Introduction of Docker and Docker Compose

Class 17

24/5/2025

Acknowledgement

The series of the IT & Japanese language course is Supported by AOTS and OEC.

Ministry of Economy, Trade and Industry

Overseas Employment Corporation

What you have Learnt Last Week

We were focused on following points.

- Usage of control and loop flow statement
- Performing Linear Algebra in Numpy
- Why Requirement Analysis is so important in the process?
- Machine Learning algorithms
- Software development Life cycle
- Importance of Security compliance
- Basic Linux and its networking Commands.
- Introduction of Bash Scripting

What you will Learn Today

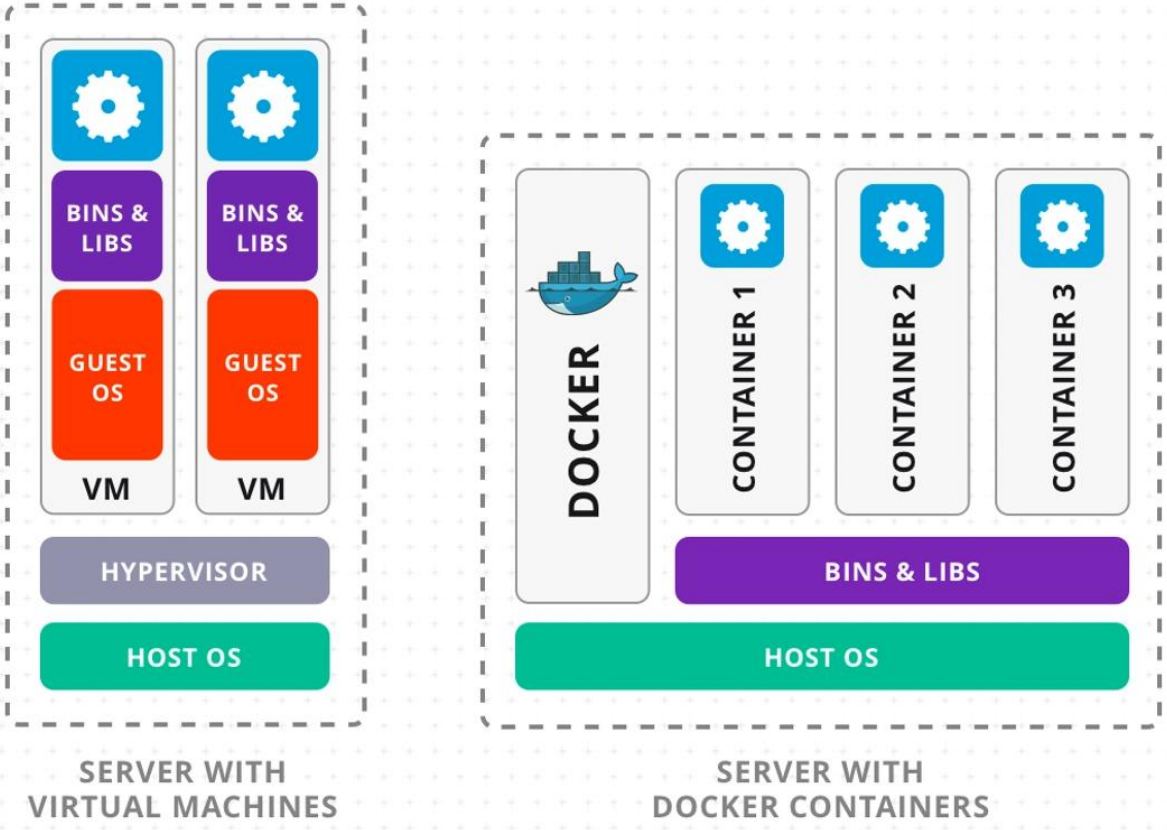
We will focus on following points.

1. Introduction of docker and docker compose
2. Key differences between containers and traditional VMs
3. Step-by-step guide to installing Docker on Linux
4. Build docker image and run docker container
5. Create docker and docker compose file using YAML
6. Quiz
7. Q&A Session

Introduction to Docker

Simplifying Software Delivery Through Containerization

- **Docker** is an open-source platform that enables developers to build, package, and run applications in lightweight containers.
- Containers include everything an app needs to run: code, runtime, libraries, and system tools.
- Docker ensures that the app runs the same across all environments – from a developer's laptop to a production server.



What is Containerization?

