรับขนาดได้โดยไม่กระทบกับบริการอื่น**
  - **หน้าที่เฉพาะ (Single Responsibility):**
    - **มุ่งเน้นไปที่การทำงานเฉพาะ เช่น การส่งอีเมลหรือการจัดการฐานข้อมูล**
  - **การสื่อสารระหว่างบริการ:**
    - **ใช้ API เช่น REST, gRPC หรือ Message Queue เพื่อให้บริการต่าง ๆ สื่อสารกัน**

By Sommanut Ketpong

Microservices

- ตัวอย่าง:
  - ระบบ E-commerce:
    - บริการ A: การจัดการสินค้าคงคลัง
    - บริการ B: การจัดการคำสั่งซื้อ
    - บริการ C: การชำระเงิน
  - บริการเหล่านี้สามารถทำงานแยกกันได้และปรับขยายได้ตามความต้องการ เช่น เพิ่มทรัพยากรในบริการชำระเงินในช่วงโปรโมชั่น

ข้อดีของ Microservices

- ความยืดหยุ่น: แก้ไขหรืออัปเดตส่วนใดส่วนหนึ่งได้โดยไม่กระทบส่วนอื่น
- การปรับขนาด: สามารถปรับขนาดบริการที่ต้องการได้โดยเฉพาะ
- ประสิทธิภาพทีม: ทีมพัฒนาแต่ละทีมสามารถทำงานบนบริการของตนเองได้

ข้อเสียของ Microservices

- ความซับซ้อนในการจัดการระบบ
- การสื่อสารระหว่างบริการที่เพิ่มขึ้น

ตัวอย่างการนำไปใช้งาน

- **E-commerce Platform:** มีบริการแยกต่างหากสำหรับการแสดงสินค้า การสั่งซื้อ และการชำระเงิน
- **Streaming Platform:** มีบริการสำหรับการจัดการบัญชี การสตรีม และคำแนะนำ

Cloud Native Concepts

By Sommanut Ketpong

ความหมายของ Cloud Native

- **Cloud Native หมายถึง การพัฒนาและปรับใช้แอปพลิเคชัน ที่ออกแบบมาเพื่อทำงานในสภาพแวดล้อม Cloud อย่างมีประสิทธิภาพ โดยมุ่งเน้นการใช้ Container, Microservices, และ DevOps เป็นเครื่องมือหลัก:**
  - **Container: ทำให้แอปพลิเคชันเป็นหน่วยอิสระ ง่ายต่อการปรับใช้และจัดการ**
  - **Microservices: ช่วยแบ่งระบบใหญ่ให้เป็นส่วนเล็ก ๆ ที่ยืดหยุ่น**
  - **DevOps: ช่วยเร่งกระบวนการพัฒนาและปรับปรุงระบบอย่างต่อเนื่อง**

Cloud Native Foundation (CNCF)

- CNCF (Cloud Native Computing Foundation) เป็นองค์กรที่สนับสนุนการใช้งาน **Open Source Tools** สำหรับระบบ Cloud Native:
  - **Kubernetes:** ระบบจัดการ Container แบบอัตโนมัติ
  - **Prometheus:** เครื่องมือสำหรับการ Monitoring
  - **Envoy:** Proxy สำหรับการสื่อสารระหว่าง Microservices

แนวคิดการสร้าง Application ที่รองรับการ Scaling

- Cloud Native ช่วยให้องค์กรสามารถสร้างแอปพลิเคชันที่สามารถขยายตัวและรองรับการใช้งานได้หลากหลายสถานการณ์โดยการใช้แนวคิดดังนี้:

1.Stateless Services:

1. ออกแบบบริการให้ไม่พึ่งพาข้อมูลที่อยู่ภายในตัว Container
2. ข้อมูลที่สำคัญจะถูกจัดเก็บใน **Database หรือ Persistent Storage**
   ตัวอย่าง:
   บริการ Login ที่จัดเก็บ Session ใน Redis แทนที่จะเก็บในตัว Container

2.Auto Scaling:

1. ใช้ Kubernetes หรือ Cloud Provider (เช่น AWS, Azure) ในการเพิ่มหรือลดจำนวน Container ตามปริมาณงาน
   ตัวอย่าง:
   ในช่วงที่มีผู้ใช้จำนวนมาก ระบบสามารถเพิ่ม Container เพื่อรองรับผู้ใช้งาน

3.Resilience:

1. ระบบสามารถรองรับข้อผิดพลาดและกู้คืนได้อัตโนมัติ
2. ใช้ Kubernetes เพื่อ Restart Pod ที่ล้มเหลวหรือลดผลกระทบจากปัญหา
   ตัวอย่าง:
   หาก Node หนึ่งใน Cluster มีปัญหา Kubernetes จะย้าย Pod ไปยัง Node อื่นโดยอัตโนมัติ

Analyzing Existing Applications

By Sommanut Ketpong

แนวคิดการสร้าง Application ที่รองรับการ Scaling

- **กิจกรรม:** วิเคราะห์และออกแบบแอปพลิเคชันให้เป็น Microservices
- **โจทย์:**
  - ผู้เรียนได้รับข้อมูลของแอปพลิเคชันแบบ Monolithic เช่น ระบบจองโรงแรม
  - ฟังก์ชันหลัก:
    - การจองห้องพัก
    - การจัดการผู้ใช้
    - การชำระเงิน
- **ขั้นตอน:**
  - ระบุฟังก์ชันที่สามารถแยกเป็นบริการ
  - ออกแบบแต่ละบริการให้ทำงานแบบอิสระ
  - วางแผนการสื่อสารระหว่างบริการ (เช่น REST หรือ gRPC)
  - สร้าง Diagram ของระบบ Microservices

Analyzing Existing Applications

By Sommanut Ketpong

แนวคิดการสร้าง Application ที่รองรับการ Scaling

- **กิจกรรม:** วิเคราะห์และออกแบบแอปพลิเคชันให้เป็น Microservices
- **โจทย์:**
  - ผู้เรียนได้รับข้อมูลของแอปพลิเคชันแบบ Monolithic เช่น ระบบจองโรงแรม
  - ฟังก์ชันหลัก:
    - การจองห้องพัก
    - การจัดการผู้ใช้
    - การชำระเงิน
- **ขั้นตอน:**
  - ระบุฟังก์ชันที่สามารถแยกเป็นบริการ
  - ออกแบบแต่ละบริการให้ทำงานแบบอิสระ
  - วางแผนการสื่อสารระหว่างบริการ (เช่น REST หรือ gRPC)
  - สร้าง Diagram ของระบบ Microservices

Introduction to Pod, Replica Set, Deployment, and Service

By Sommanut Ketpong

Understanding Kubernetes Architecture

- **Pod: เป็นหน่วยพื้นฐานใน Kubernetes ที่ใช้สำหรับบรรจุ Container ซึ่งสามารถมี Container หลายตัวอยู่ภายใน Pod เดียว**
  - **ลักษณะสำคัญ: Pods จะแบ่งปัน IP Address, Port Space, และ Storage**
- **Replica Set: ใช้ในการจัดการจำนวน Pod ให้มีจำนวนตามที่ต้องการ เช่น หาก Pod ล้มเหลว Replica Set จะสร้าง Pod ใหม่ขึ้นมาแทน**
  - **ฟังก์ชัน: รับประกันว่ามี Pod ทำงานอยู่จำนวนที่กำหนดเสมอ**

Deployment and Service in Kubernetes

- **Deployment: เป็นเครื่องมือที่ช่วยควบคุมการ release และ update ของ Pods โดยสามารถ rollback ไปยังเวอร์ชันก่อนหน้าได้**
  - **ฟังก์ชัน: ทำให้การอัปเดตแอปพลิเคชันทำได้อย่างราบรื่น**
- **Service: ทำหน้าที่เป็น load balancer ที่ช่วยเชื่อมต่อระหว่าง Pods และการเข้าถึงแอปพลิเคชันจากภายนอก**
  - **ประเภทของ Service: ClusterIP (เข้าถึงได้ภายใน Cluster), NodePort (เข้าถึงได้จากภายนอก)**

By Sommanut Ketpong

Introduction to Pod, Replica Set, Deployment, and Service

By Sommanut Ketpong

What is a Dockerfile?