Lightweight and Consistent Application Execution

Containerization is a technology that packages applications and their dependencies into isolated units called containers.

- Containers share the host OS kernel
- Run consistently across environments (Dev, QA, Prod)
- Start quickly and use fewer resources than virtual machines
- Ideal for deploying microservices

Why Docker?

Benefits and Real-World Use Cases

Docker simplifies development, testing, and deployment of applications.

- Eliminates "it works on my machine" problems
- Enables fast and repeatable deployments
- Great for CI/CD pipelines
- Used in microservices, testing environments, and cloud deployments
- Easily share and reuse container images via Docker Hub

Docker Architecture Overview

Key Components of the Docker Ecosystem

- **Docker Engine:** The core service that builds and runs containers
- **Docker CLI:** Command-line tool to interact with Docker
- **Docker Daemon (dockerd):** Manages images, containers, and networks
- **Docker Hub:** Online registry for sharing container images
- **Dockerfile:** Script to define custom image builds

Key Docker Terminology

Understand the Building Blocks

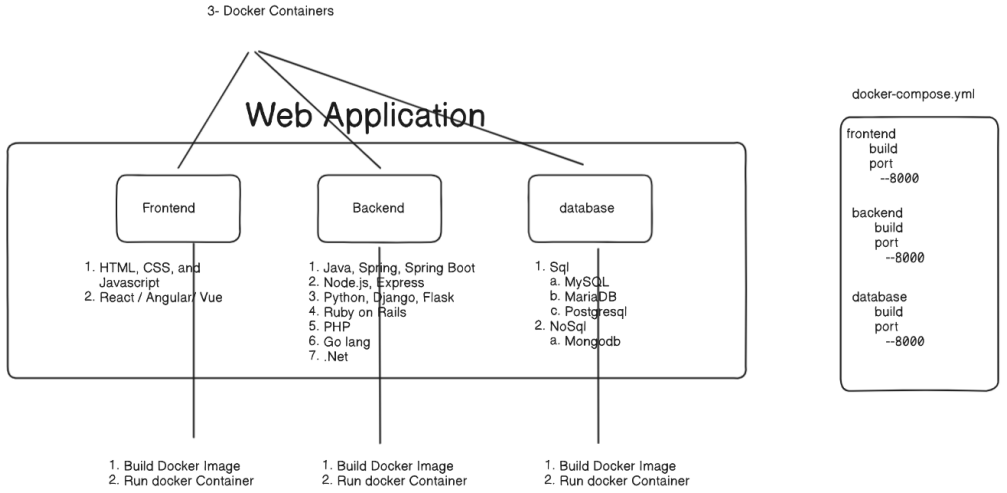
- **Image:** A read-only template used to create containers
- **Container:** A running instance of an image
- **Volume:** Stores persistent data used by containers
- **Network:** Enables communication between containers
- **Dockerfile:** A file that automates image creation

Introduction to Docker Compose

Managing Multi-Container Applications

Docker Compose allows you to define and run multi-container Docker applications using a YAML file.

- Define services, volumes, and networks in docker-compose.yml
- Run the whole stack with docker-compose up
- Useful for local development and integration testing



Example:

```
version: '3'
services:
  web:
    image: nginx
  db:
    image: mysql
```

When to Use Docker vs Docker Compose

Choosing the Right Approach

- **Docker:**
  - Best for running single-container apps
  - Quick testing of small components
  - One-off commands
- **Docker Compose:**
  - Best for managing multi-service environments
  - Useful for development and automation
  - Simplifies infrastructure as code for containers

Comparing Container and VM Architecture

Virtual Machines vs Containers – How They Run

- **Virtual Machines:** Run on a hypervisor. Each VM includes a full OS, binaries, and apps.
- **Containers:** Run on a container engine (like Docker). Share the host OS kernel.
- **Key Point:** Containers are lightweight; VMs are heavyweight due to full OS per instance.

Resource Usage and Efficiency

Memory, CPU, and Storage Usage

- **VMs:** Higher resource consumption due to full OS per instance.
- **Containers:** Share system resources efficiently; more containers can run on the same machine.
- **Example:** You can run 10–20 containers on the same system that might only handle 2–3 VMs.

Boot Time and Startup Speed

Start-up Speed – Containers Win

- **VMs:** Require several minutes to boot a full OS.
- **Containers:** Start in seconds, often instantly.
- **Why it matters:** Faster app deployment and scaling with containers.