- **Dockerfile: เป็นไฟล์ที่ใช้ในการสร้าง Docker Image โดยประกอบด้วยชุดคำสั่งที่บอกให้ Docker รู้ว่าควรทำอะไรในการสร้าง Image**
- **คำสั่งพื้นฐานใน Dockerfile:**
  - **FROM: ใช้เพื่อกำหนด Image พื้นฐานที่จะใช้สร้าง**
  - **COPY: ใช้เพื่อคัดลอกไฟล์จากเครื่องหลักไปยัง Image**
  - **RUN: ใช้สำหรับรันคำสั่งในระหว่างการสร้าง Image**
  - **CMD: ใช้กำหนดคำสั่งที่ต้องการให้รันเมื่อ Container เริ่มทำงาน**

Example of a Dockerfile

- **ตัวอย่างโค้ด Dockerfile:**

dockerfile Copy code

```dockerfile
# กำหนด image พื้นฐาน
FROM python:3.8-slim

# กำหนด working directory
WORKDIR /app

# คัดลอกไฟล์ requirements.txt
COPY requirements.txt .

# ติดตั้ง dependencies
RUN pip install --no-cache-dir -r requirements.txt

# คัดลอกโค้ดแอปพลิเคชัน
COPY . .

# กำหนดคำสั่งที่จะรันเมื่อ Container เริ่ม
CMD ["python", "app.py"]
```

Example of a Dockerfile

- คำอธิบาย:
  - **FROM python:3.8-slim: ใช้ Python 3.8-slim เป็น Image พื้นฐาน**
  - **WORKDIR /app: ตั้งค่า directory สำหรับการทำงานใน Container**
  - **COPY: คัดลอกไฟล์ไปยัง Container**
  - **RUN: ติดตั้ง dependencies**
  - **CMD: รัน app.py เมื่อ Container เริ่มทำงาน**

Introduction to Docker Compose

- **Docker Compose: เครื่องมือที่ช่วยให้เราจัดการหลาย Container ในโครงการเดียวกันด้วยไฟล์ YAML (docker-compose.yml)**
- **ความสำคัญ:**
  - **ช่วยลดความยุ่งยากในการจัดการ Containers หลายตัว**
  - **สามารถกำหนด services, networks, และ volumes ในไฟล์เดียว**

Sample docker-compose.yml

- ตัวอย่างไฟล์ **docker-compose.yml:**

yaml Copy code

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "80:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
```

Introduction to Docker Compose

- คำอธิบาย:
  - **services:** กำหนด **Containers** หลายตัว เช่น **web** และ **db**
  - **build:** สร้าง **Container** จาก **Dockerfile** ใน **directory** ปัจจุบัน
  - **ports:** ทำการแมพพอร์ตจาก **Container** สู่เครื่องหลัก
  - **environment:** กำหนดตัวแปรสภาพแวดล้อมสำหรับ **Container**

High Availability and Health Check

By Sommanut Ketpong

Understanding Health Check

- **Health Check: ฟีเจอร์ที่ใช้ในการตรวจสอบสถานะการทำงานของ Container โดยการรันคำสั่งหรือสคริปต์ที่กำหนดเพื่อเช็คว่าทำงานอย่างถูกต้องหรือไม่**
- **ทำไมต้องใช้ Health Check?:**
  - **ช่วยให้เราทราบว่า Container ยังทำงานได้ตามปกติหรือไม่**
  - **ช่วยในการจัดการปัญหาที่อาจเกิดขึ้น เช่น การรีสตาร์ท Container ที่ไม่ตอบสนอง**

Understanding Health Check

- **คำสั่ง Health Check ใน Dockerfile:**

```
HEALTHCHECK --interval=30s --timeout=10s --retries=3 CMD curl -f http://localhost/ || exit 1
```

  - --interval: เวลาที่จะรอระหว่างการตรวจสอบ
  - --timeout: เวลาที่จะรอให้คำสั่งเสร็จ
  - --retries: จำนวนครั้งที่ให้ลองก่อนที่จะระบุว่า Container ไม่พร้อมใช้งาน

Achieving High Availability with Health Checks

- **High Availability (HA): แนวทางการออกแบบระบบเพื่อให้มั่นใจว่าแอปพลิเคชันยังคงทำงานได้ตลอดเวลา แม้ว่า Container หรือ Services จะเกิดความล้มเหลว**
- **การจัดการด้วย Health Check:**
  - **เมื่อ Health Check พบว่า Container ไม่พร้อมใช้งาน จะมีการรีสตาร์ท Container โดยอัตโนมัติ**
  - **สามารถตั้งค่าให้แจ้งเตือนเมื่อ Container ไม่ตอบสนองหรือมีปัญหา**
- **ตัวอย่าง Workflow:**
  - **Container ทำงานอยู่**
  - **Health Check รันทุก ๆ 30 วินาที**
  - **หากพบว่า Container ไม่ตอบสนอง จะรีสตาร์ท Container โดยอัตโนมัติ**

Scaling with Deployment Replica

By Sommanut Ketpong

What is Scaling with Deployment Replica?

- **Scaling:** เป็นกระบวนการเพิ่มหรือลดจำนวน **Pods** ที่รันใน **Kubernetes** เพื่อรองรับการใช้งานที่มีการเปลี่ยนแปลง
- **Replica:** เป็นการสร้างสำเนาของ **Pod** เพื่อให้มีความพร้อมในการให้บริการมากขึ้น
- เหตุผลในการทำ **Scaling:**
  - รองรับปริมาณการใช้งานที่เพิ่มขึ้น
  - เพิ่มความทนทานของแอปพลิเคชันเมื่อ **Pod** ใด **Pod** หนึ่งล้มเหลว
  - ช่วยลดเวลาในการตอบสนอง **(Response Time)** ของแอปพลิเคชัน

Example of Scaling with Deployment Replica

- การเพิ่มจำนวน Replica:
- คำสั่งในการเพิ่ม Replica ใน Deployment:

bash

```
kubectl scale deployment <deployment-name> --replicas=<number>
```

- ตัวอย่าง:
  - หากต้องการเพิ่มจำนวน Replica เป็น 5:

bash

```
kubectl scale deployment my-app --replicas=5
```

- ผลลัพธ์ที่ได้:
  - Kubernetes จะสร้าง Pods เพิ่มเติมขึ้นมาให้มีจำนวนตามที่กำหนด
  - ช่วยให้แอปพลิเคชันรองรับการใช้งานที่สูงขึ้นได้อย่างมีประสิทธิภาพ

สรุปและ Q&A

By Sommanut Ketpong

Day 2: Docker Desktop and Kubernetes

By Sommanut Ketpong

Setting Up Minikube on Docker Desktop

By Sommanut Ketpong

Introduction to Minikube

- **Minikube: เครื่องมือที่ใช้ในการจำลอง Kubernetes Cluster บนเครื่อง Local เพื่อให้ผู้ใช้สามารถทดลองใช้งาน Kubernetes ได้ง่ายและรวดเร็ว**
- **คุณสมบัติหลัก:**
  - **สามารถสร้าง Cluster Kubernetes ขนาดเล็กได้ในเครื่องเดียว**
  - **รองรับการทำงานร่วมกับ Docker Desktop ทำให้ไม่จำเป็นต้องติดตั้ง VM แยกต่างหาก**
  - **ช่วยในการพัฒนาและทดสอบแอปพลิเคชันที่ใช้ Kubernetes**

Step-by-Step Installation

- **ขั้นตอนการติดตั้ง Minikube:**
  - **ตรวจสอบว่า Docker Desktop เปิดอยู่: ก่อนเริ่มติดตั้งให้แน่ใจว่า Docker Desktop เปิดใช้งานอยู่**
  - **ติดตั้ง Minikube: สามารถติดตั้งได้โดยการดาวน์โหลด Binary จาก Minikube Releases**
    - **ใช้คำสั่งสำหรับติดตั้ง:**

```bash
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

  - **เริ่มต้น Minikube: รันคำสั่งเพื่อเริ่มต้น Cluster:**

```bash
minikube start --driver=docker
```

- **ผลลัพธ์ที่คาดหวัง: หลังจากรันคำสั่งข้างต้น จะเห็นข้อความยืนยันว่าการติดตั้ง Minikube สำเร็จ**