Isolation: OS vs Hardware Level

OS-level vs Hardware-level Isolation

- **VMs:** Provide hardware-level isolation through a hypervisor.
- **Containers:** Offer OS-level isolation using Linux features (namespaces, cgroups).
- **Result:** VMs are more secure by default; containers are faster but need security best practices.

Choosing Between VMs and Containers

Where to Use VMs and Containers

Use Case	VMs	Containers
Full OS/Legacy App	✅	❌
Microservices	❌	✅
Cross-OS Development	✅	❌
Rapid Deployment & Scaling	❌	✅
Heavy Resource Apps	✅	❌

Security and Performance Factors

Pros and Cons at a Glance

- **Security:**
  - VMs have stronger isolation.
  - Containers must be hardened (image scanning, runtime protection).
- **Performance:**
  - Containers offer near-native performance.
  - VMs have overhead due to the guest OS and hypervisor.

Docker Installation

Check Basic Requirements

Before installing Docker, make sure:

- You're using a Linux system (like Ubuntu or Debian)
- You have internet access
- You can use sudo or have root access
- Remove any old Docker versions (docker, docker.io, etc.)

Install Docker

Now Install Docker on Your System

sudo apt update

sudo apt install docker.io

Docker Image can upload on Docker Hub. Docker hub is just like the Github Repository

This will install the main Docker engine and tools like Docker Compose.

Start Docker Service

Make Sure Docker is Running

sudo systemctl enable docker

sudo systemctl start docker

This will start Docker and make sure it runs automatically after reboot.

Check if Docker Works

Test the Installation

Run these commands:

docker --version

docker run hello-world

If installed correctly, you'll see a welcome message from Docker.

Use Docker Without Sudo

Make Docker Easier to Use

Add your user to the Docker group:

sudo usermod -aG docker $USER

After this, logout and log back in. Now you won’t need to type sudo with every Docker command.

Fixing Common Issues

What to Do If Something Goes Wrong

- If Docker doesn't start, restart your system

- Check status:

```
sudo systemctl status docker
```

- For permission errors, make sure you’re in the Docker group
- View logs for issues:

```
journalctl -u docker
```

What Is a Dockerfile?

Build Instructions for Docker Images

A Dockerfile is a text file with step-by-step instructions to build a Docker image.

Common Dockerfile instructions:

- **FROM** – Base image
- **RUN** – Run commands
- **COPY** – Copy files into image
- **CMD** – Start the app

Example Dockerfile:

```
FROM node:18
COPY . /app
WORKDIR /app
RUN npm install
CMD ["node", "index.js"]
```

Build Docker Image

Turn Dockerfile Into an Image

Use the docker build command to create an image:

docker build -t my-node-app .

- t gives the image a name (tag)
- means the Dockerfile is in the current folder

Check your built image:

docker images

Run a Container

Start the App from Image

Use docker run to create and run a container:

docker run -d -p 3000:3000 my-node-app

- -d runs in background
- -p maps ports: host:container

Check running containers:

docker ps

Manage Containers

Stop, Remove, and Clean

Stop a running container:

docker stop <container_id>

Remove a stopped container:

docker rm <container_id>

Remove an image:

docker rmi <image_name>

Access and Debug Containers

Run Commands Inside a Container

Enter a container’s terminal using:

docker exec -it <container_id> bash

This is helpful for debugging or exploring your app's environment.

What is YAML?