Understanding Basic Kube-Object

By Sommanut Ketpong

Introduction to Kube-Objects

- **Kube-Objects: หน่วยพื้นฐานที่ใช้ในการจัดการ Resource ใน Kubernetes ซึ่งช่วยให้การบริหารจัดการและการทำงานของ Cluster เป็นไปได้อย่างมีประสิทธิภาพ**
- **ประเภทของ Kube-Objects พื้นฐาน:**
  - **Pod: หน่วยพื้นฐานที่บรรจุ Container อย่างน้อยหนึ่งตัว และทำงานร่วมกันในเครือข่ายเดียวกัน**
  - **Node: เครื่องที่ทำงานใน Cluster ซึ่งสามารถเป็น Virtual Machine หรือ Physical Machine และมีการรัน Pods**
  - **Service: เป็น Abstraction ที่ช่วยในการเข้าถึง Pods โดยใช้ IP และ DNS ช่วยให้สามารถเชื่อมต่อกับ Pods ได้อย่างต่อเนื่องแม้ว่า Pods จะถูกสร้างหรือทำลาย**

Relationship Between Kube-Objects

- **การเชื่อมต่อระหว่าง Kube-Objects:**
  - **Pod: ทำงานอยู่บน Node ซึ่งหมายถึงว่าแต่ละ Node สามารถมีหลาย Pods รันอยู่**
  - **Service: ทำหน้าที่เป็นตัวกลางในการเข้าถึง Pods โดยสามารถกำหนดให้ Service เชื่อมต่อกับ Pods หลาย ๆ ตัวได้**
  - **การ Load Balance: Service จะทำการกระจายการเชื่อมต่อจาก Client ไปยัง Pods ที่ทำงานอยู่ใน Node ต่าง ๆ เพื่อให้มีการทำงานที่มีประสิทธิภาพ**
- **ตัวอย่าง: หากต้องการเข้าถึงแอปพลิเคชันที่รันอยู่ใน Pods ผู้ใช้จะเชื่อมต่อผ่าน Service แทนการเชื่อมต่อกับ Pods โดยตรง**

Persistent Volume (PV) & Persistent Volume Claim (PVC)

By Sommanut Ketpong

What is Persistent Volume (PV)?

- **Persistent Volume (PV): พื้นที่เก็บข้อมูลที่ Kubernetes จัดให้ เป็น Resource แบบ Abstract ที่สร้างขึ้นโดยผู้ดูแลระบบ (Administrator) ซึ่งมีการกำหนดขนาดและประเภทของ Storage**
- **คุณสมบัติของ PV:**
  - **สามารถเก็บข้อมูลได้อย่างถาวร แม้ว่าจะมีการสร้างหรือทำลาย Pods**
  - **รองรับการเข้าถึงจากหลาย Pods ได้ (ขึ้นอยู่กับประเภทของ Storage)**
- **ประเภทของ Storage:**
  - **NFS (Network File System)**
  - **iSCSI (Internet Small Computer Systems Interface)**
  - **Cloud Storage เช่น AWS EBS, Google Persistent Disk**

What is Persistent Volume Claim (PVC)?

- **Persistent Volume Claim (PVC): คำร้องขอใช้พื้นที่เก็บข้อมูลจาก PV ซึ่งถูกสร้างขึ้นโดยผู้ใช้หรือแอปพลิเคชัน**
- **วิธีการทำงาน:**
  - **เมื่อผู้ใช้ต้องการพื้นที่เก็บข้อมูล จะสร้าง PVC เพื่อระบุความต้องการ เช่น ขนาดและประเภท**
  - **Kubernetes จะทำการค้นหา PV ที่มีคุณสมบัติตรงกับ PVC และผูก PV นั้นกับ PVC**
- **ความสำคัญ: PVC ช่วยให้ผู้ใช้สามารถขอพื้นที่เก็บข้อมูลได้โดยไม่ต้องรู้จักรายละเอียดของ PV ที่มีอยู่**

Configuring Application Environments Efficiently

By Sommanut Ketpong

Introduction to Environment Variables

- **Environment Variables:** ตัวแปรที่ใช้เก็บค่าคอนฟิกต่าง ๆ สำหรับแอปพลิเคชัน ช่วยให้สามารถเปลี่ยนแปลงค่าที่ใช้ในการทำงานของแอปพลิเคชันได้โดยไม่ต้องแก้ไขโค้ด
- ประโยชน์ของ **Environment Variables:**
  - รองรับการตั้งค่า **Config** ที่แตกต่างกันในแต่ละสภาพแวดล้อม **(development, testing, production)**
  - ป้องกันการเปิดเผยข้อมูลสำคัญ เช่น รหัสผ่าน หรือ **API Keys** ในโค้ด
  - ทำให้การพัฒนาและทดสอบแอปพลิเคชันเป็นไปได้อย่างมีประสิทธิภาพ

Configuring Environment Variables in YAML

- **การตั้งค่า Environment Variables ใน Kubernetes:**

yaml Copy code

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: app-container
    image: my-app-image
    env:
    - name: DATABASE_URL
      value: "mysql://user:password@mysql:3306/dbname"
```

Configuring Environment Variables in YAML

- **การตั้งค่า Environment Variables ใน Docker Compose:**

yaml Copy code

```yaml
version: '3'
services:
  app:
    image: my-app-image
    environment:
      - DATABASE_URL=mysql://user:password@mysql:3306/dbname
```

- **ผลลัพธ์ที่คาดหวัง: เมื่อแอปพลิเคชันทำงาน จะสามารถเข้าถึงค่าตัวแปรที่ตั้งไว้ผ่าน Environment Variables ได้**

Introduction to Kubernetes Concepts

By Sommanut Ketpong

Kubernetes Architecture and Concepts

1.1 โครงสร้างของ Kubernetes

- **Cluster: โครงสร้างหลักของ Kubernetes ที่ประกอบด้วยหลาย Node**
- **Node:**
  - **Worker Node: ทำหน้าที่รัน Container**
  - **Master Node: จัดการและควบคุมการทำงานของ Cluster**

- **Pod:**
  - **หน่วยพื้นฐานที่ Kubernetes ใช้จัดการ Container**
  - **1 Pod อาจมีหลาย Container ที่ทำงานร่วมกัน**

Kubernetes Architecture and Concepts

1.2 วิธีการทำงานของ Kubernetes

- **Control Plane:**
  - ประกอบด้วย **API Server, Scheduler, Controller Manager** และ **etcd** เพื่อควบคุม **Cluster**
- **Kubelet:**
  - **Agent** บนแต่ละ **Node** ที่ใช้สื่อสารกับ **Control Plane**
- **Service:**
  - ช่วยให้ **Container** ใน **Pod** เข้าถึงได้ทั้งภายในและภายนอก **Cluster**

How to Run Kubernetes Locally

2 การติดตั้ง Minikube และการตั้งค่า

- Minikube ช่วยให้รัน Kubernetes บนเครื่อง Local สำหรับการทดลองและพัฒนา:

1.ติดตั้ง Minikube:

1. ดาวน์โหลดและติดตั้งตามระบบปฏิบัติการ (Windows, macOS, Linux)

2.เริ่มต้น Minikube:

bash

```
minikube start
```

3.ตรวจสอบสถานะ Cluster:

bash

```
kubectl cluster-info
```

Hands-On: Running Applications on Kubernetes

3 การสร้าง Kubernetes YAML ไฟล์

- ตัวอย่าง YAML สำหรับ Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: nginx
        ports:
        - containerPort: 80
```

Hands-On: Running Applications on Kubernetes

Deploy Application ใน Kubernetes Cluster

Apply ไฟล์ YAML:

bash

```
kubectl apply -f deployment.yaml
```

ตรวจสอบสถานะ Pod:

bash

```
kubectl get pods
```

สร้าง Service เพื่อเข้าถึง Application:

bash

```
kubectl expose deployment my-app --type=NodePort --port=80
```

เข้าถึง Application ผ่าน URL:

bash

```
minikube service my-app
```


By Sommanut Ketpong

Kubernetes Tools and Commands

By Sommanut Ketpong

Using kubectl

1. คำสั่งพื้นฐานของ kubectl

- **ดูสถานะ Cluster และ Resources:**

  bash

  kubectl cluster-info kubectl get nodes kubectl get pods

- **จัดการ Pod:**

  Bash

  kubectl delete pod <pod_name> kubectl logs <pod_name> kubectl exec -it <pod_name> -- bash