Human-readable config for Docker Compose

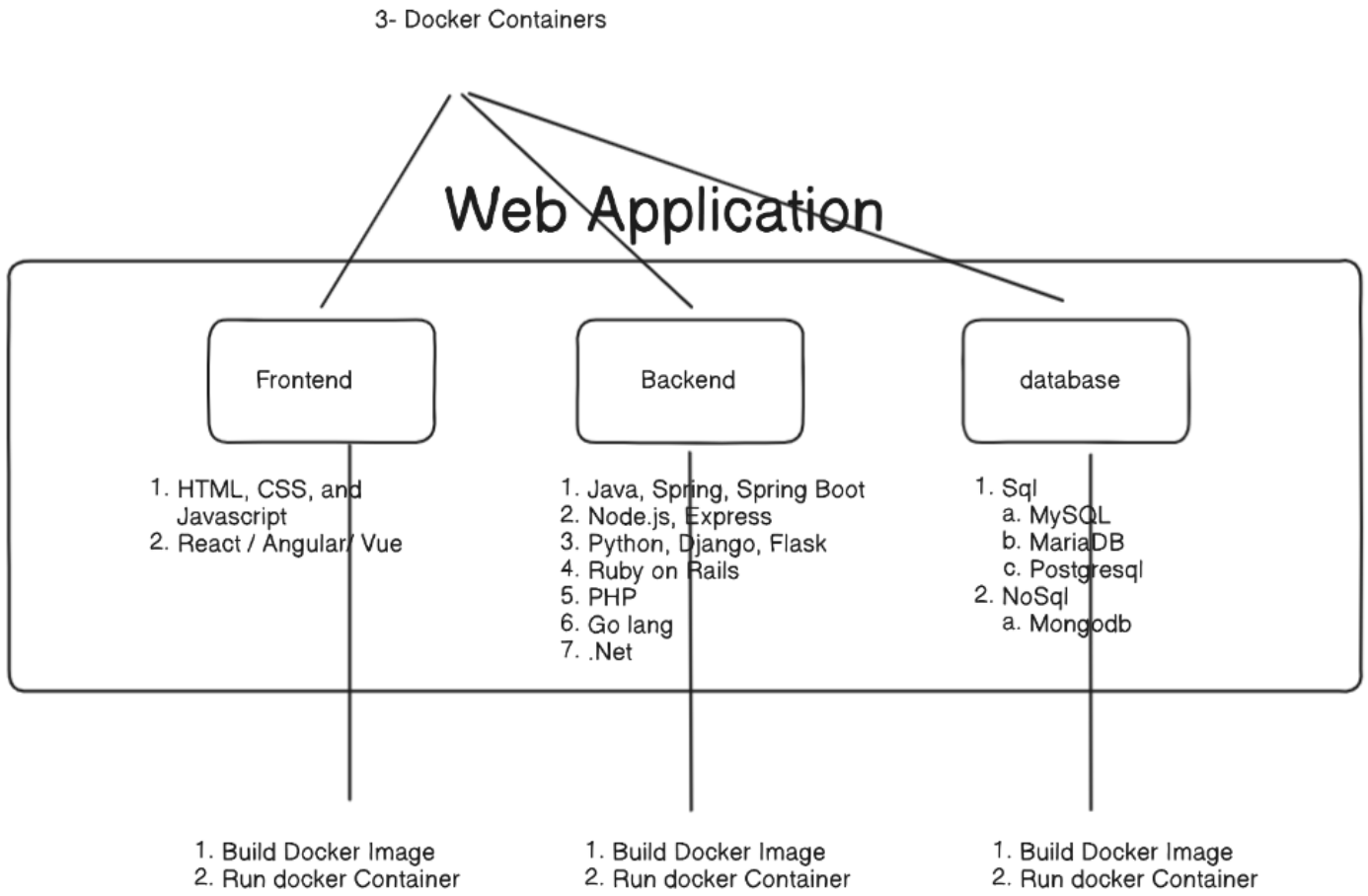
YAML (YAML Ain’t Markup Language) is a simple way to describe configuration in key-value pairs.
Used by Docker Compose to define services and settings.

Example:

```
name: myapp
version: '3'
```

Docker Compose Structure

Key Parts of docker-compose.yml



docker-compose.yml

```
frontend
    build
    port
      --8000

backend
    build
    port
      --8000

database
    build
    port
      --8000
```

Docker Compose Structure

Key Parts of docker-compose.yml

A typical Compose file has:

- version: Compose file format version
- services: App containers (e.g., web, db)
- volumes: Storage shared between host and container
- ports: Port mapping between host and container

Example:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
```

Define Services & Build

Build from Dockerfile and link files

You can build services from Dockerfile using build and context.

Example:

```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - .:/app
```

Run Docker Compose

Start, stop, and view logs

Use these commands to manage containers:

```
docker-compose up       # Start containers
docker-compose down     # Stop and remove containers
docker-compose logs     # View logs
```

Add -d to run in background:

```
docker-compose up -d
```

Assignment

Docker_Tasks

Practical Example 1

Commands

[Example 1]

1.Install docker in your AWS server

2.sudo docker pull hello-world

3.sudo docker ps

4.sudo docker images

5. sudo docker run hello-world

[Example 2]

1.Install docker in your AWS server

    1.sudo apt install docker.io

2.sudo docker pull nginx

3.sudo docker ps

4.sudo docker images

5. sudo docker run -p 5000:80 nginx

Practical Example2_Flask Application

app.py

```
from flask import Flask


app = Flask(__name__)
@app.route('/')
def home():
    return "Hello from Flask in Docker!"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

requirements.txt

```
flask
```

Create Dockerfile

Dockerfile

```
FROM python:3.11-slim

WORKDIR /app

COPY . .

RUN pip install --no-cache-dir -r requirements.txt


EXPOSE 8000

CMD ["python", "app.py"]
```

Practical Example 2

Commands

[Example 1]

1.mkdir app-python

2.cd app_python

3.Install docker in your AWS server

4.Create a dockerfile

    1.sudo vi Dockerfile

5.sudo docker build -t python_app .

6.sudo docker images

7. sudo docker run -p 8000:8000 python_app

8.sudo docker ps

[Install Requirements_Go step by step]

1.sudo apt update

2.sudo apt install python3-venv python3-pip

3.python3 -m venv app_env

4.source app_env/bin/activate

5.pip install flask

Practical Example3_Node.js Application

index.js

```
const express = require('express');

const app = express();

app.get('/', (req, res) => {

  res.send('Hello from Node.js Express!');

});

app.listen(3000, () => {

  console.log('Node app listening on port 3000');

});
```

package.json

```
{
 "name": “node-app",
 "version": "1.0.0",
 "main": "index.js",
 "scripts": {
  "start": "node index.js"
 },
 "dependencies": {
  "express": "^4.18.2"
 }
}
```

Create Dockerfile

Dockerfile

```
FROM node:18-alpine

WORDIR /app

COPY . .

RUN npm install


EXPOSE 3000

CMD ["node", "index.js"]
```

Practical Example 3

Commands

[Example 1]

1.mkdir node-app

2.cd node_app

3.Install docker in your AWS server

4.Create a dockerfile

    1.sudo vi Dockerfile

5.sudo docker build -t node_app .

6.sudo docker ps

7.sudo docker images

8. sudo docker run -p 3000:3000 node_app

[Install Requirements_Go step by step]

1.sudo apt update

2.sudo apt install python3-venv python3-pip

3.python3 -m venv app_env

4.source app_env/bin/activate

5.pip install flask

Docker Compose_Tasks

Practical Example1

docker-compose.yml

```
version: "3.8"

services:
  python-app:
    build: .
    container_name: myfirst_app
    ports:
      - “8000:8000"  # Map host port
```

Practical Example2

docker-compose.yml

```
version: "3.8"

services:
  python-app:
    build:
      context: ./app-python
    ports:
      - “8000:8000"

  node-app:
    build:
      context: ./node-app
    ports:
      - "3000:3000"
```

Run Docker Compose

Start, stop, and view logs

Use these commands to manage containers:

```
docker-compose up       # Start containers
docker-compose down     # Stop and remove containers
docker-compose logs     # View logs
```

Add -d to run in background:

```
docker-compose up -d
```

Task 1

Task 1

Create a Simple docker-compose.yml

Define a basic Docker Compose file.

Instructions:

- Create a docker-compose.yml file.
- Add version '3'.
- Define a service web using the nginx image.
- Expose port 8080 on the host and map it to 80 in the container.

Yaml

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
```

Task 2

Task 2

Create Multi-container App (Web + DB)

Instructions:

Create docker-compose.yml:

Yaml

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD:
root123
```

Quiz Section

Quiz

Everyone student should click on submit button before time ends otherwise MCQs will not be submitted

[Guidelines of MCQs]

1. There are 20 MCQs
2. Time duration will be 10 minutes
3. This link will be share on 12:25pm (Pakistan time)
4. MCQs will start from 12:30pm (Pakistan time)
5. This is exact time and this will not change
6. Everyone student should click on submit button otherwise MCQs will not be submitted after time will finish
7. Every student should submit Github profile and LinkedIn post link for every class. It include in your performance

Assignment

Assignment should be submit before the next class

[Assignments Requirements]

1. Create a post of today's lecture and post on LinkedIn.
2. Make sure to tag @Plus W @Pak-Japan Centre and instructors LinkedIn profile
3. Upload your code of assignment and lecture on GitHub and share your GitHub profile in respective your region group WhatsApp group
4. If you have any query regarding assignment, please share on your region WhatsApp group.
5. Students who already done assignment, please support other students

Q&A Session

ありがとうございます。
Thank you.
شکریا

For the World with Diverse Individualities