การจัดการ Resources ใน Kubernetes

- ใช้คำสั่ง kubectl เพื่อ **สร้าง, อ่าน, อัปเดต, ลบ** (CRUD) Resources
- ตัวอย่าง:
  - สร้าง Deployment:

    bash
    ```
    kubectl create deployment my-app --image=nginx
    ```
  - Scale Deployment:

    bash
    ```
    kubectl scale deployment my-app --replicas=3
    ```

Declarative vs Imperative

2.1 Imperative Approach:

- ใช้คำสั่ง kubectl เพื่อจัดการ Resources ทันที
- **ตัวอย่าง:**

bash

```
kubectl run my-app --image=nginx
```

2.2 Declarative Approach:

- ใช้ไฟล์ YAML เพื่อกำหนดสถานะที่ต้องการ แล้ว Apply ด้วยคำสั่ง
- **ตัวอย่าง:**

bash

```
kubectl apply -f deployment.yaml
```

3.1 การใช้งานคำสั่ง kubectl

สร้างและลบ Pod โดยใช้คำสั่ง:

Bash

```
kubectl run demo-pod --image=busybox --restart=Never --command -- sleep 3600 kubectl delete pod demo-pod
```

3.2 สร้าง Resource แบบ Declarative และ Imperative

1.Declarative:

เขียนไฟล์ service.yaml และใช้คำสั่ง:

bash

```
kubectl apply -f service.yaml
```

2.Imperative:

1. สร้าง Service แบบตรง ๆ:

bash

```
kubectl expose deployment my-app --type=NodePort --port=80
```

Kubernetes

- **apiVersion**: กำหนดเวอร์ชันของ Kubernetes API ที่จะใช้สำหรับ Deployment ในที่นี้ใช้ apps/v1 ซึ่งเป็นเวอร์ชันที่รองรับการจัดการ Deployment ใน Kubernetes รุ่นล่าสุด**kind**: ระบุชนิดของทรัพยากรใน Kubernetes โดยในที่นี้คือ Deployment ที่ใช้ในการจัดการการปรับใช้แอปพลิเคชัน**metadata**:**name**: ชื่อของ Deployment คือ react-app, ซึ่งจะใช้ชื่อเดียวกันนี้ในการตั้งชื่อ Pods, Services ฯลฯ
- **spec**: กำหนดข้อมูลและการตั้งค่าที่เกี่ยวข้องกับ Deployment**replicas**: จำนวนของ Pods ที่ต้องการให้ Kubernetes รัน โดยในที่นี้กำหนดให้มีเพียง 1 replica (1 Pod)

Kubernetes

- **template**: กำหนด template สำหรับ Pod ที่จะถูกสร้าง
  - **metadata**: กำหนด labels ที่จะใช้กับ Pods ที่ถูกสร้างขึ้น
  - **spec**: กำหนดรายละเอียดของคอนเทนเนอร์ที่รันใน Pod:
    - **containers**: รายการคอนเทนเนอร์ใน Pod
      - **name**: ชื่อของคอนเทนเนอร์คือ react-app
      - **image**: กำหนด Image ที่จะใช้ในการสร้างคอนเทนเนอร์ ซึ่งในที่นี้คือ node:18 ซึ่งใช้ Node.js เวอร์ชัน 18
      - **ports**: ระบุพอร์ตที่คอนเทนเนอร์จะเปิดใช้ (ในที่นี้คือ 3000)
      - **volumeMounts**: ระบุการแมป volume (แหล่งข้อมูลภายใน Pod) ไปยังพาธภายในคอนเทนเนอร์ (/app)
      - **command**: คำสั่งที่คอนเทนเนอร์จะรันเมื่อเริ่มต้น ในที่นี้คือการติดตั้ง dependencies และเริ่มต้นแอป React โดยใช้คำสั่ง npm install && npm start
- **volumes**: กำหนดแหล่งข้อมูลที่ใช้ใน Pod
  - **name**: ชื่อของ volume คือ app-source
  - **emptyDir**: ใช้ volume แบบ emptyDir ซึ่งเป็น storage ชั่วคราวที่จะถูกลบเมื่อ Pod หยุดทำงาน

สรุปและ Q&A

By Sommanut Ketpong

Thank you